RedHat Cluster Environment

Authors:
Jennifer Beattie
Edmund Sutcliffe

Summary

This document demonstrates the integration process for using RedHat Cluster Suite tools in a diskless PXE environment. We provide setup steps and a simple service administration example for HTTP.

We also include some simple introductory notes on the fundamentals of the RedHat Cluster Suite, to motivate use of these techniques in an automatically provisioned infrastructure.

Overview

The Red Hat Cluster Suite (RHCS) bundles open source components: OpenAIS to provide high availability / service failover features, and Linux Virtual Server (LVS) to provide IP load balancing services. These components thus come bundled with every CentOS image, and in fact are available in other Linux distributions too.

This document concentrates on the group management and service failover features, and the related cluster management tools and daemons. IP-based load-balancing with LVS is not discussed here.

We largely focus on the command-line or manual methods of performing cluster administration, and mention the GUI driven methods in passing, in order to illustrate using these tools in an automatically provisioned infrastructure.

Once the relevant packages are installed into the target machine image, we describe the configuration of fence devices and the options available. In addition we walk through the steps and daemons involved in starting a cluster and making it quorate.

The control of resources on top of this cluster is a distinct process in the operation of a cluster, so we describe this in a separate section, along with some example management commands.

Finally, we provide a brief discussion of configuring additional nodes and resources, updating the cluster information at runtime, and performing failover.

For additional information, we recommend the **RedHat Enterprise Linux Cluster Suite Overview** and **RedHat Enterprise Linux Cluster Administration** documents, plus a small list of other internet resources. Links to these may be found in the References section.

Prerequisites

You must have a build machine with a PXE boot environment installed as per an earlier document in this series, the Hive Diskless CentOS document.

You need at least two bootable target VMs to participate in a 2-node cluster environment, and at least three to use a cluster with normal quorum rules.

Introduction

The core of a high availability cluster is always built upon some robust distributed messaging algorithms, which allow the different nodes to maintain reliable notions of group (cluster) membership, message ordering, quorum, locking, and other features.

Red Hat Enterprise Linux 4 used some custom kernel code to perform this messaging, but starting with Red Hat Enterprise Linux 5, these daemons use code from the OpenAIS project. That project essentially exposes two parts:

- An implementation of the SA Forum (Service Availability Forum) API for applications to gain access to cluster information, called the Application Interface Specification (AIS).
- A core clustering engine, called **corosync**. This engine uses the Totem group communication protocol to manage its state.

See the References section for more useful links on these core components.

Note: that the specific implementation of these daemons will be changing in Red Hat Enterprise Linux 6 and thus CentOS 6.

Most of the configuration of the cluster is controlled by the file `/etc/cluster/cluster.conf`.
This file contains information about *all* cluster nodes and resources, so it should have the same contents on every node in the cluster. Utilities are provided to update this file while the cluster is running. You can choose between command-line utilities, a GUI utility (**`system-config-cluster`**) and a system with a web interface called **Conga**.

The Conga system is composed of a manager daemon called **`luci`** which only needs to be run on the management node, and an agent called **`ricci`** that runs on every cluster node. If you see these process names on a running system, it may indicate that Conga is in use for managing the cluster.

When initially configuring the cluster however, you must manually copy the **`cluster.conf`** file to each node.

Resources

Once a cluster has determined which nodes are members of the group, they can begin to negotiate over shared resources. Resources may be filesystems, IP addresses, processes, or any other conceptual entity where an application service relies on a particular named instance of that entity.

Therefore, the cluster configuration file also contains details of all resources used in the cluster, and provides resource type and ordering information so that the cluster can start and stop resources in the correct order to satisfy the dependencies between them. RHCS uses the **`rgmanager`** service to control this behaviour.

Quorum and Fencing

An active cluster needs a means of avoiding a network partition, usually referred to as a split-brain scenario. This can occur if cluster nodes are unable to communicate with each other, for example due to a network failure, but both are still running. In this case they may each assume that the other(s) have failed, and attempt to take over a shared resource. This in turn can lead to data corruption as multiple nodes try and write to a resource such as a filesystem.

Therefore, nodes in the cluster normally try and ensure they are part of a communicating majority group of configured nodes; this guarantees that only one group will attempt to take over a shared resource. Two-node clusters are treated as a special case, and it is also possible to configure *quorum devices* for this scenario, such as twin-tailed disks where more than one machine is connected to a single shared device.

In order to guarantee data integrity, the remaining nodes in a cluster will attempt to *fence* a failed node, which refers to cutting off its access to any shared resources such as a shared filesystem.

A brutal but effective method of doing this is to remotely kill the power to the failed node, for instance using a network power switch (NPS), thus guaranteeing that it must power back up and rejoin the cluster before attempting to use any shared resources. This is generally known as **`STONITH`** - Shoot The Other Node In The Head.

The choice of *fencing agent* therefore depends on your available hardware. It is even possible to select a "manual" fencing method which requires human intervention, although this is *not* supported in production. Each supported hardware device relies on a supporting fencing agent script. In this document we look at how fencing can be achieved in an ESX or VMWare server environment.

In RHCS, the fence daemon, **`fenced`** is started as part of the cluster manager service (**`cman`**).

Filesystems and Distributed Locking

When a cluster provides filesystem access to storage, where multiple cluster nodes all have access to the same shared storage device, a more sophisticated means is needed to manage concurrent access to the storage. In RHCS, a clustered version of the logical volume manager is provided (CLVM) which runs under the service **`clvmd`** . In order to create a robust filesystem spread across multiple devices in a cluster, RHCS also provides GFS (Global File System), in the **`gfs`** service.

However, as this document is more introductory, we do not consider those services here.

Filesystem Paths and Network Addresses

Several of the pathnames used in this install process are configurable. This document assumes particular locations for these, and for ease of reading, the explanations below use these paths without qualifying them as potentially variable names.

The following paths are used as standard:

TFTP root directory	`/tftpboot`
NFS base directory	`/data/bootimage`
HTTP boot server filesystem location	`/data/bootserv`
CentOS HTTP repository root	`/data/bootserv/centos`

The network topology is assumed to be on a private network `10.1.4.0/24` in this example. The machines being created are intended to be part of a domain `test.example.com`.

We further assume that the build machine will be acting as the DNS nameserver for the local network, and this is reflected in the resolv.conf we create. The build machine itself is named `core` in this example (ie `core.test.example.com`) and is assumed to be on `10.1.4.1`.

Installation and Initial Cluster Configuration

Several packages must be installed for clustering support, most importantly **`cman`**, **`openais`** and **`rgmanager`**. In the document which follows, we assume that you have those packages installed, plus the contents of the package group **`Clustering`** .

```
yum install cman openais rgmanager
yum groupinstall Clustering
```

Now, the lowest layer of the cluster infrastructure in CentOS 5 is **`ccsd`** , the Cluster Configuration System daemon, which synchronizes the cluster configuration file across all nodes. The **`ccsd`** daemon can be controlled with the **`ccs_tool`** command line utility, and indeed once the cluster is running, any updates to the cluster configuration *must* be made using this tool (or one of the graphical tools) rather than manually.

Initially, we create a skeleton `cluster.conf` file, with just the name of the cluster plus a cman flag set to indicate that this will be a two node cluster. Then we add a fence device; for now, we use a manual fence device until we have covered the available ESX fencing methods. You cannot use a manual fence device in production. Then, we add two nodes to the cluster, assigning each an ID, along with their fence devices and node-specific fence parameters.

This is similar to the output of:

```
ccs_tool create -2 VMCluster
ccs_tool addfence -C manual fence_manual
ccs_tool addnode -C vapp1 -n 1 -f manual
ccs_tool addnode -C vapp2 -n 2 -f manual
```

where the `-C` flag prevents us trying to communicate with the CCS daemon, since it is not yet running. The resulting config file looks like this:

```
<?xml version="1.0"?>
<cluster name="VMCluster" config_version="4">
  <clusternodes>
    <clusternode name="vapp1" votes="1" nodeid="1">
      <fence>
        <method name="single"><device name="manual"/></method>
      </fence>
    </clusternode>
    <clusternode name="vapp2" votes="1" nodeid="2">
      <fence>
        <method name="single"><device name="manual"/></method>
      </fence>
    </clusternode>
  </clusternodes>
  <cman expected_votes="1" two_node="1"/>
  <fencedevices>
    <fencedevice name="manual" agent="fence_manual"/>
  </fencedevices>
  <rm>
    <failoverdomains/>
    <resources/>
  </rm>
</cluster>
```

Note that the config_version is an update counter which increments each time a change is made to the file.

Although we have not yet defined any services, we can in fact start up the cluster core at this point. The above `/etc/cluster/cluster.conf` file must be copied to every node; for a diskless install it is shared by definition.

Next, we must ensure that all necessary firewall ports are open, otherwise the startup process will stall while waiting for communication from other nodes.

Port number	Protocol	Component
5404, 5405	UDP	cman (Cluster Manager)
11111	TCP	ricci (Conga configuration agent)
14567	TCP	gnbd (Global Network Block Device)
16851	TCP	modclusterd (Configuration daemon)
21064	TCP	dlm (Distributed Lock Manager)
50006, 50008, 50009	TCP	ccsd (Cluster Configuration System)
50007	UDP	ccsd

This install also assumes that SELinux has already been disabled as per earlier documents in this series.

On CentOS 5, running a

```
service cman start
```

then starts up the `ccsd`, `aisexec`, `fenced` and `groupd` processes, plus `dlm_controld` and `gfs_controld`.

It should now be possible to use the `clustat` command line utility to check the status of the cluster nodes, although no services have been configured yet. You should see output like:

```
[root@vapp1 ~]# clustat
Cluster Status for VMCluster @ Tue May 25 06:50:40 2010
Member Status: Quorate

 Member Name                                             ID   Status
 ------ ----                                             ---- ------
 vapp2                                                      1 Online
 vapp1                                                      2 Online, Local
```

Next we can add a service, by configuring shared resources for the cluster to manage.

Shared Cluster Resources

RHCS supports a variety of shared resources. In this example we configure a shared alias IP address `10.1.4.150` and a "script" resource which represents the Apache HTTP daemon. We add the following to `/etc/cluster/cluster.conf` :

```
<cluster>
        <rm>
                <failoverdomains/>
                <resources/>
                <service autostart="1" name="VirtualHTTP">
                        <ip address="10.1.4.150" monitor_link="1"/>
                        <script name="http" file="/etc/init.d/httpd" />
                </service>
        </rm>
</cluster>
```

Note: There are not any command line tools to update `cluster.conf` with resource details; you must either edit the file manually and update using `ccs_tool update` , or use one of the GUI tools.

If the cluster is being given control of httpd, then we must ensure that the system never tries to autostart it:

```
chkconfig httpd off
```

We also assume here that Apache has been configured to listen on the shared IP address, `10.1.4.150` . This can be achieved, for example, with a `Listen 10.1.4.150:80` directive in `httpd.conf`, or simply with `Listen *` to listen to every configured address.

In addition, we assume that the resources that Apache is serving are being shared via another means, such as a SAN NFS mount. For another scenario using a cluster-controlled NFS service, see the **Red Hat Cluster Suite NFS Cookbook** reference below.

At this point we can start up the resource manager service.

Note: If using cluster LVM and GFS, you would start those services at this point, ahead of the resource manager, with:

```
service clvmd start
service gfs start
```

However, we do not use those components in this scenario.

We can now perform a

```
service rgmanager start
```

and we should then find that the `clurgmgrd` process and some supporting kernel processes have been started.

We should now see `clustat` output like:

```
[root@vapp1 ~]# clustat
Cluster Status for VMCluster @ Tue May 25 07:14:04 2010
Member Status: Quorate

Member Name                              ID Status
------ ----                              ---- -------
vapp2                                    1 Online, rgmanager
vapp1                                    2 Online, Local, rgmanager

Service Name                             Owner (Last)              State
-------- ----                            ----- ------              -----
service:VirtualHTTP                      vapp1                     started
```

If the service state is failed, the cluster resource manager outputs errors into `/var/log/messages` under the default syslog configuration, allowing you to diagnose the specific error.

Service Administration

To administer services from the command line, use the `clusvcadm` command.
The man page and default help text are quite sufficient, but to illustrate, you can enable the VirtualHTTP service (if not already started), migrate it onto a new node vapp2, and disable it, with:

```
clusvcadm -e VirtualHTTP
clusvcadm -r VirtualHTTP -m vapp2
clusvcadm -d VirtualHTTP
```

When migrating a service, the cluster manager will shutdown all resources on its current node before attempting to start them on the target node. (Obviously, HTTP serving would more likely be performed with an IP load balancer for load sharing and avoiding service interruption, but it is the simplest example of using a shared IP resource for demonstration purposes).

It is also possible to stop a service without disabling it, using `clusvcadm -s` . The service is then stopped in place, awaiting re-enabling or migration.

In the example above, you can use `ip addr list` on each machine to see the IP address migrate from node to node. Notice that it will not appear when using `ifconfig`, however.

```
[root@vapp2 ~]# ip addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
qlen 1000
    link/ether 00:0c:29:10:52:d5 brd ff:ff:ff:ff:ff:ff
    inet 10.1.4.202/24 brd 10.1.4.255 scope global eth0
    inet 10.1.4.150/24 scope global secondary eth0
    inet6 fe80::20c:29ff:fe10:52d5/64 scope link
       valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

Adding and removing cluster nodes

It is likely that at some point in the lifecycle of a cluster, we will want to provision extra nodes. The exact procedure for this depends on what automation tools you are bringing to bear, but if a running cluster has a consistent `cluster.conf` file, then you can run a command like:

```
ccs_tool addnode vapp3 -n 3 -f manual
```

and the cluster should automatically synchronize the new information across all nodes; the output of `clustat` should now show the new node.

If you are using a cluster with more than two nodes, then you can simply run

```
service cman start
service rgmanager start
```

to start up the new node. However, if you are adding a node to a two-node cluster, then you must shut down the existing cluster nodes first and remove the
`<cman expected_votes="1" two_node="1" />`
flag from `cluster.conf` .

You can remove cluster nodes using

```
ccs_tool delnode <nodename>
```

which will remove the node's record from `cluster.conf`, and resynchronize across all nodes. Note that if you run `clustat` on any given remaining cluster node, you may see a lingering node record with the removed node listed as `Estranged` until the node you ran `clustat` on is restarted and builds a fresh nodelist from the cluster.conf initial configuration.

VMWare Fence devices

We cannot use the manual fence device in production (the fencing agent merely creates a fifo on the local filesystem, and waits for a human to send a confirmation signal using the same tool, to acknowledge that they have manually powered down the targeted device).

We need a reliable way to power off an existing node. Fortunately, tools are readily available to achieve this in a VMWare environment. We provide a small wrapper Perl script called `fence_esxi.pl` in Appendix A which achieves this for an ESX virtual machine. It uses the `fence_vmware_helper` script from the `cman` package to do its job, and this in turn requires that the VI Perl toolkit is installed on the machine running the script.

This script can be used with a fence device configuration like:

```
<fencedevices>
        <fencedevice agent="fence_esxi" name="esxnode1"
server="10.1.4.20" username="root" password="b0gus" />
</fencedevices>
```

where the name `esxnode1` is a unique name for the physical esx node (this could be the DNS name) and the server IP `10.1.4.20` is the IP address of the esx server to send the fence request to.

Each node would then contain a fence device configuration like:

```
<clusternode name="vapp1" nodeid="1" votes="1">
    <fence>
        <method name="single">
            <device name="esxnode1" operation="off" vmname="vapp1"/>
        </method>
    </fence>
</clusternode>
```

In a VMWare Server environment, one can use the `vmrun` command to achieve the same goal. See the VMWare document **Using vmrun to Control Virtual Machines** for details: http://www.vmware.com/pdf/vix160_vmrun_command.pdf .

The vmrun utility is normally found in the VMWare VIX install location, and can stop a virtual machine with:

```
vmrun stop <path to vmx file>
```

References

Red Hat Enterprise Linux 5 Cluster Administration
HTML version:
`http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/html/Cluster_Administration/index.html`

PDF version:
`http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/pdf/Cluster_Administration.pdf`

Red Hat Enterprise Linux 5 Cluster Overview
PDF version:
`http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/pdf/Cluster_Suite_Overview.pdf`

OpenAIS and Corosync project home pages
`http://www.openais.org/`
`http://www.corosync.org/`

Red Hat Cluster Suite NFS Cookbook
(describes setting up a highly available NFS infrastructure)
`http://sources.redhat.com/cluster/doc/nfscookbook.pdf`

CMAN and Fencing Frequently Asked Questions pages
`http://sources.redhat.com/cluster/wiki/FAQ/CMAN`
`http://sources.redhat.com/cluster/wiki/FAQ/Fencing`

Whatever happened to cman? **document, by Christine Caulfield at Red Hat**
(describes the relationship between and evolution of the various cluster subsystems)
`http://people.redhat.com/ccaulfie/docs/Whither%20cman.pdf`

Using vmrun to Control Virtual Machines:
`http://www.vmware.com/pdf/vix160_vmrun_command.pdf`

Appendix A: Sample ESX fence script

```
#!/usr/bin/perl

use strict;
use warnings;

use IPC::Open3;
use POSIX;

# Provided as-is by Jon Topper, 2010.
# None of the RH-supplied methods worked properly.
# Some of this code taken cargo-cult from fence_scsi,
# also a perl script.

my $ME = $0;

END {
  defined fileno STDOUT or return;
  close STDOUT and return;
  warn "$ME: failed to close standard output: $!\n";
  $? ||= 1;
}

$_ = $0;
s/.*\///;
my $pname = $_;

my %args;
my %pass_through = map { $_ => 1 } qw( username password operation
vmname server datacenter ) ;

while( <> ) {
    chomp;
    my( $k, $v ) = split( /=/, $_, 2 );

    if( exists( $pass_through{ $k } ) ) {
        $args{ $k } = $v;
    }
}

my ( $in, $out, $err );
my $cmd = 'fence_vmware_helper '. join( ' ', map { '--'.$_.'='.$args{
$_ } } keys %args ).' </dev/null';
my $pid = open3( *HELPER_IN, *HELPER_OUT, *HELPER_ERR, $cmd ) or die(
"$!\n" );
waitpid( $pid, 0 );
if( $? >> 8 ) {
    die( "Unable to execute fence_vmware_helper\n" );
}
close( HELPER_IN  );
close( HELPER_OUT );
close( HELPER_ERR );
```