Zenoss

The Zenoss Enablement Series:

Resource Manager Collector RHCS Guide (CentOS 5 Version)

Document Version 424-P1

Zenoss, Inc.

www.zenoss.com

Table of Contents

Applies To

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

- Resource Manager Version 4.2.4 (Build 1859)
- CentOS Linux 5.10
- VMware ESXi 5.0 hypervisor (for cluster nodes)
- Luci version 0.12.2-68.el5.centos
- Ricci version 0.12.2-68.el5.centos
- DRBD Version 8.3.15-2.el5.centos

Summary

The objective of setting up Zenoss collectors 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 system hosting Zenoss. High availability clusters can have various configurations, including, but not limited to:

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

This document describes an *Active – Passive* high availability cluster without geo diversity that uses Red Hat Cluster Suite (RCHS) and Distributed Replicated Block Device (DRBD). For our scenario, at least two identical servers per cluster 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 provision of the key Zenoss services in the event the first server fails or otherwise becomes unavailable. This solution lacks *geo diversity* in the sense that the two servers are co-located in the same facility. As such, no protection against a scenario that destroys or renders unreachable the facility hosting Zenoss is provided by this solution.

This manual provides an overview and step-by-step configuration directions to deploy highly-available Resource Manager collectors on a fast local area network with RHEL or CentOS 5.

Overview

Understanding Node Roles

Nodes within each cluster assume one of two roles; *active* or *standby*. Each cluster consists of two nodes with only one node assuming the active role at a time. The other node serves as the standby node. The active node responds to requests from users until the service is interrupted, then the roles interchange.

Replicating Data with DRBD

Instead of using shared storage, the cluster replicates data to the passive node through DRBD. DRBD eliminates shared storage as a single point of failure.

Using Multiple NICs

Although one network interface card (NIC) can suffice, two NICs are recommended for each member node when fencing is employed. One NIC is used for the public network external requests and heartbeat. The second NIC is used by DRBD for replication service. This method prevents the public network from becoming saturated by the disk replication or synchronization process.

Understanding the VIP

Each cluster has a floating Virtual IP Address (VIP) on the public network. The VIP is assigned to the active node. In case of interruption, the VIP is re-assigned to the standby node.

Managing the Cluster (Remotely)

The cluster can be managed remotely through *luci*, a web based GUI for managing RHCS. All cluster-related configurations can be performed through the luci GUI. Services can be disabled or restarted through the GUI for performing maintenance tasks. A single luci instance can be used to manage multiple clusters. For example, a single luci server could be used to manage separate clusters for Resource Manager, Resource Manager collectors, Service Impact, and Analytics.

Fencing

Red Hat Clustering requires that a cluster be configured such that a failing node can be disabled by its peer nodes to prevent the failing node from adversely affecting the cluster services. This is accomplished by a method known as *fencing* that takes advantage of fencing agents included with RHCS. For example, when cluster nodes are deployed onto virtual machines on the VMware vSphere platform, your fencing agent connects to the vSphere API to reboot the failing VM. Conversely, when nodes are hosted on physical servers, those physical servers must be fenced at the hardware level (for example via an out-of-band interface card). The large number of fencing and hardware configuration possibilities makes it impractical to document fencing for every possible scenario. Zenoss administrators must work with their IT departments to develop a fencing solution that works with their infrastructure. Although fencing is strongly recommended, administrators who cannot implement a fencing solution can refer to the '"No-Operation" Fencing' section of Appendix C: Fencing for a temporary workaround.

Naming Conventions

The following naming conventions are used in this guide. Replace the example names with the names or values in your environment. Note that all IP addresses should be statically assigned.

collector – cluster name.

COLLECTORVIP - virtual IP address of the zenoss cluster.

collector{1,2} - general name for the collector node.

collector{1,2}-PRIVATE - hostname of the collector node that resolves to its private IP address. The node should have this name as its hostname, as returned by `uname -n`.

collector{1,2}-PUBLIC - hostname of the collector node that points to its public IP address.

collector{1,2}-PUBLICIP - public IP of the collector node. It is not necessarily a public IP address, but should be reachable by its users.

collector{1,2}-PRIVATEIP - private IP of the collector node.

CLUSTERMGR - hostname of the luci cluster management interface.

The following acronyms are used in this guide:

RHCS – Red Hat Cluster Suite.

RM – Zenoss Service Dynamics Resource Manager.

luci – an agent/server architecture for remote administration of clusters.

Sample commands are prepended with prompts that indicate which user issues the command. These prompts include:

- # (pound/hash sign) - execute the command as *root*
- $ (dollar sign) - execute the command as *zenoss*

Text in sample commands might be enclosed in less than (**<**) and greater than (**>**) symbols. This indicates the text is a placeholder and the placeholder must be replaced with an actual value for your environment. Examples of text that can include placeholders are *version numbers* and *hostnames*.

Prerequisites

The following hardware requirements must be satisfied to successfully install and configure the Zenoss collectors in a high availability environment:

- One machine for the **luci server**. A single server can be used to manage multiple clusters. Ideally, this machine has the same architecture as the node systems.
- At least two identical RHEL or CentOS machines to function as **collector nodes**. Consult the *Resource Management Installation* manual for the required system specifications.
- When fencing is employed, two network interface cards per machine (except luci) with IP addresses

configured for both public and private networks. When fencing cannot be employed, all cluster traffic should go through a single network interface.

- The same architecture for all node systems. The cluster manager node and cluster nodes should have the same processor architecture (x86_64) and OS version (RHEL or CentOS 5). Because the cluster manager node configures and manages the clusters, it should share the same architecture as the node systems.
- At least two filesystems per node for the OS and Zenoss data that must be replicated.
- Optionally: a supported fencing device.
  Note: For VMware vSphere, a SOAP license is required. See the RHCS manual for a list of supported devices. See *Appendix C: Fencing* for additional information about fencing.

Consider the following prior to implementing Resource Manager collectors with Red Hat Cluster Suite:

- The host clocks must be synchronized to a time server via Network Time Protocol (NTP).
- SELinux must be disabled on all hosts because it is not supported by Zenoss.
- Nodes should be located within a single LAN with multicast support.
- Nodes must have hostname(s) that resolve to their private IP address. (for example: `collector1.private.domain.com`)
- There must be a resolvable *hostname* or *domain* name for both private and public IP addresses. If an authoritative DNS server is not available, you can add the entries to the `/etc/hosts` file for each node as shown in the example file entries below. Replace the values with actual IP addresses and hostnames in your environment:

```
COLLECTORVIP                collector
collector1-PUBLICIP         collector1-PUBLIC
collector2-PUBLICIP         collector2-PUBLIC
collector1-PRIVATEIP        collector1-PRIVATE
collector2-PRIVATEIP        collector2-PRIVATE
```

Configuration

The following sections describe the installation and configuration tasks that result in working Zenoss collectors on Red Hat Cluster Suite (RHCS).

Installing luci on the Controller Server

The cluster manager host is used to configure and manage the clusters. It is also used to create DRBD RPM packages.

Perform the following procedure to install luci as the cluster manager on the *CLUSTERMGR* host:

1. Verify that the `/etc/hosts` file does not include a record that resolves the system's FQDN to the loopback interface (127.0.0.1). Instead, only 'localhost' and / or *localhost.localdomain* should resolve to the loopback address, while the system's fully qualified domain name should resolve to its public IP address. Examples of correct hosts file records:

   ```
   127.0.0.1   locahost   locahost.localdomain
   192.168.1.100     collector1.doctest.loc
   ```

2. Update CLUSTERMGR:

   ```
   # yum update
   ```

3. Enter the following commands to ensure the CLUSTERMGR time is synchronized:

   ```
   # yum –y install ntp
   # chkconfig ntpd on
   # ntpdate pool.ntp.org
   # /etc/init.d/ntpd start
   ```

4. For setup purposes, enter the following commands to disable the internal software firewall:

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

   Note: After you identify the ports for cluster and Zenoss service communications (defined later in this guide), the firewall can be re-enabled with the appropriate ports opened.

5. Reboot the machine:

   ```
   # shutdown –r now
   ```

6. On CLUSTERMGR, install `luci` using yum:

   ```
   # yum install luci
   ```

7. Configure the luci admin user:

   ```
   # /usr/sbin/luci_admin init
   ```

8. Restart `luci`:

   ```
   # service luci restart
   ```

9. Configure `luci` to start on boot:

```
# chkconfig --level 12345 luci on
```

10. Verify that the cluster hostnames can be resolved from the cluster manager through DNS or the hosts table.

Preparing the Collector Nodes

Note: The inclusion of three nodes in each of your clusters is recommended. See the 'Building a Three Node Cluster' section of Appendix C: Fencing for more information.

All steps in this section and subsections must be completed for both primary Collector nodes unless stated otherwise. Prepare two identical machines for the collector with at least two disks (one for the operating system and one for Zenoss data). Make sure that public and private hostnames of member nodes are resolvable by each other.

Collector Node Prerequisites

Verify that the `/etc/hosts` file does not include a record that resolves the system's FQDN to the loopback interface (127.0.0.1). Instead, only 'localhost' and / or *localhost.localdomain* should resolve to the loopback address, while the system's fully qualified domain name should resolve to its public IP address. Examples of correct hosts file records:

```
127.0.0.1  locahost  locahost.localdomain
192.168.1.100    collector1.doctest.loc
```

2. For setup purposes, enter the following commands on all nodes to disable the internal software firewall:

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

Note: After you identify the communications ports for the cluster and the Zenoss service, you can re-enable the firewall with the appropriate ports opened.

3. Disable SELinux. Because SELinux is not compatible with Zenoss, you must disable it. Enter the following commands on *collector1* and *collector2*:

```
# sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Update the nodes:

```
# yum update
```

5. Enter the following commands on each node to ensure their times are synchronized:

```
# yum -y install ntp
# chkconfig ntpd on
# ntpdate pool.ntp.org
# /etc/init.d/ntpd start
```

6. Reboot each machine:

```
# shutdown -r now
```

Creating the LVM Disks

The following procedure describes how to create the LVM disk for each node. The commands in the procedure assume:

- The disk dedicated to the LVM volume is located on `/dev/sdb`.
- The volume group is *zenoss_data*.
- The logical volume is *lv_zenoss.* The logical volume is used for Zenoss application data and performance data.

Perform the following procedure to create LVM disks on two nodes:

1. Issue the following command to create a partition on `/dev/sdb` using the `fdisk` utility:

   ```
   # fdisk /dev/sdb
   ```

2. Create the disk partition on `/dev/sdb` and tag the disk as **LVM partition (8e)**. Use the following sequence to create the first LVM partition from the first to last block:

   ```
   n,p,1,<enter>,<enter>,t,8e,w
   ```

3. Create the *zenoss_data* volume group:

   ```
   # pvcreate /dev/sdb1
   ```

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

   Note: When the pvcreate command is run, you may see an error similar to the following:

   ```
   "dev_is_mpath: failed to get device for 8:17”
   ```

   The error can be safely ignored.

4. Create the logical volume:

   ```
   # lvcreate -L <size> -n lv_zenoss zenoss_data
   ```

Creating the DRBD Resource

Perform the following procedure to create the DRDB resource:

1. Install DRBD version 8.3:

   ```
   # yum –y install drbd83.x86_64 kmod-drbd83.x86_64
   ```

2. Add the following lines to the /etc/drbd.conf file:

   ```
   include "/etc/drbd.d/global_common.conf";
   ```

   ```
   include "/etc/drbd.d/*.res";
   ```

3. Replace the `/etc/drbd.d/global_common.conf` file with the following configuration:

   ```
   global {
   usage-count no;
   }
   common {
   ```

```
protocol C;

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";

}

disk {

  on-io-error detach;

}

syncer {

   rate 300m;

}

}
```

Note: The global DRBD configuration shown here intentionally omits fencing directives to prevent multiple fencing. Multiple fencing can cause an infinite reboot loop. Fencing is handled by RHCS instead of DRBD.

4. Create the */etc/drbd.d/r0.res* file containing the following configuration. Replace collector*{1,2}-PRIVATE* and collector*{1,2}-PRIVATEIP* with actual private hostnames and IP addresses, respectively. When fencing is not possible, substitute the PUBLICIP value for the single NIC being used.

   Note: The hostnames must be consistent with the output of the command '`uname –n`' on all nodes. If they are not, you will encounter the following error:

   ```
   r0 not defined in your config (for this host)
   ```

Use the following information for the configuration:

```
resource r0 {
    device /dev/drbd0;
    disk /dev/zenoss_data/lv_zenoss;
    flexible-meta-disk internal;
on [Collector1] {
    address [Collector1-PUBLICIP]:7788;
    }
on [Collector2] {
       address [Collector2-PUBLICIP:7788;
    }
}
```

5. Create the resource *r0* on both nodes:

   ```
   # drbdadm create-md r0
   ```

6. Verify that the resource on the other node is configured before starting the DRBD service.
7. Start the DRBD service:

   ```
   # service drbd start
   # chkconfig --level 12345 drbd on
   ```

8. You might need to invalidate a node before specifying one as primary. Because there is no valid data on the disk yet, you can choose any node to invalidate. Issue the following command to invalidate *r0*:

   ```
   # drbdadm invalidate r0
   ```

   Allow the nodes enough time to synchronize their disks. Check the status by running:

   ```
   # drbd-overview
   ```

   Continue to the next step when you see the following output from `drbd-overview` for each of the volumes:

   ```
   Connected Secondary/Secondary UpToDate/UpToDate C
   ```

9. Make collector*1* the primary node. Run the following command on collector*1*:

   ```
   # drbdadm primary r0
   ```

10. On collector*1*, initialize the filesystems for the DRBD disk:

    ```
    # mkfs -t ext3 /dev/drbd0
    ```

    Note: It is not necessary to initialize the file systems on the second node because the action is replicated automatically.

11. On both nodes, create the `/opt/zenoss` mount point:

    ```
    # mkdir -p /opt/zenoss
    ```

12. On collector*1*, mount **drbd0** on `/opt/zenoss`:

    ```
    # mount /dev/drbd0 /opt/zenoss
    ```

Installing Collector Dependencies

Perform the following procedure to install the collector dependencies. Complete each step on both nodes unless instructed otherwise.

1. Confirm that **OpenJDK** is not installed. If it is installed, remove it before proceeding.
2. Install Java SE Runtime Environment version 6:
   a. Download the self-installing RPM of Oracle Java SE Runtime Environment 6u31 from the Java SE 6 Downloads page. The file to download is *jre-6u31-linux-x64-rpm.bin*.
   b. Make the RPM installer executable:

```
# chmod +x /path-to-installer/jre-6u31-linux-x64-rpm.bin
```

   c. Start the installer:

```
# /path-to-installer/jre-6u31-linux-x64-rpm.bin
```

   d. Add the following line to the end of the */etc/profile* file to append the JAVA_HOME environment variable to it:

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

   e. Verify the installed version is correct (1.6 Update 31):

```
# java –version
```

3. Install the Zenoss dependencies repository:

```
# rpm -ivh http://deps.zenoss.com/yum/zenossdeps-4.2.x-1.el5.noarch.rpm
```

4. Install Memcached and Net-SNMP:

```
# yum -y install memcached net-snmp net-snmp-utils redis rrdtool-1.4.7
```

5. *On collector1* only, start the `memcached` daemon:

```
# service memcached start
```

6. Start the `snmpd` daemon on both nodes:

```
# service snmpd start
```

7. Prevent `memcached` from starting automatically on reboot:

```
# chkconfig memcached off
```

8. Configure Net-SNMP to start on boot:

```
# chkconfig snmpd on
```

9. Install Nagios Plugins:

```
# yum -y install nagios-plugins nagios-plugins-dig nagios-plugins-dns \
nagios-plugins-http nagios-plugins-ircd nagios-plugins-ldap nagios-plugins-tcp
\
nagios-plugins-ntp nagios-plugins-perl nagios-plugins-ping nagios-plugins-rpc
```

10. Install the Zenoss data store:

```
# yum --nogpgcheck localinstall zends-[version].el5.x86_64.rpm
```

11. Configure the Zenoss data store not to start:

```
# chkconfig zends off
```

12. Install the Resource Manager rpm file:

```
# yum --nogpgcheck localinstall zenoss_[resmgr-version].el5.x86_64.rpm
```

13. Configure the Zenoss daemons not to start:

```
# chkconfig zenoss off
```

14. Set a new password for the zenoss user for use in later procedures:

```
# passwd zenoss
```

15. Change to the *zenoss* user and enter the following commands:

```
$ cd $HOME
$ mkdir .ssh
$ chmod 700 .ssh
$ touch .ssh/authorized_keys
$ chmod 600 .ssh/authorized_keys
```

16. Log in to the master host as user *zenoss* and enter the following commands. Substitute the fully-qualified domain for each collector cluster member:

```
cat $HOME/.ssh/id_rsa.pub | ssh -l zenoss \
[collector1] "cat - >> /home/zenoss/.ssh/authorized_keys"
cat $HOME/.ssh/id_rsa.pub | ssh -l zenoss \
[collector2] "cat - >> /home/zenoss/.ssh/authorized_keys"
ssh –i $HOME/.ssh/id_rsa zenoss@[collector1] (to test login & add a record to
known_hosts)
ssh –i $HOME/.ssh/id_rsa zenoss@[collector2] (to test login & add a records to
known_hosts)
```

17. On *collector1,* assign the COLLECTORVIP to the eth0 network interface as the *root* user:

```
# ip addr add <COLLECTORVIP>/32 dev eth0
```

18. From the master server, as user *zenoss*, log on to the collector1 using the COLLECTORVIP as the host to add a record to known hosts:

```
ssh –i $HOME/.ssh/id_rsa zenoss@<COLLECTORVIP>
```

19. Log in to the Resource Manager console interface as a user with ZenManager or Manager permissions.
20. Click **Advanced > Collectors**. The console displays the deployed hubs and collectors.
21. Select the hub through which the new collector's data will flow by clicking on its name.
22. From the action menu next to Zenoss Collectors, select **Add Collector....** The *Add Collector* page displays.
23. Select **zenoss user SSH keys** under *Install remotely*.
24. Enter a logical name for the collector. This name is used as the prefix for Resource Manager daemons on the collector host.
25. Enter the value of the COLLECTORVIP for the *Host* field. Click **Add Collector**.

Local Zenoss Init Scripts

For the cluster manager to successfully fail over a collector instance, the local storage on each cluster member must contain several files that are present in the replicated (DRBD) version of `/opt/zenoss` but are not present on the inactive node's filesystem. These files are required by the Zenoss init script to successfully report on its `stop` and `status` commands:

```
/etc/init.d/zenoss stop
/etc/init.d/zenoss status
```

To meet this need you can perform a one-time `rsync` of the `/opt/zenoss/` directory from the DRBD volume to the local storage on each node after the initial installation. Perform the following commands as the *root* user, unless otherwise instructed:

1. With collector*1* as the active cluster node, issue the following command on *collector1*:

   ```
   # rsync -avz /opt/zenoss/ collector2:/opt/zenoss
   ```

2. On *collector1*, stop Zenoss:

   ```
   # service zenoss stop
   ```

3. On *collector1*, unmount the *drbd0* filesystem:

   ```
   # umount /opt/zenoss
   ```

4. On *collector1*, make *collector1* the *drbd secondary* for the `r0` resource:

   ```
   # drbdadm secondary r0
   ```

5. On *collector2*, make *collector2* the *primary* for the `r0` resource:

   ```
   # drbdadm primary r0
   ```

6. On *collector2*, mount the `drbd0` filesystem:

   ```
   # mount /dev/drbd0 /opt/zenoss
   ```

7. On *collector2*, copy the contents of the drbd volume to *collector1*:

   ```
   # rsync -avz /opt/zenoss/ collector1:/opt/zenoss
   ```

8. On *collector2*, unmount the /`opt/zenoss` filesystem:

   ```
   # umount /opt/zenoss
   ```

9. On *collector2*, make *collector2* the *drbd secondary* for the `r0` resource:

   ```
   # drbdadm secondary r0
   ```

10. On *collector1*, make *collector1* the *primary* for the `r0` resource:

    ```
    # drbdadm primary r0
    ```

11. On *collector1*, mount **drbd0** on `/opt/zenoss:`

    ```
    # mount /dev/drbd0 /opt/zenoss
    ```

12. Start `zenoss` on *collector1* as the *root* user:

    ```
    $ service zenoss start
    ```

Understanding Cluster Functionality

If the application is running properly, the cluster has partial functionality. Zenoss collector daemons are fully installed and operating on *collector1*. The daemons are reading and writing files on the mounted, replicating file system.

RHCS must be configured to mount the file system replicated by DRBD on the operating system of the failover node when a failure occurs. This is necessary because Zenoss collector daemons have not been started on the backup node and its version of `/opt/zenoss` does not contain the working files necessary for Zenoss to function. Upon a failover, mounting the replicated file system hides an orphaned copy of `/opt/zenoss` on the backup node by mounding the drbd volume over it. The operating system then uses the DRDB copy of the replicated `/opt/zenoss` files that contain the necessary information for Zenoss.

Configuring the Collector Cluster

Perform the following procedure as *root* to configure the cluster:

1. On all zenoss nodes, install *rgmanager*, *ricci*, and *cman*:

   ```
   # yum install rgmanager ricci cman
   ```

2. Set a password for the user *ricci:*

   ```
   # passwd ricci
   ```

3. Configure ricci, cman, rgmanager and modclusterd to start on boot:

   ```
   # chkconfig --level 12345 ricci on
   # chkconfig --level 12345 cman on
   # chkconfig --level 12345 rgmanager on
   # chkconfig --level 12345 modclusterd on
   ```

4. Start ricci on all nodes:

   ```
   # service ricci start
   ```

5. Browse to **https://CLUSTERMGR:8084** and login to luci as *admin*.

Configuring the Cluster

Perform the following procedure to configure the cluster:

1. Click on the **Cluster** tab, then click **Create a New Cluster**.
2. In the **Cluster Name**, enter *collector*.
3. Enter the *Node Hostnames* (public hostname), root passwords.
4. Ensure that the node name is a resolvable hostname and resolves to its public IP address.
5. Leave **Other Fields** as *default*.
6. Click **Submit**.
7. If you need to modify a node attribute, click the **Nodes** tab.

Creating a Failover Domain

Perform the following procedure to create a failover domain:

1. On the left of the screen, click **Failover Domains**.
2. Click **Add a Failover Domain**.
3. Enter **collector_domain** in the *Name Field*.
4. Check **Prioritized**.
5. Check the member nodes, in this example, *collector{1,2}-PUBLIC*.
6. Set the **Priority** of *collector1-PUBLIC* to **1** and *collector2-PUBLIC* to **2**. This means that *collector1*-PUBLIC has the higher priority of the two.
7. Click **Submit**.

Creating Resources & Service Groups

Because this version of Luci is not configured to manage DRBD, the easiest way to configure cluster resources is to edit the `/etc/cluster/cluster.conf` file manually.

Note: The `cluster.conf` file can be edited on any node in the cluster. It will then be synchronized across nodes when commands detailed below are run.

When the cluster and failover domain have been created in Luci, the `cluster.conf` file will be populated with the appropriate content. For example, the *cluster* and *failover* sections of the file may look similar to this:

```
<cluster alias="collector" config_version="17" name="collector">
        <fence_daemon clean_start="0" post_fail_delay="0" post_join_delay="3"/>
        <clusternodes>
                <clusternode name="collector1.doctest.loc" nodeid="1"
votes="1">
                        <fence/>
                </clusternode>
                <clusternode name="collector2.doctest.loc" nodeid="2"
votes="1">
                        <fence/>
                </clusternode>
        </clusternodes>
        <cman expected_votes="1" two_node="1"/>
        <fencedevices/>
        <rm>
                <failoverdomains>
                        <failoverdomain name="collector_domain" nofailback="0"
ordered="1" restricted="0">
                                <failoverdomainnode
```

```
name="collector1.doctest.loc" priority="1"/>
                                  <failoverdomainnode
name="collector2.doctest.loc" priority="2"/>
                        </failoverdomain>
                </failoverdomains>
```

The next section that must be added is the *resources* section, with each resource listed within the section. The file format represents resource dependencies with (i) indentation, and (ii) by enclosing a list of resources within the open and close tags of another resource. For the collector cluster, a *resources* section should be created that has the following resources, each of which serves as dependency of the service before it:

```
drbd => /opt/zenoss filesystem => cluster VIP => memcached service => zenoss service
```

The *resources* section of `cluster.conf` takes the following form, with sample values contained in brackets ("[ ]").

```
<resources/>
               <service autostart="1" domain="[DOMAIN NAME CREATED ABOVE]"
exclusive="0" name="[NAME CREATED ABOVE]" recovery="relocate">
                   <drbd name="[NAME OF DRBD RESOURCE]" resource="[NAME OF DRBD
RESOURCE]"
                      <fs device="[DRBD DEVICE]" force_fsck="0"
force_unmount="0" fsid="28367" fstype="ext3" mountpoint="/opt/zenoss"
name="zenoss_dir" options="defaults" self_fence="0"/>
                        <ip address="[COLLECTORVIP" monitor_link="1"/>
                            <script file="/etc/init.d/memcached"
name="Memcached_init"/>
                               <script file="/etc/init.d/zenoss"
name="zenoss_init"/>
                </drbd>
                </service>
```

The following example *resources* section substitutes in sample values:

```
<resources/>
               <service autostart="1" domain="collector_domain" exclusive="0"
name="zenoss_collector" recovery="relocate">
                   <drbd name="r0" resource="r0">
                      <fs device="/dev/drbd0" force_fsck="0" force_unmount="0"
fsid="28367" fstype="ext3" mountpoint="/opt/zenoss" name="zenoss_dir"
options="defaults" self_fence="0"/>
                        <ip address="10.177.215.237" monitor_link="1"/>
```

```
                            <script file="/etc/init.d/memcached"
name="Memcached_init"/>
                               <script file="/etc/init.d/zenoss"
name="zenoss_init"/>
                </drbd>
                </service>
```

The following example shows a complete sample `cluster.conf`. Note that the *resources* section follows directly after the *failoverdomains* section:

```
<?xml version="1.0"?>
<cluster alias="collector" config_version="17" name="collector">
        <fence_daemon clean_start="0" post_fail_delay="45"
post_join_delay="45"/>
        <clusternodes>
                <clusternode name="collector1.doctest.loc" nodeid="1"
votes="1">
                        <fence/>
                </clusternode>
                <clusternode name="collector2.doctest.loc" nodeid="2"
votes="1">
                        <fence/>
                </clusternode>
        </clusternodes>
        <cman expected_votes="1" two_node="1"/>
        <fencedevices/>
        <rm>
                <failoverdomains>
                        <failoverdomain name="collector_domain" nofailback="0"
ordered="1" restricted="0">
                                <failoverdomainnode
name="collector1.doctest.loc" priority="1"/>
                                <failoverdomainnode
name="collector2.doctest.loc" priority="2"/>
                        </failoverdomain>
                </failoverdomains>
                <resources/>
               <service autostart="1" domain="collector_domain" exclusive="0"
name="zenoss_collector" recovery="relocate">
                   <drbd name="r0" resource="r0">
                      <fs device="/dev/drbd0" force_fsck="0" force_unmount="0"
```

```
fsid="28367" fstype="ext3" mountpoint="/opt/zenoss" name="zenoss_dir"
options="defaults" self_fence="0"/>
                        <ip address="10.177.215.237" monitor_link="1"/>
                            <script file="/etc/init.d/memcached"
name="Memcached_init"/>
                               <script file="/etc/init.d/zenoss"
name="zenoss_init"/>
                </drbd>
                </service>
        </rm>
</cluster>
```

Important: When the `cluster.conf` file is edited manually, as opposed to being modified via the luci GUI, the `config_version` value in the second sentence of the file must be iterated to the next highest value before the edited file is saved. For example, the following phrase must be edited to iterate the `config_version` value (changing 17 →18):

```
<cluster alias="collector1" config_version="17" name="collector1">
```

To become:

```
<cluster alias="collector1" config_version="18" name="collector1">
```

This value is used by the cluster manager software to update the configuration on all of the nodes in the cluster. When making an initial edit to the cluster.conf file, it may also be handy to increase the `post_fail_delay` and `post_join_delay` values to 45 seconds. This will provide a delay on failover to allow for brief network interruptions. Additionally, it will prevent a node from failing to transfer back from one node to the other automatically if the zenoss daemons take so long to stop that the `/opt/zenoss` directory fails to unmount.

After completing each iteration of edits and saving the file, run the following commands to update the other nodes:

```
# ccs_tool update /etc/cluster/cluster.conf
# cman_tool version -r [config_version]
```

At this stage, the status of the cluster's status can be checked:

1. Click on the **Cluster** tab of the luci UI.
2. Select the *cluster name.*
3. Click **Services** on the left.
   The new service will display on the right together with its status. Note that only the running status of the service (*running* or *not running*) displays through the luci UI. More detailed information on the status of the service can be found by running the following at the command line of either node:

   ```
   clustat
   ```

Appendix A: Failure Modes

The following sections describe the various failure modes for the system.

Node Failure Types

The node failure types include:

- Active Node Failure
- Passive Node Failure

Active Node Failure

In the case where the active node suffers an outage, the cluster shifts all resources to the other (backup) node. Because the DRBD provides a synchronous replica of the cluster data to the backup node, the cluster can continue to serve from there.

To test node failure scenario, power off the machine and watch/check the syslog output. The service should relocate to the backup node.

When the active node resumes, the cluster shifts back to that node because it has the higher failover priority.

Passive Node Failure

If the passive node suffers an outage, no action from the cluster is expected. However, the cluster should recognize the passive node when it comes back online.

Network Failure Types

The network failure types include:

- Public Network Link Failure
- Private Network Link Failure

Public Network Link Failure

Because the cluster monitors the public link, a down link also results in relocation of all resources to the backup node. The scenario can be simulated by unplugging the public interface cable. The result is both nodes will attempt to fence each other. However, only the healthy node can shut down the machine. See *Appendix C: Fencing* for additional information about fencing.

Private Network Link Failure

The private network is only used for DRBD replications. When private network of any node is down, nothing is expected from the cluster. The service should run normally without any failover. DRBD replication will resume replication upon network recovery.

Other Failure Types

The following sections describe the various other failure types, including:

- Storage Subsystem Failure
- Daemon Failure

Storage Subsystem Failure

When a node mirrored hard disk fails, the DRBD of the unhealthy node will try to shut down itself. This in turn behaves in a similar way as a node failure. In a VMware environment, simulate this by removing the virtual disk that contains Zenoss data.

Daemon Failure

The active node periodically runs service status command. If the daemon is found to be not running, all resources will be relocated to the backup node. This scenario can be tested by stopping/killing the daemon.

Appendix B: Known Errors

The following are known errors for this high availability deployment:

- Upon an attempt to start, `cman` reports:
  `Can't determine address family of nodename`.
  To troubleshoot, ensure that the node name of the node is resolvable from itself. Add the hostname into the `/etc/hosts` file if you are not using DNS.
- Resources are not relocated.
  Relocation can fail if fencing an unhealthy node is unsuccessful. See *Appendix C: Fencing* for additional information about fencing. Verify the fencing parameters are correct. It is also possible that the secondary node is unable to start a resource (DRBD, IP, etc.). Consult the `/var/log/messages` file for hints.

Appendix C: Fencing

Fencing is an automated means of isolating a node that appears to be malfunctioning to protect the integrity of the DRBD volume(s).

We recommend placing the fencing device on the public network. The reason for placing the fencing device on the public network is explained by how fencing and communication work in various implementations. In this case the two implementations are a *public* versus a *private* network interface for fencing communications.

Note: Fencing is required by Red Hat. If you do not define a fencing method, your failover will fail with the error `no method defined` when the cluster attempts to fail over to the backup node.

Using a Private Network Interface

Although it is possible to pass heartbeat communications through the private network interface, it requires a complex fencing mechanism. (See the *Implementing/Deploying Quorum Disk on RHCS* manual for more information.) The complex fencing mechanism is prone to issues. Consider, for example, a heartbeat communication that passes through the private network interface. If the private network link fails for either of the nodes, heartbeat communications fail. Each node perceives the other as offline although the active node is still online from the point of view of the user. Because the nodes perceive each other to be off line, each machine initiates fencing of the other node. Both nodes can access the fencing device because the public network links of both nodes are still functioning. The fencing device successfully fences, or shuts down both machines. Fencing both nodes leaves the cluster without a healthy, functioning node online. The complicated work around for this scenario is:

1. Move the fencing device to the private network.
2. Renumber the fencing device.
3. Reconfigure systems that use the fencing device.

Using a Public Network Interface

If heartbeat communications pass through the public network and the public network link for a node goes down, both nodes still try to fence each other. The difference in this case however is that the node with the down public network link cannot communicate with the fencing device. This means that the healthy node can fence the unhealthy node, but the unhealthy node cannot fence the healthy node.

Building a Three Node Cluster

Your cluster will become more robust and resilient if you add a third node to the cluster. The third node does not need to host resources and can act only as a voting member of the quorum. This means that if the cluster heartbeat link between the nodes is interrupted, one of the nodes will still be able to reach the third member to achieve a cluster quorum vote (>50% of the nodes are required for quorum). The failing node will detect that it is isolated from the other two and does not have a quorum. It will determine that it is no longer eligible to host resources and must shut down any running resources.

Preparing and Adding the Third Cluster Node

To prepare the third node for a cluster, perform the following as *root*:

1. Verify that the `/etc/hosts` file does not include a record that resolves the system's FQDN to the loopback interface (127.0.0.1). Instead, only 'localhost' and / or localhost.localdomain should resolve to the loopback address, while the system's fully qualified domain name should resolve to its public IP address. Examples of correct hosts file records would be:

   ```
   127.0.0.1  locahost  locahost.localdomain
   192.168.1.100    node3.doctest.loc
   ```

1. Update the node:

   ```
   # yum update
   ```

2. Enter the following commands to ensure the node's time is synchronized:

   ```
   # Yum install ntp
   # chkconfig ntpd on
   # ntpdate pool.ntp.org
   # /etc/init.d/ntpd start
   ```

3. For setup purposes, enter the following commands to disable the internal software firewall:

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

   Note: After you identify the ports for cluster communications, the firewall can be re-enabled with the appropriate ports opened.

4. Reboot the machine:

   ```
   # shutdown -r now
   ```

5. Install `rgmanager` , `ricci` , and `cman` :

   ```
   # yum install rgmanager ricci cman
   ```

6. Set a password for the user *ricci* .
   This password will be used by the cluster manager to access to node.

   ```
   # passwd ricci
   ```

7. Configure `ricci`, `cman`, `rgmanager` and `modclusterd` to start on boot:

   ```
   # chkconfig --level 12345  ricci on
   # chkconfig --level 12345  cman on
   # chkconfig --level 12345  rgmanager on
   # chkconfig --level 12345 modclusterd on
   ```

8. Start `ricci` on the node:

   ```
   # service ricci start
   ```

9. Browse to https://CLUSTERMGR:8084 and login to luci as *admin*.

10. To add the third node to a cluster, perform the following:
    a. Navigate to the desired Cluster in the Cluster Manager UI.
    b. Click **Nodes** on the left.
    c. Click **Add a Node** and enter the details for the third node. Click **Submit**.
    d. Navigate to **Failover Domains** and select your failover domain.
    e. Select the checkbox next to **Restrict failover to this domain's members** to make the cluster resources only run on selected servers.
    f. Make sure that the third node is not selected as a cluster member and does not have a priority.
    g. Click **Submit**.

"No-Operation" Fencing

Zenoss strongly suggests that you use a fencing mechanism. In the event that you do not have and cannot obtain any fencing options, you can create a fencing script that can be executed by the cluster manager but performs no actions other than returning `success`. This "No Operation" fencing option bypasses the failure of "no method defined" that will be encountered upon a failover when no fencing is configured. Please note that this is not the recommended configuration, but can be used as a workaround until functional fencing can be configured.

The No-Operation fencing option has the following requirements:

- You must have a cluster with three quorum votes/nodes.
- The cluster heartbeat and drbd traffic must use the same network interface so a node that fails is completely isolated.
- You must manually reboot the failing node to enable it to rejoin the cluster after a failure.

To configure the temporary No-Operation fencing workaround, perform the following:

1. Create a new script on each node with the following content and save it as `/sbin/noFence`:

```
#!/bin/bash
echo "success: noFence"
exit 0
```

2. Make the file executable:

```
# chmod +x /sbin/noFence
```

3. On each node, rename the existing VMware Soap fencing script and replace it with a symbolic link to the new `noFence` script:

```
# mv /sbin/fence_vmware_soap /sbin/fence_vmware_soap.orig
# ln -s /sbin/noFence /sbin/fence_vmware_soap
```

4. Browse to https://CLUSTERMGR:8084 and login to luci as *admin*. Navigate to the desired cluster and click on **Shared Fence Devices**.
5. Click **Add a Fence Device** and create a new *Fence Device.*
6. In the dropdown menu for "Select a shared fence device" choose **VMware Fencing.**
7. Complete the form with these values:

```
Name: noFence
Hostname: noFence
IP port: 22
Login: noFence
Password: noFence
```

8. Click **Add this shared fence device.**
9. Add the fence instance to each node in the *Nodes* section of the *Cluster*:
    a. Under the cluster name, click on **Nodes**.
    b. Click the desired **Node** name.
    c. Click **Add a fence device to this level**.
    d. From the **Use an existing Fence Device** dropdown, choose noFence.
    e. Populate the **Virtual Machine UUID** field with the value *noFence*.
    f. Click **Update main fence properties** at the bottom.

g. Repeat for all other nodes.

With the No-Operation fencing successfully configured, upon failure, when the live node attempts to fence the failing node it immediately succeeds without triggering any actions. These steps should be reversed and functional fencing should be configured as soon as possible.

Appendix D: Handling DRBD Errors

For information about handling DRDB errors, see *The DRBD User's Guide* at http://www.drbd.org/users-guide/. For information on resolving split brain issues specifically, see http://www.drbd.org/users-guide/s-resolve-split-brain.html.