An Introduction to Clustering

Building a 2-Node Cluster That Solves More Problems Than it Introduces

Author:

Madison Kelly, mkelly@alteeve.com

Date:

May 11, 2010

Audience:

GTALUG; Greater Toronto Area Linux User's Group

Table of Contents

1 Welcome

Clustering is, perhaps, the least understood technology in IT.

The equation of “Cluster = Performance” is one of the biggest misconceptions. It's true that clustering is often critical in high performance applications. That said though, it's also one of the hardest goals to achieve unless an application has been specifically designed to take advantage of clustering.

There are two other major problems that clustering solves; Reliability and Scalability. These will be the focus of this talk.

1.1 Glossary

This talk will use some terms and acronyms that may be new to you:

dom0	In Xen virtualization environments, “dom0” refers to the first virtual machine. It has special access to the underlying hardware along with a special view of networking and other applications used in the Xen environment.
domU	In Xen virtualization environments, “domU” refers to any virtual machine other than “dom0”. These VMs have limited access to the underlying hardware and do not have direct access to the Xen architecture.
DRBD	This is an acronym for “Distributed Replicating Block Device”; the program used to ensure that data is consistent on both nodes at all times.
Fencing	In a cluster with shared storage, a “fence” is a method of disabling or disconnecting a defective node from the cluster and it's shared resources.
IPMI	This is an acronym for “Intelligent Platform Management Interface”, a very common method of implementing a fence on higher-end cluster nodes. Various OEMs implement IPMI under their own brands, like Dell's DRAC, HP's iLo, etc.
LVM	This is an acronym for “Logical Volume Management”. It is a type of block device manager that allows for highly customizable management of physical block devices and partitions.
Node	A “node” is simply a computer participating in a cluster.
Quorum	In clusters with more than three nodes, “quorum” is achieved when more than half of the maximum nodes can communicate with each other. A cluster, or cluster segment, can only start up when it has quorum. Any nodes in a segment without quorum will be fenced.
VM	This is an acronym for “Virtual Machine” and can refer to any “machine” that exists on top of virtual hardware.

2 Clustering; What is it?

At it's most basic meaning, a “Cluster” is nothing more than two or more computers working together to accomplish a common task. Beyond that, a “cluster” can be anything the implementer designs it to be.

In many cases, this means customizing a given application to (effectively) spread out the among the cluster's nodes in some way best suited to that given application. When an application is not cluster-aware, it may involve clustering a component of the system below the application, such as a clustered Virtual Server on top of a DRBD partition using LVM.

For these reasons, clusters are very rarely consistent with one another.

2.1 What You Should Know

Clustering is an advanced system's administration topic. It is not possible to cover all steps and all technologies encountered in a paper like this. To do so would require an entire book. Where possible, details will be explained, but you should be familiar with the following:

- Linux command line
- Networking
- LVM; Logical Volume Management software
- Operating System installation; Specifically, CentOS.

2.2 Our Cluster

For every problem, there is a cluster. It would be near impossible to discuss all types, so this paper will focus on one type only;

A high-availability, 2-node cluster with a shared file system.

Even with a description this specific, there are a myriad of ways that you could go about building your cluster to solve this problem. So to further refine; we will be using Red Hat clustering software to host a Xen virtual machine that will exist on both nodes via a shared DRBD block device running on LVM logical volumes hosting a shared GFS file system mounted on both nodes.

Xen is used on the nodes to cover some “gotchas” particular to Xen. Hosting floating virtual servers is not discussed in detail, but this paper should lay all the ground work you will need to implement such a setup. Similarly, You could fairly easily adapt this paper to host a iSCSI host with an IP address managed by the cluster.

3 Let's Get Building!

The first thing to choose is our hardware.

3.1 Hardware

Exactly what hardware you will use depends largely on your requirements. There are a few features you will want though:

- Virtualization-enabled CPU with 2 or more cores.
- Sufficient RAM for your dom0 and all your domU VMs.
- Three NICs, at least two of which should be gigabit.
- Some form of fencing device.

If you plan to run multiple virtual machines on your cluster, each node should have sufficient resources to host all VMs. This is to ensure that all VMs can run in a failed state. What this translates to is an exercise for the reader. If you plan to implement iSCSI on your cluster, you should pay particular attention to the latency introduced by DRBD.

3.2 Networking

Each node will need three network cards. These will connect to the following three subnets:

- Internet polluted subnet
- DRBD subnet
- “Back Channel” subnet

3.2.1 Internet Polluted Subnet

This subnet will be the Internet-facing subnet. As such, it must be treated as highly insecure and no cluster management devices or software should be visible or accessible here.

In this paper, each node's '`eth0`' will connect to this subnet. The subnet itself will use the range '`192.168.1.0/24`'.

3.2.2 DRBD Subnet

DRBD pushes disk writes over this subnet. DRBD will not report to the OS that the write is successful until the data has been successfully written to the disks on all nodes. For this reason, lag on this subnet will have a direct and noticeable impact on your cluster's performance. Thus, we want to have a dedicated connection on all nodes to this subnet. Your fastest, lowest-latency networking hardware should be used with this subnet. In two node clusters, a simple cross-over cable between each node's ethernet device will provide good performance.

In this paper, each node's 'eth1' will connect to this subnet. The subnet itself will use the range '10.0.0.0/24'.

3.2.3 "Back Channel" Subnet

The "Back Channel" subnet is where all cluster control traffic will exist. As such, this subnet should be restricted to connections from devices directly involved in cluster management.

If your node has an IPMI device that runs on top of a NICs, that NIC should be the one connected to this subnet. If you use external fence devices, they should also be connected to this subnet.

In this paper, each node's 'eth2' will connect to this subnet. The subnet itself will use the range '10.0.1.0/24'.

4 Software

This paper will use CentOS 5.4, x86_64. Any 5-series of CentOS, Red Hat or other RHEL compatible distribution on any architecture should work fine.

4.1 Operating System Install

Start with a basic install of the OS. The only important package groups to install are the 'virtualization' and 'clustering' groups.

When you get to the file system configuration section, be sure to leave enough unpartitioned space for you DRBD partition. In practice, it is best to create the smallest partition you can get away with as your dom0's 'root' and 'boot' partitions. How you interpret this is an exercise for the reader. In general, I usually use this setup for each node:

1. Two hard drives:
    1. Create a 200 MiB partition on '/dev/sd[ab]0' as the RAID 1 device '/dev/md0'. Format this partition as 'ext3' directly and mount at '/boot'.
    2. Use the remaining space to create partitions on '/dev/sd[ab]1' as the RAID 1 device '/dev/md1'. Format this partition as an LVM physical volume.
    3. Create an LVM logical volume called '/dev/an-lvmX' (where X matches the node number).
    4. Create a 2 GiB logical volume called '/dev/an-lvmX/lv00' and format it to be a swap partition.
    5. Create a 20 GiB logical volume called '/dev/an-lvmX/lv01' and format it to be dom0's root partition.
    6. Leave the remaining VG space alone for later use.
2. One hard drive:
    1. Repeat the above LVM setup using just '/dev/sda'.

If you would like to use the kickstart files used in AN!Cluster as reference, please see:

– http://wiki.alteeve.com/index.php/2-Node_CentOS5_Cluster#OS_Install

4.2 Post OS Install

Once the OS is installed, we'll want to make a few changes.

- Setup Networking
- Limit dom0's memory
- Change the default run-level
- Change when xend starts

4.2.1 Network Setup

There are a few initial changes to your nodes' networking that you may want to make:

- Update the NIC to ethX mapping
- Modify the hosts file
- Disable iptables

4.2.1.1 NIC to ethX

More often then not, you will want to shuffle around which NICs are assigned to which 'ethX' device. The easiest way to do this is to identify each NIC's MAC address, stop the networking service, change the 'HWADDR=x' entries in the appropriate 'ifcfg-ethX' files and restart networking.

If you want a complete walk-through on making these changes, please see:

– http://wiki.alteeve.com/index.php/Changing_the_ethX_to_Ethernet_Device_Mapping_in_Red_Hat/CentOS

4.2.1.2 hosts file

Once you've setup your NICs and assigned your IPs, add an entry for your nodes in each node's '/etc/hosts' file using the back-channel subnet. As an example, I like to use the following addresses:

- an-node01
  - eth0: 192.168.1.71
  - eth1: 10.0.0.71
  - eth2: 10.0.1.71
- an-node02
  - eth0: 192.168.1.72
  - eth1: 10.0.0.72
  - eth2: 10.0.1.72

Before editing '/etc/hosts', run 'uname -n' and remove any pre-existing entries with the returned name. This is almost always the case, and we don't want calls to that name to resolve to localhost. Given these addresses, I add the following to my '/etc/hosts' file:

```
10.0.1.71   an-node01 an-node01.alteeve.com
10.0.1.72   an-node02 an-node02.alteeve.com
```

A note on host names.

If you plan to use Red Hat's 'luci' cluster manager, do not use '_' in your hostnames.

4.2.1.3 Disable iptables

Be sure to flush netfilter tables and disable iptables and ip6tables from starting on your nodes. This is because the 'dom0' servers will not be connected directly to the Internet and we want to minimize the chance of an errant iptables rule messing up our configuration. If, before launch, you wish to implement a firewall, feel free to do so but be sure to thoroughly test your cluster to ensure no problems were introduced.

```
chkconfig --level 2345 iptables off
/etc/init.d/iptables stop
chkconfig --level 2345 ip6tables off
/etc/init.d/ip6tables stop
```

4.2.2 Limit dom0's Memory

This step is only important if you plan to implement float virtual machines.

Normally, dom0 will claim and use memory not allocated to virtual machines. This can cause trouble if, for example, you've moved a VM off of one node and then want to move it, or another, VM back. This is because, for a period of time, dom0 will claim that there is not enough free memory for the migration. By setting a hard limit of dom0's memory usage, this scenario won't happen and you will not need to delay migrations.

To do this, add 'dom0_mem=512M' to the Xen kernel image's first module line in grub. For example, you should have an entry like this in your grub configuration file:

```
title CentOS (2.6.18-164.15.1.el5)
       root (hd0,0)
       kernel /vmlinuz-2.6.18-164.15.1.el5 ro root=/dev/an-lvm01/lv01 rhgb quiet dom0_mem=512M
       initrd /initrd-2.6.18-164.15.1.el5.img
```

You can change the '512M' with the amount of RAM you want to allocate to dom0.

4.2.3 Change the default run-level

This is purely a performance-enhancing option.

Most clusters are headless, and as such, it's often a waste of resources to have a graphical interface running when the node boots. To free up these resources, you can change the default run level to 3. If you are working directly on your cluster nodes, you may wish to delay or skip this step.

To do this, edit '`/etc/inittab`', change the '`id:5:initdefault:`' line to '`id:3:initdefault:`' and then switch to run level 3 via '`init 3`'.

4.2.4 Change when xend starts

There is, arguably, a bug in Red Hat/CentOS 5-series with regard to how and when Xen starts.

When Xen starts, it completely changes how the networking works on dom0. Any daemons that have started and expect a stable network will fail.

The most noticeable conflict is with the cluster manager. It's default is to start before xend. So when Xen shuts down networking for a moment, it will initiate a fence on the cluster because the communication is lost between nodes.

To work around this, we edit the `xend init.d` script to change it's start position to 11, well before any other networking dependent daemons. To do this, edit '`/etc/init.d/xend`' script and change the line '`# chkconfig: 2345 98 01`' to '`# chkconfig: 2345 11 89`'. Once the change is made, remove and re-add xend to the appropriate run levels:

```
chkconfig --del xend
chkconfig --add xend
```

5 Initial Cluster Setup

Now that the OS is installed, it is time to setup the cluster infrastructure. This must be done **before** the shared file system can be setup. Specifically, setting LVM to be cluster-aware is not possible until these steps are completed.

5.1 Networking in Xen

The reason this discusses Xen is to introduce networking in Xen and how it relates to clustering. This can be a particularly tricky topic. The best way to explain this is to step through the various elements that make up networking in Xen.

5.1.1 ethX

Normally, this is the name of a "real" ethernet device. However, under Xen, this is not the case after the 'xend' daemon has started.

Under Xen, what 'ethX' actually refers to depends on where you are. Under dom0, 'ethX' is a virtual ethernet device which has a copy of the physical ethernet device's MAC and IP addresses. On domU, 'ethX' is a virtual ethernet device connecting to a matching 'vifX.Y' device created on dom0 by Xen when domU was started. See the **vifX.Y** definition below.

Replace 'X' in 'ethX' with a number reflecting a given ethernet device. For example, 'eth0' is the first ethernet device, 'eth1' the second and so on.

5.1.2 pethX

The 'pethX' device(s) exist only on dom0.

When 'xend' starts, it renames the "real" 'ethX' to 'pethX' in order to create a virtual copy of it. This is done to facilitate bridging and routing. For example, this allows you to provide a route from a domU to a dom0 without touching the physical ethernet device.

So then, 'pethX' devices are in fact the interfaces to the real hardware ethernet devices. Traffic from any domU or from dom0 must route through the bridge in order to reach the physical device and find it's way out of the node.

Replace 'X' in 'pethX' with a number reflecting a given ethernet device. For example, 'peth0' is the first ethernet device, 'peth1' the second and so on.

5.1.3 vethX

The 'vethX' device(s) exist only on dom0 and only for a moment during startup.

During the Xen startup process, 'ethX' is brought down and it's MAC and IP addresses are copied to 'vethX'. Once done, 'ethX' is renamed to 'pethX' to move it out of the way. Then 'vethX' is renamed to 'ethX' to take the place of the 'ethX' interface. As such, 'vethX' devices exist for only a short time. You should never need to worry about or factor in these devices into your networking plans.

Replace 'X' in 'vethX' with a number reflecting a given virtual ethernet device. For example, 'veth0' is the first virtual ethernet device, 'veth1' the second and so on.

5.1.4 vifX.Y

The 'vifX.Y' devices exist only on dom0.

Note; in this section, 'X' matches the domU number, 'Y' matches a given 'ethX' device on dom0 and 'Z' matches a given 'ethX' device inside a domU. I know this is a little odd, but you need to get this clear in order to see how things connect here.

These are Xen's virtual interfaces used to connect an 'ethX' device on dom0 to a 'ethZ' device in a domU. Please be aware that there need not be any correlation between 'eth0' on dom0 and 'eth0' on domU! This is why 'X' and 'Z' are used separately here.

When Xen starts a domU, that virtual machine is assigned an ID#. This ID# is changed every time a virtual server is booted.

With the domU ID number plus the number from a dom0 ethY device, a 'vifX.Y' device is created and connected to the appropriate 'xenbrY' device. You can think of this as a given port on a switch. When the domU boots, it's internal 'ethZ' device connects to the proper 'xenbrY' via this new 'vifX.Y' device.

For example, if you want to connect eth2 on dom0 to eth0 in dom4, you will do so via vif4.2 where '4' matches the domU ID# and '2' matches the ethernet device number in dom0. When dom4 brings up it's 'eth0', it “plugs” into 'vif4.2'. If this is still confusing, take a look at the network map below.

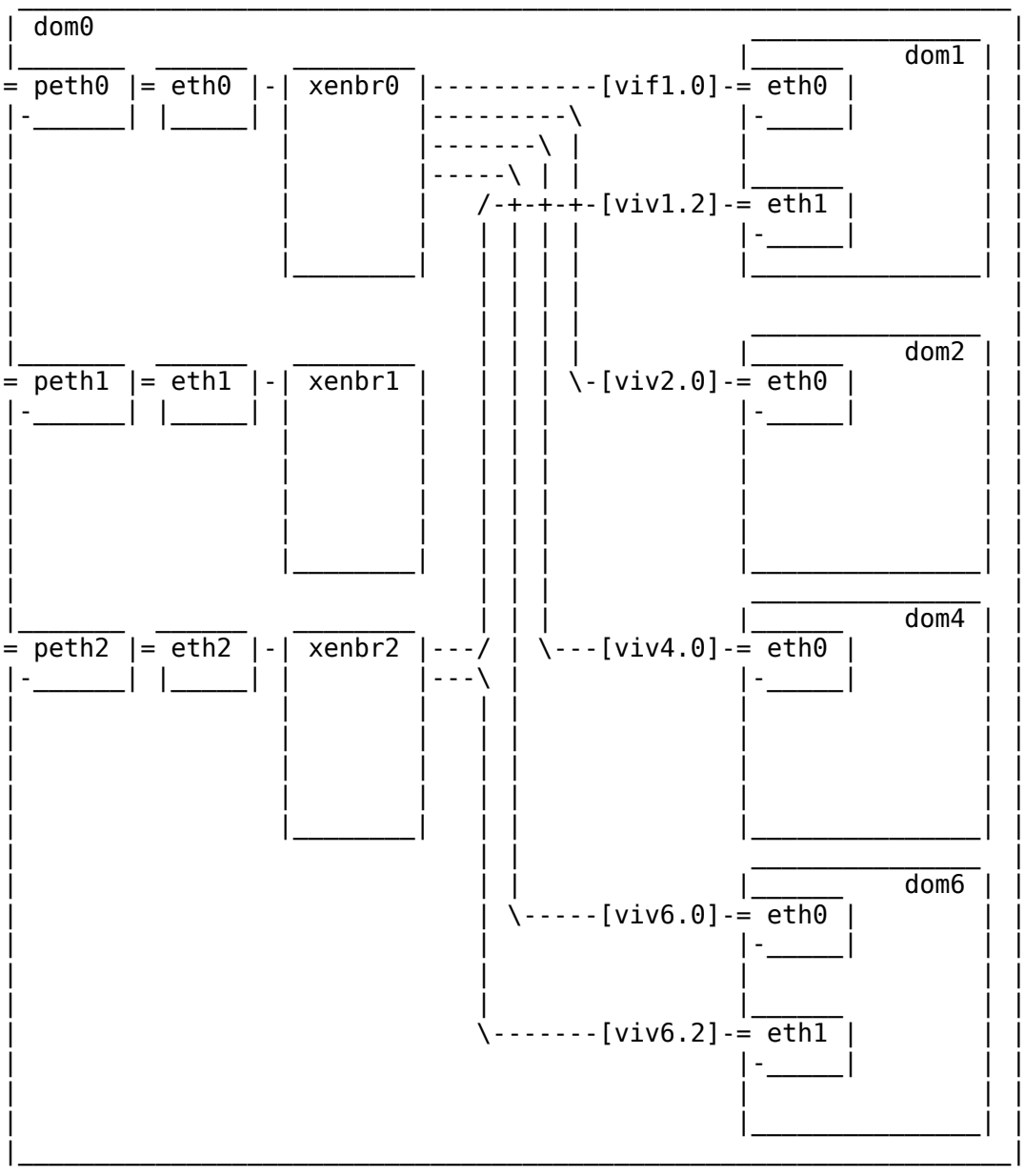
5.1.5 xenbrX

The 'xenbrX' device(s) exist only on dom0.

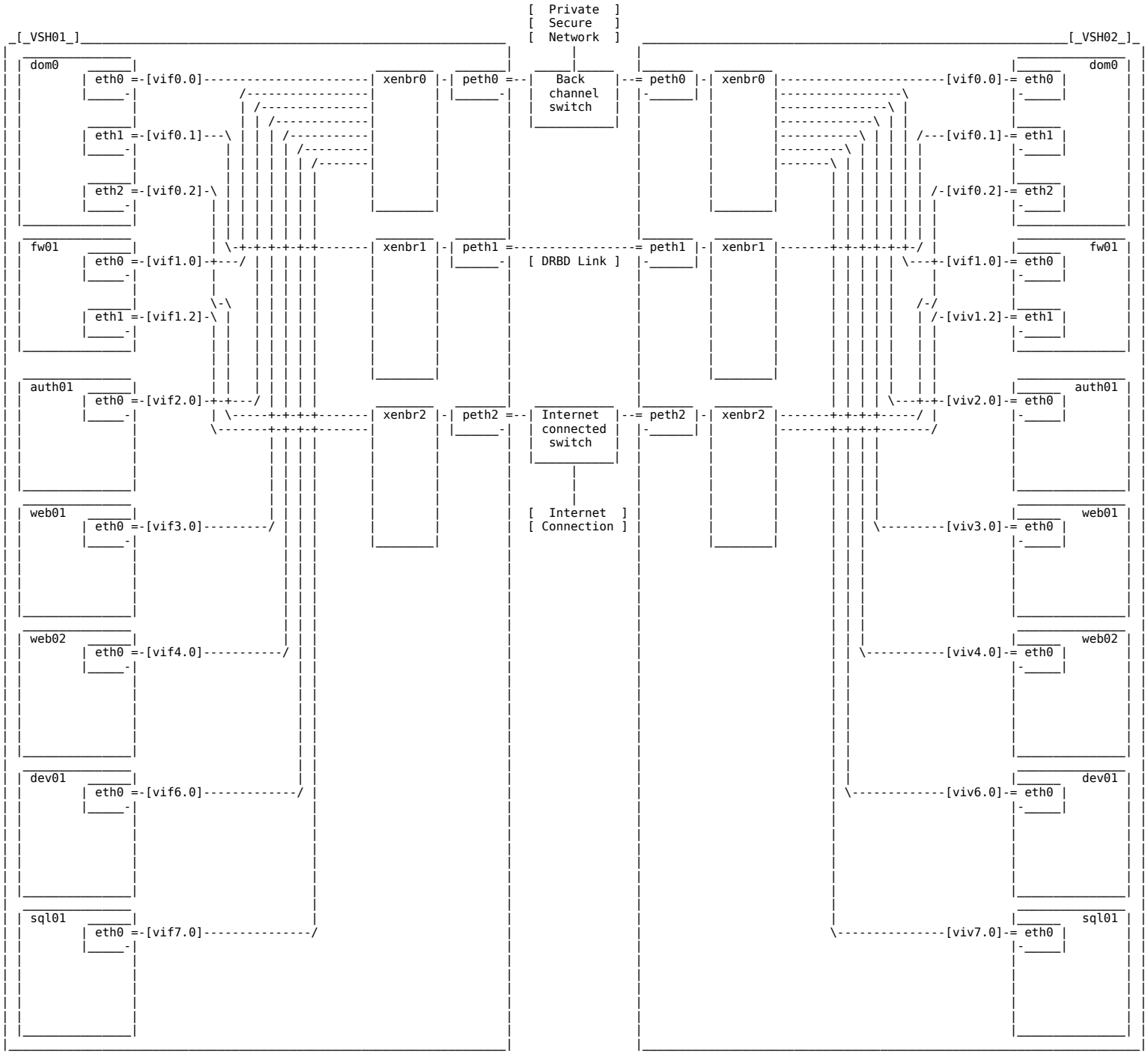
Xen uses software bridges to route traffic within it's virtual network. These can be thought of as a "switch" your would find in any normal network. A unique Xen bridge is created for each Xen-managed 'ethX' device on dom0.

5.1.6 Network Map - Putting it all together

The following two maps shows four VMs on a single node. The second shows a complete production environment hosting floating virtual machines on a 2-Node cluster. Note in the second map that VMs are listed as being on both nodes. From a data point of view, this is true, but each VM will be running on only one node at a time.



Example Networking Layout of a Production Two-Node VM Cluster Hosting Multiple Floating VMs



5.1.7 Adding NICs to Xen Management

By default, 'xend' only manages '`eth0`'. We need to add '`eth2`' to Xen and, if you wish, '`eth1`'. You can see which devices are under Xen's control by running `ifconfig` and checking to see if there is a '`pethX`' corresponding to each '`ethX`' device. For example, here is what you would see if only '`eth0`' was under Xen's control:

```
eth0      Link encap:Ethernet  HWaddr 90:E6:BA:71:82:D8
          inet addr:192.168.1.71  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::92e6:baff:fe71:82d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:121 errors:0 dropped:0 overruns:0 frame:0
          TX packets:97 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:20605 (20.1 KiB)  TX bytes:16270 (15.8 KiB)

eth1      Link encap:Ethernet  HWaddr 00:21:91:19:96:5A
          inet addr:10.0.0.71  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::221:91ff:fe19:965a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:45 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9139 (8.9 KiB)  TX bytes:10259 (10.0 KiB)
          Interrupt:16

eth2      Link encap:Ethernet  HWaddr 00:0E:0C:59:45:78
          inet addr:10.0.1.71  Bcast:10.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20e:cff:fe59:4578/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:45 errors:0 dropped:0 overruns:0 frame:0
          TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:9790 (9.5 KiB)  TX bytes:11102 (10.8 KiB)
          Base address:0xec00 Memory:febe0000-fec00000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)

peth0     Link encap:Ethernet  HWaddr FE:FF:FF:FF:FF:FF
          inet6 addr: fe80::fcff:ffff:feff:ffff/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:126 errors:0 dropped:0 overruns:0 frame:0
          TX packets:110 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20923 (20.4 KiB)  TX bytes:18352 (17.9 KiB)
          Interrupt:252 Base address:0x6000

vif0.0    Link encap:Ethernet  HWaddr FE:FF:FF:FF:FF:FF
          inet6 addr: fe80::fcff:ffff:feff:ffff/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:126 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:19306 (18.8 KiB)  TX bytes:20935 (20.4 KiB)

virbr0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          inet6 addr: fe80::200:ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:9640 (9.4 KiB)

xenbr0    Link encap:Ethernet  HWaddr FE:FF:FF:FF:FF:FF
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:148 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:24256 (23.6 KiB)  TX bytes:0 (0.0 b)
```

You'll notice that there is no 'peth1' or 'peth2' device, nor their associated virtual devices or bridges. To change this, create an **executable** '/etc/xen/scripts/an-network-script' file. This script will be used by Xen to create bridges for all NICs.

Please note three things;

1. You don't need to use the name 'an-network-script'. I suggest this name mainly to keep in line with the rest of the 'AN!x' projects.
2. If you install convirt, it will create it's own bridge script called convirt-xen-multibridge. Other tools may do something similar. If you install convirt or similar, be sure to use only one network script.
3. In this case, adding eth1 is optional, as we know ahead of time that eth1 will not be made available to any virtual machines as it is dedicated to DRBD and totem. I'm adding it here because I like having things consistent; Do whichever makes more sense to you.

First, touch the file and then chmod it to be executable.

```
touch /etc/xen/scripts/an-network-script
chmod 755 /etc/xen/scripts/an-network-script
```

Now edit it to contain the following:

```
#!/bin/sh
dir=$(dirname "$0")
"$dir/network-bridge" "$@" vifnum=0 netdev=eth0 bridge=xenbr0
"$dir/network-bridge" "$@" vifnum=1 netdev=eth1 bridge=xenbr1
"$dir/network-bridge" "$@" vifnum=2 netdev=eth2 bridge=xenbr2
```

Now tell Xen to reference that script by editing '/etc/xen/xend-config.sxp' and changing the 'network-script' argument to point to this new script:

```
#(network-script network-bridge)
(network-script an-network-script)
```

Restart 'xend' and if everything worked, you should now be able to run ifconfig and see that all the ethX devices have matching pethX.

5.2 Fencing

> *"The only thing you don't know, is what you don't know."*

Before we can discuss fencing, we must understand its critical role in clustering.

Take this scenario:

One node starts a write to the shared DRBD partition. Part way through the write, it stops responding. At some point, the other cluster manager must decide that the first node will never complete the write. At that time, it will want to replay the file system's journal. Without fencing, the second node could begin replaying the journal and then the first node could recover and complete it's write. Congratulations, you probably just corrupted your file system.

To make the above scenario safe, the cluster software adds a step. Once the cluster decides the stuck node is dead, all I/O transactions will be blocked. Then it will issue a fence against the stuck node and wait for the fence to return a success exit code. Only then, when it is confident that the stuck node **can not** return, will it begin the process of replaying the journal.

5.2.1 Warning!

It is a common beginner's mistake to think "*I'm just learning, I don't care about the data so I don't need to worry about fencing yet*".

WRONG.

I made this mistake, and it cost me the better part of two weeks and led to no end of headaches. The cluster manager doesn't have a "test" switch; It always operates under the assumption that everything is critical. Without proper fencing, ***things will not work***!

In the past, there was a "manual" fence option, but this has been removed and is no longer supported. ***You must have a fence!*** There is simply no two ways about it.

5.2.2 The Special Case of Two Node Clusters

When communication is lost in a cluster, the cluster manager running on each node must make a decision about it's state. In 3+ node clusters, each node will count how many nodes it can still talk to. If that resulting number is **less** than 'N / 2', where 'N' is the total number of nodes, the node will self-fence, removing itself from the cluster. The segment with **more** than 'N / 2' nodes is considered to have 'quorum', will issue a fence against lost nodes and then continue to operate.

In two node clusters, there can never be quorum in a degraded state, as there is never a majority because '2 / 1' is always 1. To accommodate this, there is a special switch we will see later.

With regard to fencing, when the communication link(s) break down between the two nodes, there will be a race between the two nodes to fence each other. This is okay because the faster node will kill the other node before it gets it's fence call out.

Think of this like an old west duel; Both nodes may try to pull out their guns, but only one will live long enough to shoot.

5.2.3 Implementing

In Red Hat's cluster software, the fence device(s) are configured in the main '`/etc/cluster.conf`' cluster configuration file. This configuration is then acted on via the fenced daemon. We'll cover the details of the '`cluster.conf`' file in a moment.

When the cluster determines that a node needs to be fenced, the fenced daemon will consult the `cluster.conf` file for information on how to access the fence device. Given this `cluster.conf` snippet:

```
<cluster name="an-cluster" config_version="1">
	<clusternodes>
		<clusternode name="an-node02.alteeve.com" nodeid="2">
			<fence>
				<method name="node_assassin">
					<device name="motoko" port="02" action="off"/>
				</method>
			</fence>
		</clusternode>
	</clusternodes>
	<fencedevices>
		<fencedevice name="motoko" agent="fence_na" quiet="true"
		ipaddr="motoko.alteeve.com" login="motoko" passwd="secret">
		</fencedevice>
	</fencedevices>
</cluster>
```

If the cluster manager determines that the node '`an-node02.alteeve.com`' needs to be fenced, it looks at the first (and only, in this case) `<fence>` entry's name, which is '`motoko`'. It then looks in the `<fencedevices>` section for the device with the matching name. Once found, it gets the information needed to find and access the fence device. Once it connects to the fence device, it then passes the options set in `an-node02.alteeve.com`'s `<fence>` argument.

So in this example, `fenced` looks up the details on the "*motoko*" Node Assassin fence device. It calls the `fence_na` program, called a "fence agent", and passes it the following arguments:

- `ipaddr=motoko.alteeve.com`
- `login=motoko`
- `passwd=secret`
- `quiet=true`
- `port=2`
- `action=off`

How the fence agent acts on these arguments varies depending on the fence device itself. The `FenceAgentAPI` dictates only that a fence agent return specific error codes indicating success or failure for defined actions.

If the first attempt fails, the fence agent will try the next <fence> method, if a second exists. It will keep trying fence devices listed under the node in the order they are found in the `cluster.conf`.

If the fence daemon fails to fence the node it will "block", that is, lock up the entire cluster and stop responding until the issue is resolved. The logic for this is that a locked up cluster is better than a corrupted one.

If any of the fence devices succeed, the cluster will know that it is safe to proceed and will reconfigure the cluster without the defective node. In clustering, to “reconfigure” is to take a new head-count of available nodes and restart management of the cluster with the new configuration. This generally happens silently and seamlessly.

5.2.4 Fence Devices

Fencing, isolating a node from altering shared resources, can be accomplished in a couple of ways:

- Power
  - Power fencing is where a device is used to cut the power to a malfunctioning node. This is probably the most common type. It can be accomplished using something as simple as a network-addressable power-bar or be as complex as IPMI-type baseboard management card.
- Blocking
  - Blocking is often implemented at the network level. This type of fencing leaves the node alone and running. Instead, it disconnects it from the shared resources. Often this is done by a switch which prevents traffic coming from the fenced node.

Many major OEMs have their own remote management devices that can serve as fence devices. Examples are Dell's 'DRAC' (Dell Remote Access Controller), HP's iLO (Integrate Lights Out), IBM's 'RSA' (Remote Supervisor Adapter), Sun's 'SSP' (System Service Processor) and so on. Smaller manufacturers like ASUS implement remote management via standard IPMI.

5.2.4.1 Node Assassin

A cheap alternative is the “Node Assassin”, an open-hardware, open source cluster fence device. It was built to allow the use of commodity system boards that lacked remote management support.

Full Disclosure: Node Assassin was created by the author, with much help from others.

You can learn more about Node Assassin, including full build instructions, at the project website:

- http://nodeassassin.org

5.3 Core Configuration Files

There are two core configuration files we'll need to setup; '`cluster.conf`' and '`openais.conf`'.

5.3.1 /etc/cluster/cluster.conf

The core of the cluster is the '`/etc/cluster/cluster.conf`' XML configuration file. It contains information about the cluster itself, what nodes are to be used, how to fence each node, what fence devices exist plus miscellaneous other configuration options.

By default, there is no `cluster.conf`, so you need to start by creating it.

Here is the one *AN!Cluster* uses, with in-line comments. Pay particular attention to the cman 'two_node="1"' argument near the top of the file. This is what allows the cluster to run without traditional quorum.

```
<?xml version="1.0"?>
<!--
The cluster's name is "an-cluster" and, as this is the first version of this file, it is set to version "1". Each time this file
changes in any way, the version number will have to be manually incremented by 1.
-->
<cluster name="an-cluster" config_version="1">
        <!--
        This is a special cman argument to enable cluster services to run without quorum. Being a two-node cluster,
        quorum with a failed node is quit impossible. :)  If we had no need for this, we'd just put in the self-closing
        argument: <cman/>
        -->
        <cman two_node="1" expected_votes="1">
        </cman>
        <!-- This is where the nodes in this cluster are defined. -->
        <clusternodes>
                <!-- AN!Cluster Node 1 -->
                <!-- The clusternode names must match the name returned by `uname -n`. -->
                <clusternode name="an-node01.alteeve.com" nodeid="1">
                        <fence>
                                <!--
                                The entries here reference devices defined below in the <fencedevices/> section. The
                                options passed control how the device is called. When mutitle CentOSltiple devices are
                                listed, they are tried in the order that the are listed here.
                                -->
                                <method name="node_assassin">
                                        <device name="motoko" port="01" action="off"/>
                                </method>
                        </fence>
                </clusternode>

                <!-- AN!Cluster Node 2 -->
                <clusternode name="an-node02.alteeve.com" nodeid="2">
                        <fence>
                                <method name="node_assassin">
                                        <device name="motoko" port="02" action="off"/>
                                </method>
                        </fence>
                </clusternode>
        </clusternodes>
        <!--
        The fence device is mandatory and it defined how the cluster will handle nodes that have dropped out of
        communication. In our case, we will use the Node Assassin fence device.
        -->
        <fencedevices>
                <!--
                This names the device, the agent (script) to controls it, where to find it and how to access it.
                -->
                <fencedevice name="motoko" agent="fence_na" ipaddr="motoko.alteeve.com" login="motoko"
                passwd="secret" quiet="1"></fencedevice>
                <!--
                If you have two or more fence devices, you can add the extra one(s) below. The cluster will attempt to
                fence a bad node using these devices in the order that they appear.
                -->
        </fencedevices>

        <!-- When the cluster starts, any nodes not yet in the cluster may be fenced. By default, there is a 6 second
        buffer, but this isn't very much time. The following argument increases the time window where other nodes can
        join before being fenced. I like to give up to one minute but the Red Hat man page suggests 20 seconds. Please
        do your own testing to determine what time is needed for your environment.
        -->
        <fence_daemon post_join_delay="60">
        </fence_daemon>
</cluster>
```

Once you're comfortable with your changes to the file, you need to validate it. Run:

```
xmllint --relaxng /usr/share/system-config-cluster/misc/cluster.ng /etc/cluster/cluster.conf
```

If there are errors, address them. Once you see: "`/etc/cluster/cluster.conf validates`", you can proceed to the next step. If you try to proceed before successfully validating your file, some applications like Red Hat's '`system-config-cluster`' may break.

5.3.2 openais.conf

Where `cluster.conf` is the core configuration file for the cluster, OpenAIS is the master of ceremonies. It provides all the cluster functions referencing first it's own `/etc/ais/openais.conf` then the `/etc/cluster/cluster.conf` file. You can think of this file as a "low level" configuration file controlling the underlying mechanics, like how and when to detect failures in the cluster where `cluster.conf` contains the specific cluster configuration.

The OpenAIS configuration has many options. An example configuration showing nearly all options can be found here:

- http://wiki.alteeve.com/index.php/Two-Node_openais.conf

Below is a sample skeleton file that is usable by our cluster.

```
# Totem Protocol options.
totem {
	# This is the version number of this configuration file's format. Unlike 'cluster.conf's
	# 'config_version', this value *does not* change. Further, it must always be set to '2'.
	version: 2

	# When set to 'on', data will be encrypted using sober128 and that HMAC/SHA1 is used for
	# authentication. This adds a 36 byes header to all totem messages. When enabled, this
	# accounts for 75% of the CPU usage used by the aisexec. Further, it will substantially
	# increase the CPU requirements of your nodes and will reduce transfer speeds a non-trivial
	# amount. For this reason, only enable this when you are using an insecure network and be sure
	# to test to see how much overhead it incurs so that you can increase hardware resources if
	# needed. Please see 'man openais.conf' for two specific examples of performance trade-offs
	# seen when enabling this. The default is 'on'.
	secauth: off

	# When 'secauth' is 'off', this variable is ignored. When 'secauth' is 'on', this defined how
	# many threads may be used for encrypting and sending multicast messages. A value of '0'
	# disabled multiple threads. This is most useful on non-SMP machines.
	threads: 0

	# This is used to control how the Redundant Ring Protocol is used. If you only have one
	# 'interface' directive, the default is 'none'. If you have two, then please set 'active' or
	# 'passive'. The trade off is that, when the network is degraded, 'active' provides lower
	# latency from transmit to delivery and 'passive' may nearly double the speed of the totem
	# protocol when not CPU bound. Valid options: none, active, passive.
	rrp_mode: passive

	# At least one 'interface' directive is required within the 'totem' directive. When two are
	# specified, the one with 'ringnumber' of '0' is the primary ring and the second with
	# 'ringnumber' of '1' is the backup ring.
	interface {
		### This is the back-channel subnet, which is the primary network for the totem
```

```
		### protocol.

		# Increment the ring number for each 'interface' directive.
		ringnumber: 0

		# This must match the subnet of this interface. The final octal must be '0'. In this
		# case, this directive will bind to the interface on the 192.168.1.0/24 subnet, so this
		# should be set to '192.168.1.0'. This can be an IPv6 address, however, you will be
		# required to set the 'nodeid' in the 'totem' directive above. Further, there will be no
		# automatic interface selection within a specified subnet as there is with IPv4. In this
		# case, the primary ring will be on the interface with IPs on the 10.0.0.0/24 network
		# (ie: eth1).
		bindnetaddr: 10.0.1.0

		# This is the multicast address used by OpenAIS. Avoid the '224.0.0.0/8' range as that
		# is used for configuration. If you use an IPv6 address, be sure to specify a 'nodeid'
		# in the 'totem' directive above.
		mcastaddr: 226.94.1.1

		# This is the UDP port used with the multicast address above.
		mcastport: 5405
	}
	interface {
		### This is the DRBD subnet, which acts as a secondary, backup network for the totem
		### protocol.
		ringnumber: 1
		bindnetaddr: 10.0.0.0
		mcastaddr: 227.94.1.1
		mcastport: 5406
	}
}

# This directive controls if and how OpenAIS logs it's messages. All variables here are optional.
logging {
	to_syslog: yes
}

# AMF, the Availability Management Framework, is not enabled yet in OpenAIS so leave this set to
# 'disabled'.
amf {
	mode: disabled
}
```

5.4 Cluster First Start

If everything up until now was done right, you should be able to start your cluster for the first time. It can be useful to have a separate terminal window open with a tail watching `/var/log/messages` so that you can see if there are any problems.

On both nodes, in dedicated terminals, run:

```
clear; tail -f -n 0 /var/log/messages
```

This next step must be run on both nodes as soon as possible. If you try to start one node and wait too long to start the other node, the first node will think there is a problem and it will fence the second node. Remember the `<fence_daemon post_join_delay="60"></fence_daemon>` line in

`cluster.conf`? This is where it comes into play. The value you set is the "window" you have to start both nodes before a fence is issued. The default is 6 seconds.

On both nodes, in different terminals, check that cman is indeed stopped, then start it up:

```
/etc/init.d/cman status
ccsd is stopped
/etc/init.d/cman start
```

If all goes well, you should see something like this in each node's /var/log/messages file:

```
May 10 20:54:32 an-node01 kernel: DLM (built Mar 17 2010 12:05:05) installed
May 10 20:54:32 an-node01 kernel: GFS2 (built Mar 17 2010 12:05:47) installed
May 10 20:54:32 an-node01 kernel: Lock_DLM (built Mar 17 2010 12:05:54) installed
May 10 20:54:33 an-node01 ccsd[11167]: Starting ccsd 2.0.115:
May 10 20:54:33 an-node01 ccsd[11167]:  Built: Dec  8 2009 09:20:54
May 10 20:54:33 an-node01 ccsd[11167]:  Copyright (C) Red Hat, Inc.  2004  All rights reserved.
May 10 20:54:33 an-node01 ccsd[11167]: cluster.conf (cluster name = an-cluster, version = 1) found.
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] AIS Executive Service RELEASE 'subrev 1887 version 0.80.6'
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] Copyright (C) 2002-2006 MontaVista Software, Inc and contributors.
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] Copyright (C) 2006 Red Hat, Inc.
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] AIS Executive Service: started and ready to provide service.
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] Using default multicast address of 239.192.147.72
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Token Timeout (10000 ms) retransmit timeout (495 ms)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] token hold (386 ms) retransmits before loss (20 retrans)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] join (60 ms) send_join (0 ms) consensus (4800 ms) merge (200 ms)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] downcheck (1000 ms) fail to recv const (50 msgs)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] seqno unchanged const (30 rotations) Maximum network MTU 1500
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] window size per rotation (50 messages) maximum messages per rotation (17 messages)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] send threads (0 threads)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] RRP token expired timeout (495 ms)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] RRP token problem counter (2000 ms)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] RRP threshold (10 problem count)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] RRP mode set to none.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] heartbeat_failures_allowed (0)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] max_network_delay (50 ms)
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] HeartBeat is Disabled. To enable set heartbeat_failures_allowed > 0
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Receive multicast socket recv buffer size (262142 bytes).
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Transmit multicast socket send buffer size (262142 bytes).
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] The network interface [10.0.1.71] is now up.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Created or loaded sequence id 0.10.0.1.71 for this ring.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering GATHER state from 15.
May 10 20:54:35 an-node01 openais[11175]: [CMAN ] CMAN 2.0.115 (built Dec  8 2009 09:20:58) started
May 10 20:54:35 an-node01 openais[11175]: [MAIN ] Service initialized 'openais CMAN membership service 2.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais extended virtual synchrony service'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais cluster membership service B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais availability management framework B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais checkpoint service B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais event service B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais distributed locking service B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais message service B.01.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais configuration service'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais cluster closed process group service v1.01'
May 10 20:54:35 an-node01 openais[11175]: [SERV ] Service initialized 'openais cluster config database access v1.01'
May 10 20:54:35 an-node01 openais[11175]: [SYNC ] Not using a virtual synchrony filter.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Creating commit token because I am the rep.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Saving state aru 0 high seq received 0
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Storing new sequence id for ring 4
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering COMMIT state.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering RECOVERY state.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] position [0] member 10.0.1.71:
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] previous ring seq 0 rep 10.0.1.71
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] aru 0 high delivered 0 received flag 1
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Did not need to originate any messages in recovery.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Sending initial ORF token
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] CLM CONFIGURATION CHANGE
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] New Configuration:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Left:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Joined:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] CLM CONFIGURATION CHANGE
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] New Configuration:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.71)
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Left:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Joined:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.71)
May 10 20:54:35 an-node01 openais[11175]: [SYNC ] This node is within the primary component and will provide service.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering OPERATIONAL state.
May 10 20:54:35 an-node01 openais[11175]: [CMAN ] quorum regained, resuming activity
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] got nodejoin message 10.0.1.71
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering GATHER state from 11.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Creating commit token because I am the rep.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Saving state aru a high seq received a
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Storing new sequence id for ring 8
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering COMMIT state.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering RECOVERY state.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] position [0] member 10.0.1.71:
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] previous ring seq 4 rep 10.0.1.71
```

```
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] aru a high delivered a received flag 1
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] position [1] member 10.0.1.72:
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] previous ring seq 4 rep 10.0.1.72
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] aru c high delivered c received flag 1
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Did not need to originate any messages in recovery.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] Sending initial ORF token
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] CLM CONFIGURATION CHANGE
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] New Configuration:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.71)
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Left:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Joined:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] CLM CONFIGURATION CHANGE
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] New Configuration:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.71)
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.72)
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Left:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] Members Joined:
May 10 20:54:35 an-node01 openais[11175]: [CLM  ]          r(0) ip(10.0.1.72)
May 10 20:54:35 an-node01 openais[11175]: [SYNC ] This node is within the primary component and will provide service.
May 10 20:54:35 an-node01 openais[11175]: [TOTEM] entering OPERATIONAL state.
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] got nodejoin message 10.0.1.71
May 10 20:54:35 an-node01 openais[11175]: [CLM  ] got nodejoin message 10.0.1.72
May 10 20:54:35 an-node01 openais[11175]: [CPG  ] got joinlist message from node 2
May 10 20:54:36 an-node01 ccsd[11167]: Initial status:: Quorate
```

Last, configure the cman daemon to start when the node boots:

```
chkconfig cman on
```

Congratulations, you have a cluster!

What will you do with it? There are so many options...

Let's looks at a common tool used in clusters. In our case, we can use DRBD to host floating VMs, a shared GFS file system, run an iSCSI target and so on.

6 DRBD

DRBD; Distributed Replicating Block Device, can be thought of as a cluster version or RAID level 1. It takes raw partitions from different nodes and ensures that whatever is written to one node is also written to the other.

The drbd83 and kmod-drbd83-xen packages are not included in the default Red Hat/CentOS installation media, so we will need to install them now:

```
yum -y install drbd83.x86_64 kmod-drbd83-xen.x86_64
```

6.1 Create the LVM Logical Volume

Most of the remaining space on either node's LVM PV will be allocated to a new LV. This new LV will host either node's side of the DRBD resource.

6.1.1 Getting Ready

On both nodes:

First, use `pvscan` to see how much space you have left on you LVM PV:

```
  PV /dev/sda2   VG an-lvm01   lvm2 [465.50 GB / 443.97 GB free]
  Total: 1 [465.50 GB] / in use: 1 [465.50 GB] / in no VG: 0 [0   ]
```

On my nodes, each of which has a single 500GB drive, I've allocated only 20GB to dom0 so I've got over 440GB left free. I like to leave a bit of space unallocated because I never know where I might need it, so I will allocate 400GB even to DRBD and keep the remaining 44GB set aside for future growth. The space you have left and how you want to allocate is an exercise for the reader.

Next, use `lvscan` to check that the name you will give to the new LV isn't used yet:

```
  ACTIVE            '/dev/an-lvm01/lv01' [19.53 GB] inherit
  ACTIVE            '/dev/an-lvm01/lv00' [2.00 GB] inherit
```

I can see from the above output that '`lv00`' and '`lv01`' are used, so I will use '`lv02`' for my DRBD partition. Of course, you can use '`drbd`' or pretty much anything else you want.

6.1.2 Creating the LV on Each Node

Now that I know I want to create a 400GB logical volume called '`lv02`', I can proceed.

The next two commands show what I need to call on my nodes, and will match what you need to run if you used the *AN!Cluster* Install DVD. If you ran your own install, be sure to edit the following arguments to match your nodes:

On an-node01:

```
lvcreate -L 400G -n lv02 /dev/an-lvm01
```

On an-node02:

```
lvcreate -L 400G -n lv02 /dev/an-lvm02
```

If I re-run lvscan now, I will see the new volume:

```
  ACTIVE            '/dev/an-lvm01/lv01' [19.53 GB] inherit
  ACTIVE            '/dev/an-lvm01/lv00' [2.00 GB] inherit
  ACTIVE            '/dev/an-lvm01/lv02' [400.00 GB] inherit
```

We can now proceed with the DRBD setup!

6.2 Creating /etc/drbd.conf

DRBD is controlled by the '/etc/drbd.conf' configuration file which must be identical on both nodes. This file tells DRBD what devices to use on each node, what interface to use and so on.

The main options are documented in the following example file

```
# This directive controls global settings. It should be the first directive in this configuration file.
global {
	# This is an optional argument that let's the folks at Linbit count you as a user of DRBD. This can be
	#  turned off by changing this value to 'no'.
	usage-count yes;
}

# The values here are inherited by any resources configured below.
common {
	# Protocol dictates what DRBD considers to be a "completed write".
	# There are three options;
	# A;
	#    A write is considered complete once the data has been written to the node's local disk and has been
	#    sent to the TCP send buffer. This option provides the highest performance with the greatest risk in
	#    case of a sudden node failure. Use with caution.
	# B;
	#    Like A, but instead waits for the data to reach the remote node's TCP receive buffer before reporting
	#    completed write. This is a balance of performance and safety, but can still cause problems if both
	#    nodes lose power simultaneously before the write is committed to disk on the remote node(s). Use with
	#    caution.
	# C;
	#    This is the slowest but safest option. It will not report a completed write until the data has
	#    reached both the local and remote disk(s). This is required for Primary/Primary mode and for cluster
	#    aware file systems like cluster-enabled LVM. Strongly suggested.
	protocol C;

	# This sets the upper limit on the DRBD synchronization process. It does NOT need to be very fast, as the
	# array is available on both nodes during sync. Experiment to find a speed that does not impact your
	# cluster's performance.
	syncer {
		rate 33M;
	}
}

# This is an example of a resource directive. There will be a named resource directive for each DRBD device you
# want to create. Options set in the 'common' directive are inherited here. Each resource must have at least two
```

```
# 'on' directives which indicate what device on each node that this resource will be created on. This example
# shows a resource call 'r0'. You can name a resource pretty much anything you want.
resource r0 {
	# This is the name of the device that will be created for this resource. You will use this device to
	# access the DRBD partition, NOT the underlying block devices specified in the 'disk' arguments below.
	device    /dev/drbd0;

	# Please see 'man 8 drbdsetup' for a complete list of options available.
	net {
		# This enables Primary/Primary mode needed for our cluster.
		allow-two-primaries;
	}

	# Please see 'man 8 drbdsetup' for a complete list of options available.
	startup {
		# Tell the array to bring both nodes up as Primary on start.
		become-primary-on both;
	}

	# This tells DRBD where to store the meta-disk data for the DRBD array. This is usually set to 'internal'
	# which means that some space is set aside at the end of the device. However, if you have multiple DRBD
	# devices, you may with to set this to a specific partition. In that case, the partition needs to be at
	# least 256MB. Optionally, this argument can be replaced by 'flexible-meta-disk', in which case you will
	# need to specify a specific partition and the size needed will be ((36kb+(DRBD data size))/32kb), rounded
	# up to the next even 4kb size.
	meta-disk     internal;

	###############################################################################
	# The options below are used when you want to set the same values to  #
	# two or more 'on' subdirectives. Using these options enables the use #
	# of the 'floating' subdirective in the place of 'on' subdirectives.  #
	###############################################################################

	# This defines a common block device to use on nodes without an explicitly defined 'disk' argument.
	#disk          /dev/an-lvm01/lv02;

	# The 'floating' argument can be a simple argument style or a subdirective defining different
	# configuration options from the above shared values. When used, it replaces the corresponding 'on'
	# subdirective. Here are two examples showing both methods of using it:
	#floating 10.0.0.72:7789;
	# and/or:
	#floating 10.0.0.72:7789 {
	#	disk          /dev/an-lvm02/lv02;
	#}
	# As with 'on', two or more 'floating' and/or 'on' arguments must be
	# used.

	###############################################################################
	# The 'on' subdirectives tell DRBD which nodes have which devices to  #
	# use in the DRBD array.                                              #
	###############################################################################

	# This is the first 'on' subdirective. The name after the 'on' syntax MUST be the name returned by
	# 'uname -r' on the given node.
	on an-node01.alteeve.com {
		# This is the IP address that the name above resolves to. Be sure that this IP is to one on your
		# DRBD subnet and corresponds to your desired ethernet device.
		address        10.0.0.71:7789;

		# Note that if you are not using IPv4, you will need to specify the address type between the
		# 'address' argument and the value like so:
		#address ipv6  [::f38a]:7789;
		# Valid types are:
		# ipv4         Default, not required
		# ipv6         Address must be in square-brackets

		# This is the raw block device that will be used on this node. If a common 'disk' is defined above
		# and it matches the disk to use on this node, this can be left out.
		disk          /dev/an-lvm01/lv02;
	}

	# This is the second node's 'on' subdirective. As with above, the name given MUST match the name returned
	# by 'uname -n' on this node.
```

```
	on an-node02.alteeve.com {
		address		10.0.0.72:7789;
		disk		/dev/an-lvm02/lv02;
	}
}
```

6.3 Setup the DRBD Resource r0

From the rest of this section, pay attention to whether you see:

- Primary
- Secondary
- Both

These indicate which node to run the following commands on. There is no functional difference between either node, so just randomly choose one to be *Primary* and the other will be *Secondary*. Once you've chosen which is which, be consistent with which node you run the commands on. Of course, if a command block is proceeded by *Both*, run the following code block on both nodes.

Both:

Start the DRBD daemon:

```
/etc/init.d/drbd restart
```

You should see output like this:

```
Restarting all DRBD resources: Could not stat("/proc/drbd"): No such file or directory
ERROR: Module drbd does not exist in /proc/modules
.
```

Don't worry about those errors.

You can verify that it started properly by checking the DRBD daemon's status and by checking what is in '`/proc/drbd`'.

Check the daemon:

```
/etc/init.d/drbd status
```

It should return output like:

```
version: 8.3.2 (api:88/proto:86-90)
GIT-hash: dd7985327f146f33b86d4bff5ca8c94234ce840e build by mockbuild@v20z-x86-64.home.local, 2009-08-29 14:08:07
m:res  cs         ro                 ds                 p  mounted  fstype
0:r0   Connected  Secondary/Secondary  UpToDate/UpToDate  C
```

Check the special `procfs` file:

```
cat /proc/drbd
```

It should return output like:

```
version: 8.3.2 (api:88/proto:86-90)
GIT-hash: dd7985327f146f33b86d4bff5ca8c94234ce840e build by mockbuild@v20z-x86-64.home.local, 2009-08-29 14:08:07
 0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r----
    ns:524288 nr:0 dw:0 dr:524288 al:0 bm:127 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

If you see the output above, you're good to proceed.

Both:

Initiate the DRBD device:

```
drbdadm create-md r0
```

It should return output like:

```
Device '0' is configured!
```

If you see an error after the line above, don't worry about it.

Primary:

Start the sync between the two nodes by calling:

```
drbdadm -- --overwrite-data-of-peer primary r0
```

Secondary:

At this point, we need to promote the secondary node to 'Primary' position.

```
drbdadm primary r0
```

Both:

Make sure that both nodes are now `Primary` by running:

```
cat /proc/drbd
```

It should return output like:

```
version: 8.3.2 (api:88/proto:86-90)
GIT-hash: dd7985327f146f33b86d4bff5ca8c94234ce840e build by mockbuild@v20z-x86-64.home.local, 2009-08-29 14:08:07
 0: cs:Connected ro:Primary/Primary ds:UpToDate/UpToDate C r----
    ns:524288 nr:0 dw:0 dr:524288 al:0 bm:127 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

The DRBD partition has no file system, you should not see the devices sync'ing at this point.

7 LVM on top of DRBD

If you used the *AN!Cluster* kickstart files, or if you based your install on them, then you are already using LVM on the cluster nodes as the underlying system for all but the `/boot` partition. Each node should have a VG named the same as the node itself with three LVs on them.

Now we will "stack" LVM by creating a PV on top of the new '`/dev/drbd0`' DRBD partition that we created in the previous step. Since this new LVM PV will exist on top of the shared DRBD partition, whatever get written to it's logical volumes will be immediately available on either node, regardless of which node actually initiated the change.

This capability is the underlying reason for creating this cluster; Neither machine is truly needed so if one machine dies, anything on top of the DRBD partition will still be available. When the failed machine returns, the surviving node will have a list of what blocks changed while the other node was gone and can use this list to quickly re-sync the other node!

7.1 Making LVM Cluster-Aware

Normally, LVM is run on a single server. This means that at any time, the LVM can write data to the underlying drive and not need to worry if any other device might change anything. In clusters, this isn't the case. The other node could try to write to the shared storage, so then nodes need to enable "locking" to prevent the two nodes from trying to work on the same bit of data at the same time.

The process of enabling this locking is known as making LVM "cluster-aware".

7.1.1 Updating '/etc/lvm/lvm.conf'

Note: After CentOS 5.5 is released, this step should no longer be needed. For details, see:

- https://bugzilla.redhat.com/show_bug.cgi?id=530881

By default, LVM ignores DRBD devices as candidate physical volumes. To enable DRBD, we need to change the '`filter`' argument in '`/etc/lvm/lvm.conf`' to include a regular expression that matches the name of our DRBD device and hard drives. We created our DRBD device as '`/dev/drbd`', so changing the filter to '`filter = [ "a|drbd.*|", "a|sd.*|", "r|.*|" ]`' will do this.

Edit `/etc/lvm/lvm.conf` and change it to match this:

```
    # By default we accept every block device:
    #filter = [ "a/.*/" ]
    filter = [ "a|drbd.*|", "a|sd.*|", "r|.*|" ]
```

7.1.2 Enabling Cluster Locking

LVM has a built-in tool called `lvmconf` that can be used to enable LVM locking. Simply run:

```
lvmconf --enable-cluster
```

There won't be any output from that command.

By default, `clvmd`, the cluster lvm daemon, is stopped and not set to run on boot. Now that we've enabled LVM locking, we need to start it:

```
/etc/init.d/clvmd status
```

It should return output like:

```
clvmd is stopped
active volumes: lv00 lv01 lv02
```

As expected, it is stopped, so lets start it and then use `chkconfig` to enable it at boot.

```
/etc/init.d/clvmd start
```

It should return output like:

```
Stopping clvm:                                             [  OK  ]
Starting clvmd:                                            [  OK  ]
Activating VGs:   3 logical volume(s) in volume group "an-lvm01" now active
                                                           [  OK  ]
```

Finally, make sure `clvmd` starts on boot:

```
chkconfig clvmd on
```

7.1.3 Creating a new PV using the DRBD Partition

We can now proceed with setting up the new DRBD-based LVM physical volume. Once the PV is created, we can create a new volume group and start allocating space to logical volumes.

Primary:

To setup the DRBD partition as an LVM PV, run `pvcreate`:

```
pvcreate /dev/drbd0
```

It should return output like:

```
  Physical volume "/dev/drbd0" successfully created
```

Both:

Now, on both nodes, check that the new physical volume is visible by using `pvdisplay`:

```
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               an-lvm01
  PV Size               465.51 GB / not usable 14.52 MB
  Allocatable           yes
  PE Size (KByte)       32768
  Total PE              14896
  Free PE               1407
  Allocated PE          13489
  PV UUID               JfdHFm-dWQF-7RzV-Zs9C-rUsI-ce98-H9o857

  "/dev/drbd0" is a new physical volume of "399.99 GB"
  --- NEW Physical volume ---
  PV Name               /dev/drbd0
  VG Name
  PV Size               399.99 GB
  Allocatable           NO
  PE Size (KByte)       0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               SWdWdG-IVkK-tBLc-XzrQ-809r-xpH8-BHARln
```

If you see PV Name /dev/drbd0 on both nodes, then your DRBD setup and LVM configuration changes are working perfectly!

If you only see one entry, double-check that the '`filter`' argument is accurate. If you need to change it, restart the `clvmd` daemon after the change and run `pvdisplay` again.

7.1.4 Creating a VG on the new PV

Primary:

Now we need to create the volume group using the `vgcreate` command:

```
vgcreate -c y drbd_vg0 /dev/drbd0
```

It should return output like:

```
  Clustered volume group "drbd_vg0" successfully created
```

Both:

Now we'll check that the new VG is visible on both nodes using `vgdisplay`. It should return output like:

```
  --- Volume group ---
  VG Name               an-lvm01
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                3
  Open LV               3
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               465.50 GB
  PE Size               32.00 MB
  Total PE              14896
  Alloc PE / Size       13489 / 421.53 GB
  Free  PE / Size       1407 / 43.97 GB
  VG UUID               C0kHFA-OTo8-Gshr-3wIw-3Q0I-eT3X-A9Y0NA

  --- Volume group ---
  VG Name               drbd_vg0
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  Clustered             yes
  Shared                no
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               399.98 GB
  PE Size               4.00 MB
  Total PE              102396
  Alloc PE / Size       0 / 0
  Free  PE / Size       102396 / 399.98 GB
  VG UUID               TmlQmv-eViK-7Ubr-Dyck-0u86-uEWJ-rDOt9i
```

If the new VG is visible on both nodes, we are ready to create our first logical volume using the lvcreate tool.

7.1.5 Creating the First Two LVs on the new VG

Primary:

Now we'll create two simple 20 GiB logical volumes. This first one will be a shared GFS store for source ISOs and the second will could be used for a virtual machine.

```
lvcreate -L 20G -n iso_store drbd_vg0
lvcreate -L 20G -n vm01 drbd_vg0
```

It should return output like:

```
  Logical volume "iso_store" created
  Logical volume "vm01" created
```

Both:

As before, we will check that the new logical volume is visible from both nodes by using the `lvdisplay` command:

```
  --- Logical volume ---
  LV Name                /dev/an-lvm02/lv01
  VG Name                an-lvm02
  LV UUID                Dy2MNa-EUxN-9x6f-ovkj-NCpk-nlV2-kr5QBb
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                19.53 GB
  Current LE             625
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0

  --- Logical volume ---
  LV Name                /dev/an-lvm02/lv00
  VG Name                an-lvm02
  LV UUID                xkBu7j-wtOe-ORr3-68qJ-u0ux-Qif4-stw5SY
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                2.00 GB
  Current LE             64
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:1

  --- Logical volume ---
  LV Name                /dev/an-lvm02/lv02
  VG Name                an-lvm02
  LV UUID                R20GH1-wQKq-WgUR-x1gx-Yzzp-WjND-WHAjEO
  LV Write Access        read/write
  LV Status              available
  # open                 2
  LV Size                400.00 GB
  Current LE             12800
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:2

  --- Logical volume ---
  LV Name                /dev/drbd_vg0/iso_store
  VG Name                drbd_vg0
  LV UUID                svJx35-KDXK-ojD2-UDAA-Ah9t-UgUl-ijekhf
  LV Write Access        read/write
  LV Status              available
  # open                 0
```

```
LV Size                20.00 GB
Current LE             5120
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:3

--- Logical volume ---
LV Name                /dev/drbd_vg0/vm01
VG Name                drbd_vg0
LV UUID                sceLmK-ZJIp-fN5g-RMaS-j5sq-NuY5-7hIwhP
LV Write Access        read/write
LV Status              available
# open                 0
LV Size                20.00 GB
Current LE             5120
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:4
```

The last two are the new logical volumes.

7.2 Create the Shared GFS File System

To show a simple, functioning example of how our cluster can be used, we'll create a small GFS file system and mount it on both nodes. We will use it as a place to store ISOs that we could use to provision virtual machines and will name it accordingly.

The following example is designed for the cluster used in this paper. Please be aware:

- If you have more than 2 nodes, increase the '`-j 2`' to the number of nodes you want to mount this file system on.
- If your cluster is named something other than '`an-cluster`' (as set in the `cluster.conf` file), change '`-t an-cluster:iso_store`' to match you cluster's name. The '`iso_store`' can be whatever you like, but it must be unique in the cluster. I tend to use a name that matches the LV name, but this is my own preference and is not required.

Primary:

To format the partition run:

```
mkfs.gfs2 -p lock_dlm -j 2 -t an-cluster:iso_store /dev/drbd_vg0/iso_store
```

If you are prompted, press 'y' to proceed.

Once the format completes, you can mount and use the '`/dev/drbd_vg0/iso_store`' as you would a normal file system. Try mounting it on both nodes and then copy some data to one node. You should be able to see the files appear on the other node!

7.3 Recovering From a Crash

If your cluster ever crashes, the DRBD partition and the underlying LVM logical volumes may not start. To solve this, you need to first bring the LVM devices back online and then you need to restore the DRBD partition before you can use resources hosted on DRBD.

7.3.1 Manually Restarting LVM

If you run 'pvdisplay', 'vgdisplay' and 'lvdisplay' and your DRBD devices aren't visible, you will need to tell LVM to rescan. This is easily done by calling 'pvscan', 'vgscan' and 'lvscan' in order. The results should look like this:

```
root@an-node01:~# pvscan
  PV /dev/sda2    VG an-lvm01   lvm2 [465.50 GB / 43.97 GB free]
  PV /dev/drbd0   VG drbd_vg0   lvm2 [399.98 GB / 359.98 GB free]
  Total: 2 [865.48 GB] / in use: 2 [865.48 GB] / in no VG: 0 [0   ]
root@an-node01:~# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "an-lvm01" using metadata type lvm2
  Found volume group "drbd_vg0" using metadata type lvm2
root@an-node01:~# lvscan
  ACTIVE            '/dev/an-lvm01/lv01' [19.53 GB] inherit
  ACTIVE            '/dev/an-lvm01/lv00' [2.00 GB] inherit
  ACTIVE            '/dev/an-lvm01/lv02' [400.00 GB] inherit
  inactive          '/dev/drbd_vg0/iso_store' [20.00 GB] inherit
  inactive          '/dev/drbd_vg0/vm01' [20.00 GB] inherit
```

You'll notice that the DRBD logical volumes are listed 'inactive'. This is to be expected and is easily corrected using the 'lvchange' command. Here is an example of it's use:

```
root@an-node02:~# lvchange -a y /dev/drbd_vg0/iso_store
root@an-node02:~# lvchange -a y /dev/drbd_vg0/vm01
root@an-node02:~# lvscan
  ACTIVE            '/dev/an-lvm02/lv01' [19.53 GB] inherit
  ACTIVE            '/dev/an-lvm02/lv00' [2.00 GB] inherit
  ACTIVE            '/dev/an-lvm02/lv02' [400.00 GB] inherit
  ACTIVE            '/dev/drbd_vg0/iso_store' [20.00 GB] inherit
  ACTIVE            '/dev/drbd_vg0/vm01' [20.00 GB] inherit
```

7.3.2 Restoring Resources

At this point, your DRBD resources should be available again. Check this by looking at the /proc/drbd file. If the state is 'Primary/Primary' then you are fine. If it's not, then you will need to manually rebuild the resource.

Determine which node you trust to have the most recent and up to date copy of the DRBD data and run the following commands:

```
drbdadm primary r0
drbdadm -- --overwrite-data-of-peer connect r0
```

On the second node, run:

```
drbdadm primary r0
```

Check the `/proc/drbd` file again. If the state is '`Primary/Primary`' then you are fine. If it's still not, then please see:

- http://www.drbd.org/users-guide-emb/ch-troubleshooting.html

For advanced disaster recovery options.

8 Post

There are so many different ways that you could implement a cluster that it was very difficult to narrow down some topics for this talk. Over the next several months I will be building a set of wiki articles based on this talk.

This format will allow you to follow a set of detailed, step-by-step instructions in a “Build Your Own Adventure” style tutorial. Please visit:

- http://wiki.alteeve.com/index.php/2-Node_CentOS5_Cluster

Feedback will be very much appreciated and will help shape how the tutorials are developed.