Zenoss®

The Zenoss Enablement Series:

Resource Manager Corosync/DRBD HA Installation Guide

Document Version 422-P2

Zenoss, Inc.

www.zenoss.com

Table of Contents

Applies To

The procedure outlined in this document applies to the following Linux and Zenoss versions:

- Centos Linux 6.3
- Resource Manager Version 4.2.2 (Build 1675)

The procedure outlined in this document was tested on the following hypervisor platforms, using the following software versions:

- VMWare ESXi 5.0 and VMWare Workstation 8.04 hypervisors for cluster nodes
- Centos Linux 6.3 Linux nodes
- Resource Manager Version 4.2.2 (Build 1675)
- LCMC 1.4.2 (on windows 7 64 bit)

Introduction

The objective of setting up Zenoss in a 'High Availability' cluster is to minimize, to the greatest degree possible, the downtime associated with a hardware or (non Zenoss) software failure of the server hosting Zenoss. High Availability clusters come in a variety of possible configurations, including, but not necessarily limited to:

- Active-Passive, non geo-diverse
- Active-Active non geo-diverse
- Active-Passive, geo-diverse

This document will detail an 'Active – Passive' High Availability cluster without geo diversity using Corosync and DRBD. For our scenario two identical servers are deployed. At any given time, one node serves as the 'primary' active server and a second identical server stands by ready to take over the provision of the key Zenoss services in the event the first server fails or otherwise becomes unavailable. We label this solution as lacking 'geo diversity' in the sense that the two servers will be co-located in the same facility. This means that this solution provides no protection from a scenario that destroys or renders unreachable the facility hosting Zenoss.

Setting up an Active-Passive HA cluster to host Zenoss requires the following:

1. Two identical, fully dedicated servers for Zenoss cluster nodes.
2. Two Network Interface Cards (NICs) per node server. Ideally the second NIC card would be reserved for DRBD traffic.
3. The replication of key file system directories across the two cluster nodes.
4. The assignment of a cluster virtual IP address (VIP). The VIP can be claimed by either of the two nodes when needed to move access to Zenoss from one node to the other. The VIP is set up as a second IP address for one of the NIC cards residing on one of the nodes (at any given time).
5. Cluster resource manager software to manage the mounting of file systems, assignment of the VIP, and startup of key services on the backup node should the primary node fail (Or should a migration of services be desired for some other reason).
6. 'Heartbeat' monitoring of critical services running on the primary node, so that a migration from the primary to the backup may be triggered when needed.

The following free and open source software packages will be used in our example:

- DRBD for block level replication of the key file systems (http://www.drbd.org/)
- Corosync as the cluster resource manager (http://www.corosync.org/)
- Pacemaker for node process monitoring (http://clusterlabs.org)

For the purposes of this document, an example scenario will be referred to including sample values for items such as IP addresses, hostnames, subnet masks, etc. Administrators using this guide to deploy Zenoss in HA will need to substitute actual values where applicable. The two servers in our scenario will have two physical hard drives: `/dev/sda` and `/dev/sdb`. In our scenario, `sdb` is reserved purely for Zenoss data. Each server uses two NIC cards (eth0 and eth1). The second NIC card, eth1, is dedicated to DRBD and cluster traffic. These instructions assume that Centos 6 was installed as the operating system and that the 'Basic Server' install option was chosen. We assume that Centos has been fully updated. The commands listed are run as the *root* user unless otherwise specified. We also assume that the Linux servers hosting Zenoss have access to the standard Yum repositories (i.e., they have internet access).

Set Up Cluster Nodes

Before installing Zenoss for use in a high available configuration, you must set up two identical host system machines (referred to as Node 1 and Node 2), each with two NICs. See the *Resource Manager Installation Guide* for system requirements.

1. Install CentOS 6 on both systems and update all software packages on both nodes (required for recent versions of Corosync):

   ```
   yum update
   ```

2. Configure the network cards. The following are sample values for illustrative purposes (edit the appropriate Ethernet card configuration files to specify the appropriate values in your production environment):

   - Node 1

     ```
     eth0
     ip: 192.168.87.21/24 gateway: 192.168.87.2 dns: 192.168.87.2
     eth1
     ip: 192.168.10.5/30
     ```

   - Node 2

     ```
     eth0
     ip: 192.168.87.22/24 gateway: 192.168.87.2 dns: 192.168.87.2
     eth1
     ip: 192.168.10.6/30
     ```

3. Update the hostnames for each node by editing the /etc/sysconfig/network file:

   - Node 1

     ```
     HOSTNAME=node1.example.com
     ```

   - Node 2

     ```
     HOSTNAME=node2.example.com
     ```

4. For setup purposes, on Node 1 and Node 2, enter the following commands to disable the internal software firewall (after choosing the ports for DRBD and Corosync communication, the firewall can be re-enabled with the appropriate ports opened):

   ```
   chkconfig iptables off
   service iptables stop
   ```

5. SELinux is not compatible with either Zenoss or Corosync; therefore on Node 1 and Node 2, enter the following commands to disable SELinux

   ```
   echo 0 >/selinux/enforce
   sed -i 's/SELINUX=enforcing/SELINUX=disabled/'  /etc/selinux/config
   ```

6. To provide seamless connectivity between the two nodes, copy the master SSH keys from Node 1 to Node 2. Run the following commands on Node 1:

   ```
   scp -r /etc/ssh root@[node 2 hostname or ip]:/etc
   ```

Define the Shared Hostname / VIP

For a cluster to work, systems and administrators seeking access to Zenoss need to be guided to the VIP instead of the IP address for any of the individual nodes' NIC cards. Therefore identify and reserve a free IP address on your LAN, then associate a DNS record with it. In the following example the hostname associated with the VIP, and therefore used to access Zenoss, is `node.example.com`.

Once you have defined your VIP address hostname, update the `/etc/hosts` files on each node to be sure they can resolve it. Although DNS should have this record as well, we recommend these hosts file entries be made to prevent cluster problems in the event of DNS service interruptions.

Important: The short version of the hostname will be used by a RabbitMQ, so make sure to include it as noted in the following example:

1. In our example, the hosts file entries (made to `/etc/hosts` on both Node 1 and Node 2) are:

   ```
   192.168.87.20 node.example.com node
   ```

2. Reboot Node 1 and Node 2 after making all of the changes listed above:

   ```
   shutdown -r now
   ```

Create File system(s)

Next, you need to set up the file systems to be replicated between the two cluster nodes. At this stage, you have to options for setup of the file systems: you can partition a hard drive and install a file system on the partition, or you can use the more flexible (but more complicated) approach of creating Logical Volume Manager (LVM) volumes. The advantage of the latter approach is that you have more options to resize or move the partitions later on. In this example, we will demonstrate the LVM approach.

You also need to decide what data you wish to replicate across cluster nodes. The Zenoss software relies upon files both within and outside of its own home directory of `/opt/zenoss`. Files associated with dependency libraries are just one example of files Zenoss requires that reside outside of `/opt/zenoss`. More broadly, one could theoretically consider *all* Linux operating system files to be dependencies of Zenoss. So which files need to be replicated between cluster nodes and which should simply be duplicated on both nodes? To determine this, you balance the burden of duplicating files across the nodes against the need to replicate files that are dynamically changed by Zenoss while it operates. Replicating the contents of `/opt/zenoss` strikes a good balance between these competing needs, since the balance of the files on the Linux file system are not changed by Zenoss.

Rather than simply creating and replicating a single `/opt/zenoss` directory, we will choose a slightly more complex approach to file system replication for a production server. Specifically, in production we want to mount `/opt/zenoss/perf` and `/opt/zenoss/var/zeneventserver` with different options than other directories in `/opt/zenoss` due to the uniquely intensive read/write activities in these directories. For example, administrators may choose to mount these filesystems with the *noatime* flag set.
In this example, we will illustrate the creation of four different file systems to gain the desired level of flexibility when mounting.

Optionally, begin by updating the `/etc/lvm/lvm.conf` file on both Node 1 and Node 2 to allow only `/dev/sd*` devices to be used by the LVM. The following command searches for and replaces the line defining which types of hardware may be used for LVM:

```
sed -i 's/\[ \"a\/\.\*\/\" \]/[ "a|sd.*|", "r|.*|" ]/' /etc/lvm/lvm.conf
```

1. Next, on Node 1 and Node 2, create a partition on the disk that will host the Zenoss LVM volumes:

   ```
   echo -e "n\np\n1\n\n\nt\n8e\nw"  | fdisk /dev/sdb
   ```

2. On Node 1 and Node 2 initialize the new partition for use by LVM:

   ```
   pvcreate /dev/sdb1
   ```

3. On Node 1 and Node 2, create the *Zenoss_data* volume group:

   ```
   vgcreate zenoss_data /dev/sdb1
   ```

4. Create the logical volumes within the new logical volume group. On Node 1 and Node 2, create a logical volume named zends.

Note: The size of our logical volumes in the example is arbitrary; carefully consider production deployments' storage needs before sizing production LVM volumes:

   ```
   lvcreate -L 5Gb -n lv_zends zenoss_data
   ```

5. On Node 1 and Node 2, create a logical volume named zenoss:

   ```
   lvcreate -L 5Gb -n lv_zenoss zenoss_data
   ```

6. On Node 1 and Node 2, create a logical volume named `zenoss_perf`:

   ```
   lvcreate -L 5Gb -n lv_zenoss_perf zenoss_data
   ```

7. On Node 1 and Node 2, create a logical volume named `zenoss_index`:

   ```
   lvcreate -L 4.95Gb -n lv_zenoss_index zenoss_data
   ```

Install DRBD For File System Replication

Next download and build the DRBD package from the source (on both Nodes). Depending on the kernel version, the drbd-km package (the kernel dependent driver for Linux) may differ; it must therefore be compiled and installed again if the kernel is upgraded subsequent to the initial DRBD installation.

1. Install the prerequisites needed to compile packages from the source:

   ```
   yum -y install gcc make automake autoconf flex rpm-build kernel-devel
   libxslt
   ```

2. Download and compile the DRBD package to create RPM installers:

   ```
   cd ~
   mkdir -p rpmbuild/{BUILD,BUILDROOT,RPMS,SOURCES,SPECS,SRPMS}
   wget http://oss.linbit.com/drbd/8.4/drbd-8.4.2.tar.gz
   tar xvfz drbd-8.4.2.tar.gz
   cd drbd-8.4.2
   ./configure
   make rpm && make km-rpm
   ```

3. You have now created RPM installers for DRBD. Next, you need to run the new RPMs to install DRBD on your systems. To do so, complete the following:

   ```
   cd /root/rpmbuild/RPMS/x86_64
   rpm -Uhv drbd-utils-[your version] example: rpm -Uhv drbd-utils-8.4.2-
   2.el6.x86_64.rpm
   rpm -Uhv drbd-km-[your version]  example: rpm -Uhv drbd-km-
   2.6.32_279.19.1.el6.x86_64-8.4.2-2.el6.x86_64.rpm
   rpm -Uhv drbd-pacemaker-[your version]  example: rpm -Uhv drbd-
   pacemaker-8.4.2-2.el6.x86_64.rpm
   ```

4. On Node 1 and Node 2, disable the DRBD service from starting automatically during system boot. You will want Corosync to handle the starting of DRBD as one of its managed *Corosync services* (*Corosync services* will be defined later when configuring Corosync on the nodes):

   ```
   chkconfig drbd off
   ```

5. Now that DRBD has been installed, configure DRBD on both Nodes. These settings define, for example, how DRBD should react in the face of a disk IO error on one of the nodes (among other settings). Replace the default contents in the file `/etc/drbd.d/global_common.conf` with the following:

   ```
   global {
   usage-count no;
   }
   common {
   handlers {
   pri-on-incon-degr "/usr/lib/drbd/notify-pri-on-incon-degr.sh;
   /usr/lib/drbd/notify-emergency-reboot.sh; echo b > /proc/sysrq-trigger ;
   reboot -f";
   pri-lost-after-sb "/usr/lib/drbd/notify-pri-lost-after-sb.sh;
   /usr/lib/drbd/notify-emergency-reboot.sh; echo b > /proc/sysrq-trigger ;
   reboot -f";
   local-io-error "/usr/lib/drbd/notify-io-error.sh; /usr/lib/drbd/notify-
   emergency-shutdown.sh; echo o > /proc/sysrq-trigger ; halt -f";
   fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
   after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
   }
   disk {
   on-io-error detach;
   fencing resource-only;
   resync-rate 300M;
   }
   }
   ```

Next, on both Node 1 and Node 2, you will need to create the DRBD Resource and to define the file system(s) DRBD will replicate across the two nodes. It's important to note that this is block level replication, so while these file systems are mounted on the operating system of the primary node (and therefore can be read and written to) the replication is occurring across the two nodes at the block level. When a replicated file system is needed on the backup node, it must be mounted on the backup node first.

We will name our `drbd` resource `r0.` Sample host names and IP addresses are used below; substitute actual values for production environments.

Note: The values detailed below correspond to the `eth1` NIC card, which is the secondary NIC card reserved for DRBD and cluster traffic. The IP addresses are the internal addresses for `eth1` to be used for replication. The value 7788 is merely an example of an unused port; substitute an alternative value in production if needed.

Note: In this single resource we are including all four LVM volumes we created earlier.

6. Create the file `/etc/drbd.d/r0.res` and populate it with the following content:

```
resource r0 {

volume 0 {
  device  /dev/drbd0;
  disk   /dev/zenoss_data/lv_zends;
  flexible-meta-disk  internal;
 }

volume 1 {
  device  /dev/drbd1;
  disk   /dev/zenoss_data/lv_zenoss;
  flexible-meta-disk  internal;
}
volume 2 {
  device  /dev/drbd2;
  disk   /dev/zenoss_data/lv_zenoss_perf;
  flexible-meta-disk  internal;
}
volume 3 {

  device  /dev/drbd3;
  disk   /dev/zenoss_data/lv_zenoss_index;
  flexible-meta-disk  internal;
}
net {
  use-rle;
}
on node1.example.com  {
  address 192.168.10.5:7788;
}
on node2.example.com  {
  address 192.168.10.6:7788;
}
}
```

7. Create the `r0` resource using the drbdadm command:

```
drbdadm create-md r0
```

8. On Node 1 and Node 2, start the DRBD service:

```
service drbd start
```

9. On Node 1, promote one DRBD host to primary:

```
drbdsetup /dev/drbd0 primary –-force
drbdsetup /dev/drbd1 primary –-force
drbdsetup /dev/drbd2 primary --force
drbdsetup /dev/drbd3 primary --force
```

10. On Node 1 and Node 2, confirm the status of DRBD:

```
drbd-overview
```

11. On Node 1, after initial synchronization has completed the following should be shown:

```
0:r0/0  Connected Primary/Secondary UpToDate/UpToDate C r-----
1:r0/1  Connected Primary/Secondary UpToDate/UpToDate C r-----
2:r0/2  Connected Primary/Secondary UpToDate/UpToDate C r-----
3:r0/3  Connected Primary/Secondary UpToDate/UpToDate C r-----
```

12. On node 2, after initial synchronization has completed the following should be shown:

```
0:r0/0  Connected Secondary/Primary UpToDate/UpToDate C r-----
1:r0/1  Connected Secondary/Primary UpToDate/UpToDate C r-----
2:r0/2  Connected Secondary/Primary UpToDate/UpToDate C r-----
3:r0/3  Connected Secondary/Primary UpToDate/UpToDate C r-----
```

13. Next, you will format the file systems on the DRBD volumes. On Node 1, the following commands would create file systems on the four LVM volumes:

```
mkfs -t ext3 /dev/drbd0
mkfs -t ext3 /dev/drbd1
mkfs -t ext3 /dev/drbd2
mkfs -t ext3 /dev/drbd3
```

14. On Node 1 and Node 2, create the mount points:

```
mkdir –p /opt/zends/data
mkdir –p /opt/zenoss
```

15. On Node 1, mount the file systems:

```
mount /dev/drbd0 /opt/zends/data
mount /dev/drbd1 /opt/zenoss
```

Note: The `/opt/zends/data` directory is explicitly chosen (instead of `/opt/zends`). The Pacemaker scripts (required as part of the Corosync Cluster Manager) can't check the program status on `/opt/zends` while it is unmounted on the secondary node.

16. On Node 1, make nested mount points:

```
mkdir -p /opt/zenoss/perf
mkdir -p /opt/zenoss/var/zeneventserver
```

17. On Node 1, mount the nested file systems:

```
mount /dev/drbd2 /opt/zenoss/perf
mount /dev/drbd3 /opt/zenoss/var/zeneventserver
```

Install Zenoss Resource Manager

Requirements

Ensure that your system meets all hardware requirements detailed in the Installation Guide. In addition, make sure that:

1. You have disabled SELinux, as instructed above.
2. The `/opt/zenoss` directory is not a symbolic link to another location.
3. The default file/directory creation mask, or the 'umask', is set to 022. This is the default for most Linux variants.
4. The `/home` directory is writable by the root user, or the `/home/zenoss` directory exists as the zenoss user's home directory.
5. The standard Yum repositories are available via an internet connection or you have access to local repositories.
6. The Linux servers hosting Zenoss have been configured to access a DNS server that can resolve the hostnames of resources you wish to monitor with Zenoss and of Zenoss component servers (if multiple servers are part of your Zenoss deployment).

Remove Conflicting Messaging Systems

Zenoss relies on the RabbitMQ messaging system. Newer versions of CentOS may include alternative messaging systems (Matahari and Qpid). You must remove these messaging systems by completing the following:

- On Node 1 and Node 2, use the following commands to determine if Matahari or Qpid packages are installed and to remove them if found:

```
rpm -qa | egrep -i 'matahari|qpid' |while read pkg; do rpm -e --nodeps
$pkg; done
```

Download the Zenoss Installation Files

1. Browse to the following URL: https://support.zenoss.com

Note: Contact your Zenoss representative for site login credentials.

2. In the Downloads area of the Home tab, locate and download the current Zenoss installation files (including the rpm installer for zends).

Install Oracle Java

OpenJDK is not supported for Zenoss, but is installed by default when Centos 6 is installed with the 'Basic Server' option. Open JDK must therefore be removed prior to installing Oracle Java.

1. To identify which version (if any) of OpenJDK is installed, run

```
yum search openjdk
```

2. If you find that OpenJDK is installed, remove it using Yum (do not copy and paste the following command, instead substitute in the value for the version installed:

   ```
   Yum –y remove [open jdk version]  (example: yum -y remove java-1.6.0-
   openjdk.x86_64)
   ```

3. Download Oracle JRE:

   ```
   wget -O jre-6u31-linux-x64-rpm.bin  \
   http://javadl.sun.com/webapps/download/AutoDL?BundleId=59622
   ```

4. Change mode:

   ```
   chmod +x ./jre-6u31-linux-x64-rpm.bin
   ```

5. Install Oracle JRE:

   ```
   ./jre-6u31-linux-x64-rpm.bin
   ```

6. Add the following line to the end of the `/etc/profile` file:

   ```
   export JAVA_HOME=/usr/java/default/bin
   ```

7. Verify the installed version:

   ```
   java -version
   ```

Install the Zenoss Dependencies Repository

- On Node 1 and Node 2, install the Zenoss dependencies repository:

  ```
  rpm -ivh http://deps.zenoss.com/yum/zenossdeps-4.2.x-1.el6.noarch.rpm
  ```

Install and Configure RabbitMQ

Use the following commands to install and configure RabbitMQ:

Note: If this procedure is adapted to a scenario with an external Zends server, then choose an alternate location for the mnesia directory.

1. On Node 1 and Node 2, install RabbitMQ:

   ```
   yum -y install rabbitmq-server-2.8.6
   ```

2. On Node 1 create and change ownership of the mnesia directory:

   ```
   mkdir -p /opt/zends/data/rabbitmq/mnesia
   chown -R rabbitmq:rabbitmq /opt/zends/data/rabbitmq/mnesia
   ```

3. On Node 1 and Node 2 change ownership of other directories used by rabbitmq:

   ```
   chown -R rabbitmq:rabbitmq  /var/lib/rabbitmq  /var/log/rabbitmq
   ```

4. On Node 1 and Node 2, disable RabbitMQ from starting automatically during reboot (Corosync must be used handle the starting of RabbitMQ):

   ```
   chkconfig rabbitmq-server off
   ```

5. On Node 1, manually start the Cluster Virtual IP before starting rabbitmq (remember to substitute your values for the sample values used in this instance):

   ```
   ip addr add 192.168.87.20/32 dev eth0
   ```

6. On Node 1 and Node 2, update the `/etc/rabbitmq/rabbitmq-env.conf` to contain the following content, updating the node name to reflect your production values:

Note: The CLUSTER_SHARED_NODENAME is the short DNS name created in the `/etc/hosts` earlier that resolves to the VIP.

```
NODENAME="rabbit@<CLUSTER_SHARED_NODENAME>"
MNESIA_BASE="/opt/zends/data/rabbitmq/mnesia"
```

7. On Node 1, start the `rabbitmq-server` daemon:

   ```
   service rabbitmq-server start
   ```

8. On Node 1, update RabbitMQ permissions, using the appropriate CLUSTER_SHARED_NODENAME value:

   ```
   rabbitmqctl -n rabbit@<CLUSTER_SHARED_NODENAME>  add_user zenoss zenoss
   rabbitmqctl -n rabbit@<CLUSTER_SHARED_NODENAME>  add_vhost /zenoss
   rabbitmqctl -n rabbit@<CLUSTER_SHARED_NODENAME>  set_permissions -p
   /zenoss zenoss '.*' '.*' '.*'
   ```

Install and Configure memcached and snmpd

Use the following commands to install and configure the memcached and snmpd daemons:

1. On Node 1 and Node 2 install memcached and net-snmp:

   ```
   yum -y install memcached net-snmp net-snmp-utils
   ```

2. On Node 1 only, start the `memcached` daemon:

   ```
   service memcached start
   ```

3. On Node 1 and Node 2 start the `snmpd` daemon:

   ```
   service snmpd start
   ```

4. On Node 1 and Node 2 disable memcached from starting automatically on reboot:

   ```
   chkconfig memcached off
   ```

5. On Node 1 and Node 2 enable snmpd on both Nodes.

   ```
   chkconfig snmpd on
   ```

Install the Zenoss DataStore

Perform the following steps on the local server:

1. On Node 1 and Node 2, install the Zenoss DataStore (again substituting your version number of the rpm):

   ```
   yum --nogpgcheck localinstall zends-[your version]
   (example: yum --nogpgcheck localinstall zends-5.5.25a-
   1.r64630.el6.x86_64.rpm)
   ```

2. On Node 1 and Node 2, disable the Zenoss DataStore from starting automatically on reboot:

   ```
   chkconfig zends off
   ```

3. On Node 1, start the data store:

   ```
   service zends start
   ```

Install and Configure Zenoss

Install the Zenoss RPM

- On Node 1 and Node 2, install the Zenoss RPM file:

```
yum -y --nogpgcheck localinstall zenoss_[your rpm version]
```

Install MySQLTuner

Complete the following to download and install the MySQLTuner Perl script, on Node 1:

1. Change to the Zenoss user, then change to the `/opt/zenoss/bin` directory:
   ```
   su - zenoss
   cd /opt/zenoss/bin
   ```
2. Retrieve the MySQLTuner script:
   ```
   wget mysqltuner.pl
   ```
3. Change read and execute access to the file:
   ```
   chmod 755 mysqltuner.pl
   ```
4. Exit back to the root user to continue the installation:
   ```
   exit
   ```

Configure the Zenoss System

1. On Node 1, set post-installation permissions as the root user:
   ```
   chown -R zenoss:zenoss /opt/zenoss
   chown -R rabbitmq:rabbitmq /opt/zends/data/rabbitmq
   ```
2. On Node 1 and Node 2, disable Zenoss from starting automatically on reboot (This is to be sure only Corosync is managing the starting and stopping of Zenoss on either Node):
   ```
   chkconfig zenoss off
   ```
3. On Node 1, as the Zenoss user, update the `/opt/zenoss/etc/global.conf.example` to refer to the clustered IP or shared node name by editing the following values as shown (substituting in the VIP IP address):
   - zodb-host localhost should change to zodb-host 192.168.87.20
   - zodb-port 3306 should change to zodb-port 13306
   - amqphost localhost should change to amqphost 192.168.87.20
   - zep-host localhost should change to zep-host 192.168.87.20
   - zep-port 3306 change to zep-port 13306
   - zep-uri http:// localhost:8084 change to zep-uri http://192.168.87.20:8084
4. On node 1, change to the Zenoss user, then enter the Zends interactive command shell:
   ```
   zends –u root
   ```
5. Grant all privileges to the root user at the hostname that resolves to the VIP, using a command such as that shown below (replacing the %, which is a wildcard that should not be used in production, with the hostname):
   ```
   grant all on *.* to 'root'@'%' with grant option;
   flush privileges;
   ```
6. Exit the Zends interactive command prompt using **CTRL-D**.
7. Exit to the root user:
   ```
   exit
   ```

8. On Node 1, as the root user, run the following command to start the system:

```
service zenoss start
```

Note: Zenoss is configured to write out key files during this stage, including creation and indexing of the ZODB database. Note that this is all occurring on the mounted, replicated file system on Node 1. In the background, DRBD is replicating all of the changes to the un-mounted, replicated file systems on Node 2.

Once the configuration and startup is complete, on Node 1, update the shared IP for the localhost hub (to ensure all Zenoss processes connect to the hub on the VIP) by completing steps 9 through 12:

9. Change to the Zenoss user:

```
su – zenoss
```

10. Invoke the zendmd interactive command shell:

```
zendmd
```

11. At the zendmd command shell, enter the following (substituting your VIP address for the sample value below):

```
dmd.Monitors.Hub.localhost.hostname  = "192.168.87.20"
dmd.Monitors.Hub.localhost._isLocalHost  = False
commit()
```

12. Enter **CTRL-D** to exit the Zendmd command line.

Optional: If you wish to monitor windows devices using WMI, install the Windows monitoring rpm by completing steps 15 through 16:

14. Exit to the root user:

```
exit
```

15. Stop Zenoss:

```
service zenoss stop
```

16. Install the Windows monitoring rpm:

```
yum -y --nogpgcheck localinstall zenoss_msmonitor-[your version]
service zenoss restart
```

Once Zenoss has started, log on to the user interface of your Zenoss instance using the VIP hostname and verify that the application is operating correctly.

If the application is running properly, you now have a partial cluster functioning. Zenoss is fully installed and operating on Node 1, reading and writing to its files on the mounted, replicating files systems. On the backup node, you have Zenoss installed, but it has written many of its dynamic files (i.e., the contents of `/opt/zenoss`) to a duplicate set of directories that will be orphaned when the second node is called up on to stand in for the first node. In subsequent steps you will be configuring Corosync to, upon a 'failover scenario', mount the file system replicated by DRBD on the operating system of the failover node. When this occurs, the act of mounting will effectively 'cover up' or 'hide' the orphaned copy of `/opt/zenoss` on the backup node, and will present to the backup node operating system the copy of `/opt/zenoss` that has been replicated by DRBD. Additionally, at this stage Zenoss has not been started on the backup node, so its orphaned version of `/opt/zenoss` does not contain the databases, indexes, or other working files necessary for Zenoss to function.

The next step in setting up the cluster will be to install and configure Corosync to handle the rest of the cluster management. This step involves defining *services* in Corosync which will be watched on the primary node and can be started – in a predefined manner and sequence – on the backup node when needed.

Note: The term *service* has two possible meanings in the context of any given Corosync cluster. Naturally our nodes continue to have their own services within the Linux operating system (indeed, the Corosync process is itself a service within Linux). However, the use of Corosync creates a new meaning of the term: anything that needs to be managed by Corosync from a failover perspective is created as a *Corosync service*. Examples might include mounting a file system, assigning the VIP to one of the Ethernet cards on a Node, or – confusingly – starting a service such as memcached on the Linux operating system of a node. In this document we will endeavor to clear up the confusion by referring distinctly to *Linux services* and *Corosync services*.

To install Corosync and begin defining Corosync services, you will need to stop Zenoss (and its dependencies) and unmount your shared file systems by completing the following steps:

1. Stop Zenoss and related processes:
   ```
   service zenoss stop
   service zends stop
   service rabbitmq-server stop
   ```
2. On Node 1 stop memcached:
   ```
   service memcached stop
   ```
3. On Node 1, unmount the file systems:
   ```
   umount /opt/zenoss/var/zeneventserver
   umount /opt/zenoss/perf
   umount /opt/zenoss
   umount /opt/zends/data
   ```
4. On Node 1 and Node 2, stop DRBD:
   ```
   /etc/init.d/drbd stop
   chkconfig drbd off
   ```
5. On Node 1, stop the virtual IP (substituting your VIP in the command):
   ```
   ip addr del 192.168.87.20/32 dev eth0
   ```

Install and Configure Corosync

Complete the following steps to install and enable the Corosync cluster manager software:

1. On Node 1 and Node 2, install the Pacemaker cluster resource manager:

   ```
   yum -y install pacemaker fence-agents resource-agents
   ```

2. On Node 1, create a cluster key:

   ```
   corosync-keygen
   ```

3. Copy the `/etc/corosync/authkey` key (created in the previous step) from Node 1 to Node 2:

   ```
   scp /etc/corosync/authkey root@node2.example.com:/etc/corosync/
   ```

4. On Node 1, create the `/etc/corosync/corosync.conf` file and populate it with the content below, edited to include your production values.

Note: Set the value of bindnetaddr to the subnet address of the interfaces on each node (in this example, 192.168.10.4 or 192.168.87.0). Also, Corosync documentation recommends the use of a minimum of two NICs with separate hardwired connections.

```
   aisexec {
   user: root
   group: root
   }
     corosync {
         user: root
         group: root

 }
 amf {
     mode: disabled
 }
     logging {
        to_stderr: yes
        debug: off
        timestamp: on
        to_logfile: no
        to_syslog: yes
     syslog_facility: daemon
 }
     totem {
         version: 2
         token: 3000
         token_retransmits_before_loss_const:  10
         join: 60
         consensus: 4000
          vsftype: none
          max_messages: 20
         clear_node_high_bit: yes
         secauth: on
    threads: 0
    # nodeid: 1234
```

```
        rrp_mode: active

       interface {

              ringnumber: 0
              bindnetaddr: 192.168.87.0
              mcastaddr: 226.94.1.1
              mcastport: 5405

       }
       interface {
              ringnumber: 1
              bindnetaddr: 192.168.10.4
              mcastaddr: 226.94.1.1
              mcastport: 5406

       }
    }
    service {
       ver: 0
       name: pacemaker
       use_mgmtd: no

    }
```

5. Copy the `/etc/corosync/corosync.conf` file (created in the previous step) from Node 1 to Node 2:

   ```
   scp /etc/corosync/corosync.conf root@Node2.example.com:/etc/corosync/
   ```

6. On Node 2, change ownership and permissions of the `/etc/corosync/authkey` key:

   ```
   chown root:root /etc/corosync/authkey
   chmod 0400 /etc/corosync/authkey
   ```

7. On Node 2, change ownership and permissions of the `/etc/corosync/corosync.conf` file:

   ```
   chown root:root /etc/corosync/corosync.conf
   chmod 0644 /etc/corosync/corosync.conf
   ```

8. On Node 1 and Node 2, enable Corosync at reboot:

   ```
   /sbin/chkconfig corosync on
   ```

9. On Node 1 and Node 2, start Corosync:

   ```
   /etc/init.d/corosync start
   ```

10. On Node 1, check to ensure nodes are online:

    ```
    crm_mon -1V
    ```

11. If both nodes are online, you should see results similar to:

    ```
    Online: [ Node1.example.com Node2.example.com ]
    ```

Define Corosync Services

You are now ready to define Corosync services. Before beginning the configuration, it's important to review some key concepts of the active-passive cluster when using Corosync. The first of these concepts is that of a Corosync service, which was alluded to earlier. A Corosync service is anything that you wish Pacemaker to monitor and for Corosync to manage in the cluster. For example, what do you need to transition from the primary to the backup node when a failure of the primary occurs? Anything that falls into that category is defined as a 'service' in Corosync. In the context of Zenoss, you will create the following Corosync services in your cluster:

1. The DRBD service (managing it on both Nodes).
2. The replicated file systems created earlier (mounting on the backup and un-mounting on the primary upon a switchover).
3. Your VIP (assigning it to `eth0` on the backup and deleting it from `eth0` on the primary upon a switchover).
4. The memcached service (starting it on the backup node and stopping it on the primary upon switchover from primary to backup).
5. The Rabbitmq service (starting it on the backup node and stopping it on the primary upon switchover from primary to backup).
6. The Zends service (starting it on the backup node and stopping it on the primary upon switchover from primary to backup).
7. The Zenoss service (starting it on the backup node and stopping it on the primary upon switchover from primary to backup).

Two additional concepts are important to review, both of which address the issue of how Corosync services relate to one another. These are:

1. *Colocation*: Two Corosync services can be defined such that they must always be located on the same node (i.e., if one service is migrated to another node, then the Corosync will automatically move the other service along with it). In the context of Zenoss, all of the Corosync services you define must be collocated on the same node. However, one could well imagine clusters being implemented for non-Zenoss scenarios where this need not be the case.
2. *Order*: Two services can be defined such that one service must be running before the second service can start. In the case of a Zenoss cluster there are several Corosync services that depend on other services to start. For example, you must mount the file systems before you start Zenoss, or Zenoss cannot access its databases. As another example, you must assign the VIP to your active node before starting Zenoss because Zenoss can't access Rabbitmq or other key internal services while they are configured to access the VIP unless the VIP is correctly assigned.

In the case of a Zenoss cluster all Corosync services must be collocated, and their order should be as follows:

DRBD Primary => mount file systems => Start VIP => Start memcached (memcached could be started in any order but it is convenient for performance reasons to start it early) **=> Start Rabbitmq => Start Zends => Start Zenoss**

Now that you have identified the list of needed Corosync services and their colocation & order, you can take two broad approaches to configuring them. One option is use the crm command line interface, while the second option is to use the LCMC graphical user interface. We will examine each in turn.

Configuring Corosync With The CRM Command

The *crm* command can be invoked interactively to complete tasks such as viewing the configuration, placing nodes in standby, etc. The entire cluster configuration can be loaded using a 'crm<<EOF' type of method. The drawback to this method, however, is that the entire cluster configuration loads all at once, meaning this method is best performed on a cluster that has yet to be configured by an administrator. Using the 'crm<<EOF' type of method, the overall syntax would be in the form:

```
crm configure<<EOF
[configuration details]
```

```
commit
EOF
```

…where the *configuration details* referenced above encompass all of the service definitions and their order / colocation. The following is a sample *crm* configuration for the cluster built in this document. Note that this configuration file could not simply be fed into a 'crm configure' command as-is without adjustments to the syntax for your production environment. Still, it is included for illustrative purposes to give the reader a sense of the syntax form:

```
node rm.motorsport.loc \
   attributes standby="off"
node rm2.motorsport.loc \
   attributes standby="off"
primitive res_Filesystem_DRBD1 ocf:heartbeat:Filesystem \
   params device="/dev/drbd1" directory="/opt/zenoss" fstype="ext3" \
   operations $id="res_Filesystem_DRBD1-operations" \
   op start interval="0" timeout="60" \
   op stop interval="0" timeout="60" \
   op monitor interval="20" timeout="40" start-delay="15" \
   op notify interval="0" timeout="60" \
   meta target-role="started"
primitive res_Filesystem_DRBD2 ocf:heartbeat:Filesystem \
   params device="/dev/drbd2" directory="/opt/zenoss/perf" fstype="ext3"
options="noatime" \
   operations $id="res_Filesystem_DRBD2-operations" \
   op start interval="0" timeout="60" \
   op stop interval="0" timeout="60" \
   op monitor interval="20" timeout="40" start-delay="15" \
   op notify interval="0" timeout="60" \
   meta target-role="started"
primitive res_Filesystem_DRBD3 ocf:heartbeat:Filesystem \
   params device="/dev/drbd3" directory="/opt/zenoss/var/zeneventserver"
fstype="ext3" options="noatime" \
   operations $id="res_Filesystem_DRBD3-operations" \
   op start interval="0" timeout="60" \
   op stop interval="0" timeout="60" \
   op monitor interval="20" timeout="40" start-delay="15" \
   op notify interval="0" timeout="60" \
   meta target-role="started"
primitive res_Filesystem_drbd0 ocf:heartbeat:Filesystem \
   params device="/dev/drbd0" directory="/opt/zends/data" fstype="ext3" \
   operations $id="res_Filesystem_drbd0-operations" \
   op start interval="0" timeout="60" \
   op stop interval="0" timeout="60" \
   op monitor interval="20" timeout="40" start-delay="15" \
   op notify interval="0" timeout="60" \
   meta target-role="started"
primitive res_IPaddr2_VIP ocf:heartbeat:IPaddr2 \
   params ip="192.168.87.20" nic="eth0" \
   operations $id="res_IPaddr2_VIP-operations" \
   op start interval="0" timeout="20" \
   op stop interval="0" timeout="20" \
   op monitor interval="10" timeout="20" start-delay="0" \
   meta target-role="started"
primitive res_drbd_r0 ocf:linbit:drbd \
   params drbd_resource="r0" \
   operations $id="res_drbd_r0-operations" \
   op start interval="0" timeout="240" \
   op promote interval="0" timeout="90" \
   op demote interval="0" timeout="90" \
   op stop interval="0" timeout="100" \
   op monitor interval="10" timeout="20" start-delay="0" \
   op notify interval="0" timeout="90" \
   meta target-role="started"
primitive res_memcached_memcached lsb:memcached \
   operations $id="res_memcached_memcached-operations" \
   op start interval="0" timeout="15" \
   op stop interval="0" timeout="15" \
   op monitor interval="15" timeout="15" start-delay="15"
primitive res_rabbitmq-server_rabbitmq lsb:rabbitmq-server \
   operations $id="res_rabbitmq-server_rabbitmq-operations" \
   op start interval="0" timeout="15" \
   op stop interval="0" timeout="15" \
   op monitor interval="15" timeout="15" start-delay="15"
mnesia_base="/opt/zends/rabbitmq/mnesia" nodename="rabbit@node"
ip="192.168.87.20" \
   meta target-role="started"
primitive res_zends_zends lsb:zends \
   operations $id="res_zends_zends-operations" \
   op start interval="0" timeout="120" \
   op stop interval="0" timeout="120" \
   op monitor interval="15" timeout="15" start-delay="15"
primitive res_zenoss_zenoss lsb:zenoss \
   operations $id="res_zenoss_zenoss-operations" \
   op start interval="0" timeout="800" \
   op stop interval="0" timeout="800" \
   op monitor interval="15" timeout="15" start-delay="15"
ms ms_drbd_1 res_drbd_r0 \
   meta clone-max="2" notify="true" interleave="true"
colocation col_res_Filesystem_DRBD2_res_Filesystem_DRBD1 inf:
res_Filesystem_DRBD2 res_Filesystem_DRBD1


colocation col_res_Filesystem_DRBD3_res_Filesystem_DRBD2 inf:
res_Filesystem_DRBD3 res_Filesystem_DRBD2


colocation col_res_Filesystem_drbd0_ms_drbd_1 inf: res_Filesystem_drbd0
ms_drbd_1:Master
colocation col_res_Filesystem_drbd0_res_Filesystem_DRBD1 inf:
res_Filesystem_DRBD1 res_Filesystem_drbd0
colocation col_res_IPaddr2_VIP_res_Filesystem_DRBD3 inf: res_IPaddr2_VIP
res_Filesystem_DRBD3
colocation col_res_memcached_memcached_res_IPaddr2_VIP inf:
res_memcached_memcached res_IPaddr2_VIP
colocation col_res_rabbitmq-server_rabbitmq_res_memcached_memcached inf:
res_rabbitmq-server_rabbitmq res_memcached_memcached
colocation col_res_zends_zends_res_rabbitmq-server_rabbitmq inf:
res_zends_zends res_rabbitmq-server_rabbitmq
colocation col_res_zenoss_zenoss_res_zends_zends inf: res_zenoss_zenoss
res_zends_zends
order ord_ms_drbd_1_res_Filesystem_drbd0 inf: ms_drbd_1:promote
res_Filesystem_drbd0:start
order ord_res_Filesystem_DRBD1_res_Filesystem_DRBD2 inf: res_Filesystem_DRBD1
res_Filesystem_DRBD2
order ord_res_Filesystem_DRBD2_res_Filesystem_DRBD3 inf: res_Filesystem_DRBD2
res_Filesystem_DRBD3
order ord_res_Filesystem_DRBD3_res_IPaddr2_VIP inf: res_Filesystem_DRBD3
res_IPaddr2_VIP
order ord_res_Filesystem_drbd0_res_Filesystem_DRBD1 inf: res_Filesystem_drbd0
res_Filesystem_DRBD1
order ord_res_IPaddr2_VIP_res_memcached_memcached inf: res_IPaddr2_VIP
res_memcached_memcached
order ord_res_memcached_memcached_res_rabbitmq-server_rabbitmq inf:
res_memcached_memcached res_rabbitmq-server_rabbitmq
order ord_res_rabbitmq-server_rabbitmq_res_zends_zends inf: res_rabbitmq-
server_rabbitmq res_zends_zends
order ord_res_zends_zends_res_zenoss_zenoss inf: res_zends_zends
res_zenoss_zenoss
property $id="cib-bootstrap-options" \
   dc-version="1.1.7-6.el6-148fccfd5985c5590cc601123c6c16e966b85d14" \
   cluster-infrastructure="openais" \
   expected-quorum-votes="2" \
     no-quorum-policy="ignore" \
     stonith-enabled="false" \
     last-lrm-refresh="1358297306"
```

Configuring Corosync Services With LCMC

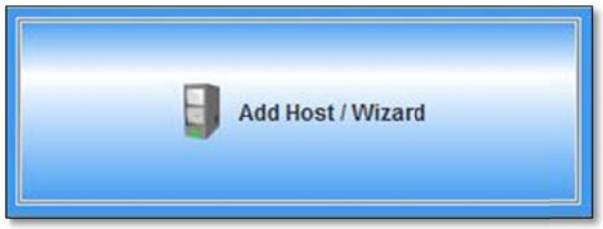
A free Java application called LCMC is available to make the process of configuring Corosync services easier (http://lcmc.sourceforge.net/). The how-to videos available at http://lcmc.sourceforge.net provide excellent guidance on how services are set up using LCMC. The first of the videos shows that many of the steps shown earlier in this guide (for example creating the LVM volumes for DRBD replication) could also have been completed using LCMC.

In order to run the application, you will need to download it to your client system (not the cluster nodes) and have Java installed on your client system. The application need not be installed on your system, but rather is run by simply opening the LCMC.jar file (or, if Java is not correctly associated with jar files on your operating system, invoking Java at the command line first, then referencing the full path to the file):

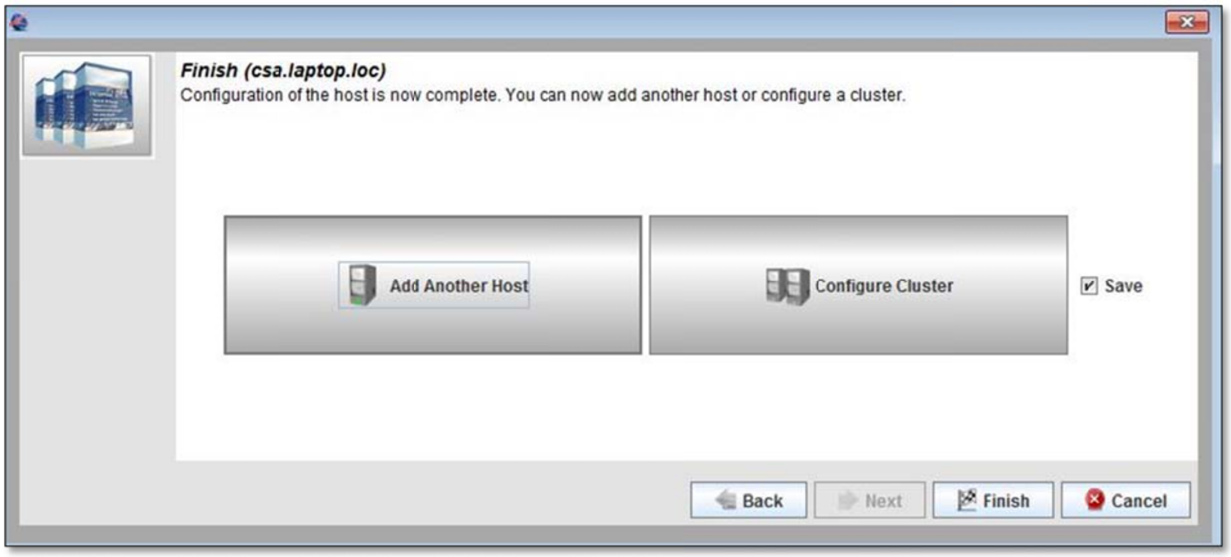
```
 java -jar [full path]LCMC.jar
```

You will need to begin by connecting LCMC to your cluster nodes to view and edit their Corosync configurations through the application.

1. Click the **Add Host/Wizard** at the top of application:

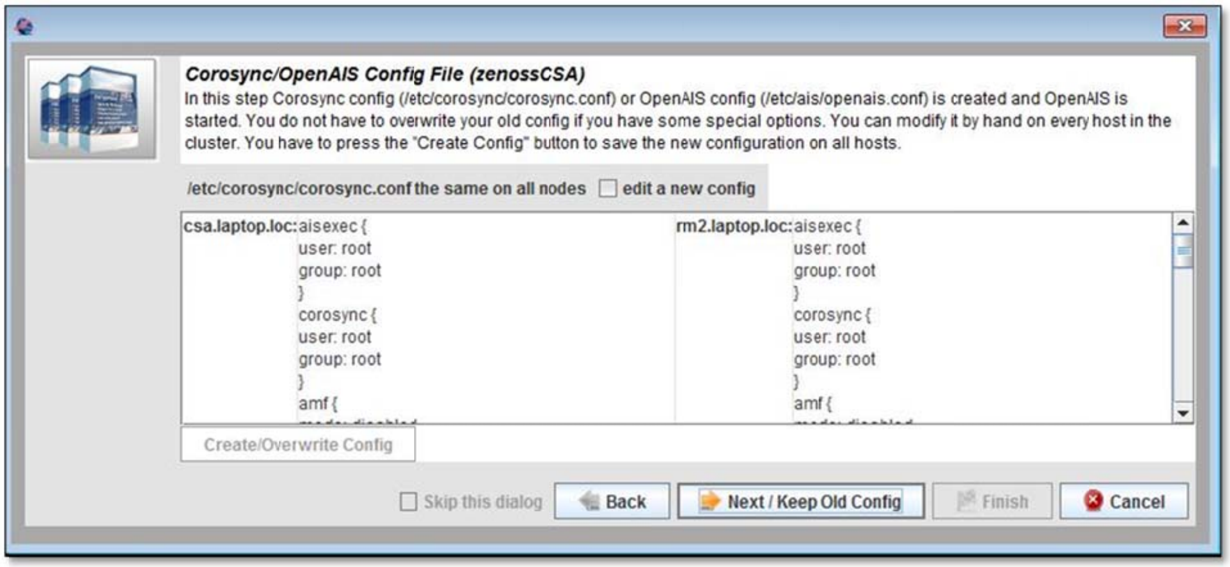


2. Complete the forms that appear in order to authenticate to both of your cluster nodes.
3. After your nodes have been added, click the **Configure Cluster** button and give your cluster a name.


Finish (csa.laptop.loc)

Configuration of the host is now complete. You can now add another host or configure a cluster.

4. Complete the following screens, and when prompted, click the **Next/Keep Old Config** button to keep the current configuration (which was created earlier at the Linux command line):


Corosync/OpenAIS Config File (zenossCSA)

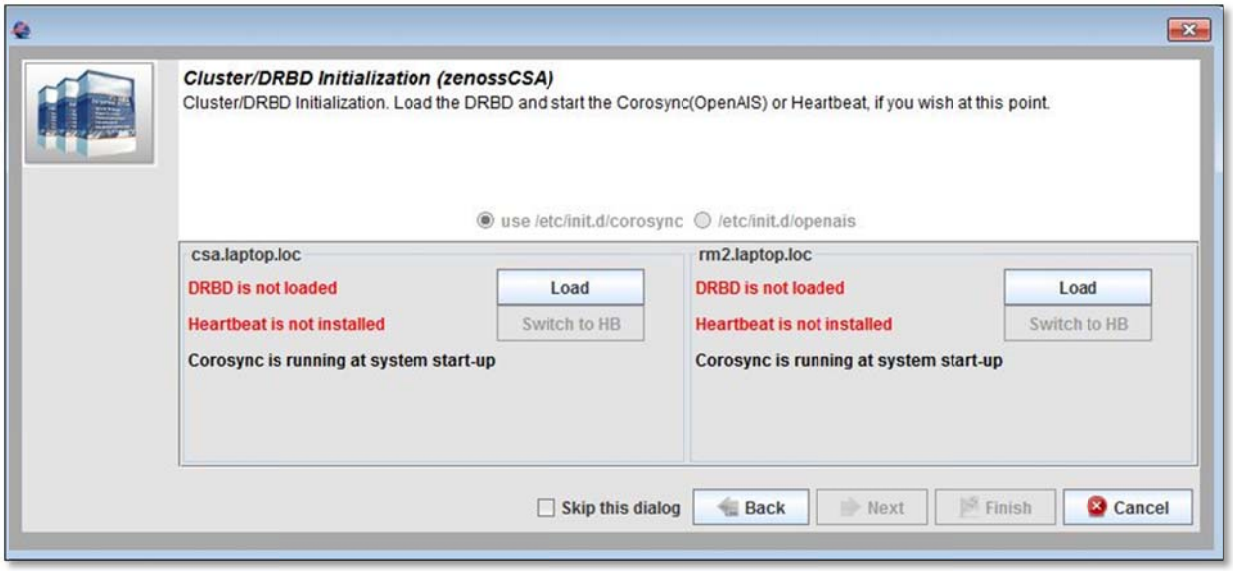
In this step Corosync config (/etc/corosync/corosync.conf) or OpenAIS config (/etc/ais/openais.conf) is created and OpenAIS is started. You do not have to overwrite your old config if you have some special options. You can modify it by hand on every host in the cluster. You have to press the "Create Config" button to save the new configuration on all hosts.

/etc/corosync/corosync.conf the same on all nodes ☐ edit a new config

```
csa.laptop.loc: aisexec {
                user: root
                group: root
                }
                corosync {
                user: root
                group: root
                }
                amf {

rm2.laptop.loc: aisexec {
                user: root
                group: root
                }
                corosync {
                user: root
                group: root
                }
                amf {
```

5. When prompted, click both **Load** buttons.


Cluster/DRBD Initialization (zenossCSA)

Cluster/DRBD Initialization. Load the DRBD and start the Corosync(OpenAIS) or Heartbeat, if you wish at this point.

[✓ use /etc/init.d/corosync] /etc/init.d/openais

csa.laptop.loc

DRBD is not loaded

Heartbeat is not installed

Corosync is running at system start-up

rm2.laptop.loc

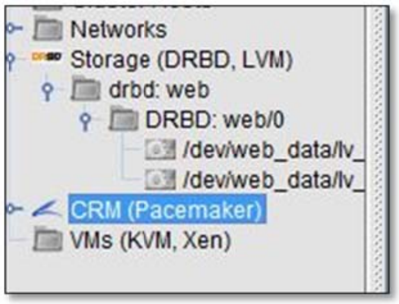
DRBD is not loaded

Heartbeat is not installed

Corosync is running at system start-up

Once these menus are completed, you are ready to begin by doing some basic Corosync configuration, then configuring Corosync services.

6. On the left, click **CRM (Pacemaker).**



7. Next, click on either of the node icons and click the **CRM Shell** button on the right. Edit the last block of the configuration to include the following:

```
cluster-infrastructure="openais" \
expected-quorum-votes="2" \
no-quorum-policy="ignore" \
stonith-enabled="false"
```

Note: It is recommended that fencing be used in production environments using Stonith, but the example examined in this document excludes it. After Stonith is configured in production, the 'Stonith-enabled' parameter specified here will have to be edited to 'true'.

8. When finished, click **Commit**.

The first step in beginning to set up our cluster Services is to designate Corosync as the manager of the DRBD resources on both Nodes.

9. Right click in the white space below the nodes to create the `r0` DRBD resource.

10. From the popup menu that appears, select **Add Service => OCF Resource Agents**.
11. Begin entering **drbd** in the search field and select **linbit:drbd** from the choices that appear.
12. In the menus at the right, enter **r0** in the drbd resource name, which is the red required field.
13. Set the target role as **started**.
14. Click **Apply** at the top.
15. Verify the **drbd status** on the drbd page of the app under Storage. Verify that all of the **drbd resources** are listed as being 'Connected' and 'Up To Date' on both Nodes.
16. Next, begin adding the Zenoss file systems as Corosync Services.
17. Click **Services** in the left pane and once again right click in the white space.
18. Select **Add Service => FileSystem + linbit:DRBD** from among the options to create the first of your file system services.
19. Once the service is created, right click the icon of the service that should precede it (in the case of file system icons this would be **DRBD**).
20. Select **Start Before** from the menu that pops up.
21. Select the new file system that was just created, leaving the colocation and order boxes both checked.
22. Next click on the new file system's icon to configure the service in the panes that appear on the right. This will be the first of four file systems that need to be created as Corosync services in this example:

```
/dev/drbd0 mounted on /opt/zends/data
/dev/drbd1 mounted on /opt/zenoss
/dev/drbd2 mounted on /opt/zenoss/perf
/dev/drbd3 mounted on /opt/zenoss/var/zeneventserver
```

Note: The DRBD 'Block Device' needs to be expressed in the form `/dev/drbd0` not in the form displayed in the drop down menu (i.e., 'DRBD r0:1', etc.).

23. Specify the mount point consistent with the DRBD resource you are creating, the file system type as **ext3**, specify any mounting options (in a production deployment one mounting option may be noatime for `/opt/zenoss/perf` and `/opt/zenoss/var/zeneventserver`) and set the 'Target Role' to **started**.
24. Then click **Apply** at the top of the right pane.

Note: If you receive an error message saying the mount point does not exist on one of the Nodes, you should click **don't create**. The mount point will not exist because it is a nested mount point that will be created only when the file systems are mounted on the backup Node.

Note: After creating each file system you may want to add a startup delay to the `r0` DRBD primary Corosync service and to each file system's Corosync service to prevent the file system mount from failing. If you do not see the choice on the menu at the right, you may want to edit the CRM configuration file for each Corosync service to add the delay (see the steps listed at the start of this section for editing the crm configuration manually). For example, the line:

```
monitor interval="20" timeout="40" start-delay="0" \
```

Becomes

```
monitor interval="20" timeout="40" start-delay="15" \
```

After creating the first file system and the DRBD service, you may want to test a failover from one node to the other before continuing to configure Corosync services. To do so, right click the active node and choose `standby / switchover`.

The DRBD primary and the first file system you created should migrate to the second Node. If the test is successful, be sure to right click the node on **Standy** and choose **go online** to bring it back online. Check to be sure that the DRBD resources revert to "Connected/Up To Date" for both nodes.

Recall the desired collocation and order between Zenoss-related Corosync services: **Start DRBD => mount file systems => Start VIP => Start memcached => Start Rabbitmq => Start Zends => Start Zenoss.** Continue by defining each of these services:

1. The VIP is created by selecting **IPaddr2** as the new Service type.
2. Memcached and RabbitMQ are created by selecting **LSB Init Scripts** as the new Service type.
3. After creation of the RabbitMQ Corosync service, you will need to stop the RabbitMQ Corosync service, then edit the CRM shell to add its specific operating parameters by adding the following lines (remembering to replace these sample values for node name and VIP with your actual production values):

   ```
   mnesia_base="/opt/zends/rabbitmq/mnesia" nodename="rabbit@rmvip" \
   ip="192.168.87.20" meta target-role="started"
   ```
4. The **Zends** service is created by selecting **LSB Init Scripts** as the new Service type. Specify a timeout of 120 seconds for Zends and a delay of 15.
5. The **Zenoss** service is created by selecting **LSB Init Scripts** as the new Service type. Specify a timeout of 800 seconds for Zenoss and a delay of 15.

Introduction to STONITH

Production clusters should always be *fenced* to be sure a malfunctioning cluster node does not endanger data integrity. Fencing is a method of remotely powering down a malfunctioning node. STONITH (“Shoot The Other Node In The Head”) is a Linux package that works well with the type of two-node cluster detailed in this document.

Unfortunately, the means of implementing STONITH will necessarily vary depending on what hardware or hypervisor solution is hosting the Zenoss nodes. For example, if the nodes are hosted on physical hardware, then STONITH needs to be configured to interact at the hardware layer via a remote access controller. Conversely, if the Zenoss nodes are virtual machines, then STONITH must be configured to interact with the hypervisor to affect node shutdowns.

The large number of possible production configurations makes it impractical to document STONITH implementation in this document. Please refer to the link below in the reference section for more on configuring STONITH.

References

Browse to these articles and samples for more information:

- Java GUI for configuration of Corosync clusters:

  http://lcmc.sourceforge.net/

- High availability with Pacemaker and DRBD:

  http://www.rabbitmq.com/pacemaker.html

- Cluster Labs STONITH Example:

  http://clusterlabs.org/doc/en-US/Pacemaker/1.1-plugin/html-single/Clusters_from_Scratch/index.html#s- stonith-example