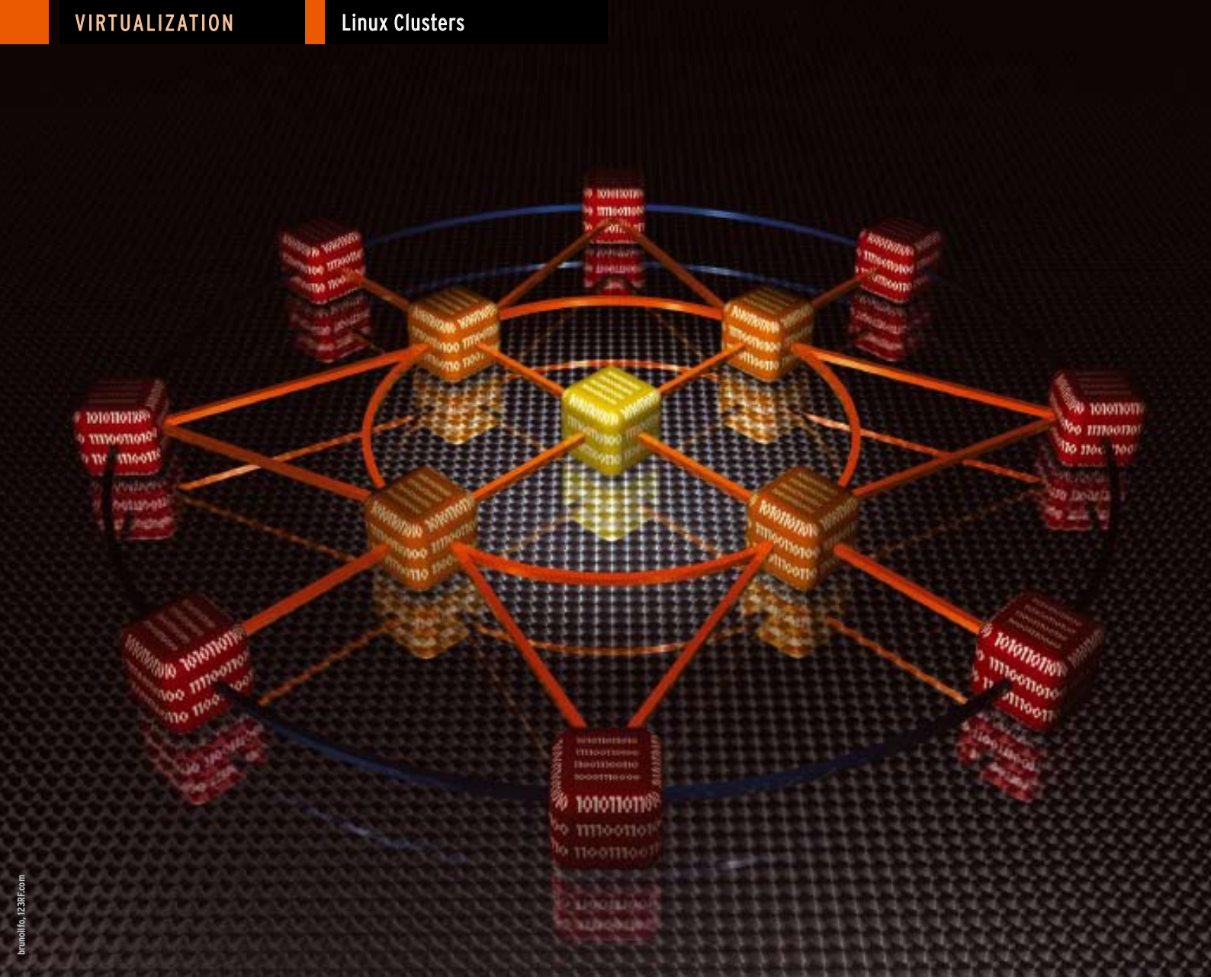

Clustering with OpenAIS and Corosync

Heartbeat

Setting up high-availability cluster resources is part of the administrator's standard bag of tricks. We look at the basic setup of a cluster using the free OpenAIS/Corosync-based cluster suite. By Thorsten Scherf

When you talk about clusters, you need to distinguish between the various types. Storage clusters support access to a single filesystem from multiple systems. This approach removes the need to synchronize the data between individual computers because they all share the same storage.
High-availability (HA) clusters bundle individual resources, such as filesystems and IP addresses, to provide a cluster service. If one resource fails, the cluster attempts to reactivate it. If the machine hosting the cluster service fails completely, the service is restarted on another computer, which ensures that the service (e.g., a web server) is available at all times.
Load distribution clusters hide multiple systems behind a single IP address and distribute the incoming requests to the back-end systems on the basis of a specific algorithm. If one of these systems fails, obviously, no requests will be forwarded to it. Clusters of this kind are particularly popular with heavily frequented web servers for which a single server would be unable to handle the high volume of requests on its own.
Finally, high-performance clusters distribute complex calculations over the cluster nodes, thus boosting the computational capacity. They are often used in research and industry, for example, for performing crash test simulations in the automotive industry. In this article, I'll look primarily at high-availability and storage clusters.

Cluster Components

A cluster always comprises multiple components. The heart is the cluster

manager – the communications system of the cluster that decides which systems belong to the cluster and which need to be removed from the cluster. Quorum rules form the basis for the decision. If a machine is not performing well, or at all, the Cluster Manager accesses another subsystem known as the Fencing System to remove the faulty node.

The Fencing daemon uses agents to communicate with fencing devices, which can be management boards, power switches, or even SAN switches. The important thing is that access to a certain cluster resource is no longer possible from a faulty node after fencing. How exactly this works depends on the cluster configuration and the requirements placed on the cluster.

A locking subsystem is required to support access to shared storage. In a cluster filesystem with GFS2 [1], communication between the individual nodes is important to ensure that only one system modifies a filesystem block at any single time; other nodes can't access the block until this action has been completed.

In volume management, too, synchronization of nodes is important. Both functions are handled by the Lock Manager. Finally, high-availability clusters also have a Resource Manager. The Resource Manager monitors and manages the configured cluster resources and services. If a node fails, the Resource Manager can restart the cluster services on another node. The service restart occurs more or less transparently from the user's point of view; in fact, the user will not typically even notice the system failure (Figure 1).

Cluster Manager

Red Hat Enterprise Linux (RHEL) and Fedora both include the Red Hat Cluster Suite (RHCS) and use CMAN as their Cluster Manager. Depending on the version you use, the implementation will look total different. The initial variant (version 1.0) included with RHEL 4 or very old versions of Fedora implemented CMAN completely in kernel space. Access by userspace applications was via the Libcman API. Networking relied on UDP broadcast/unicast. The legacy CMAN code was developed and maintained only by Red Hat.

In more recent versions (e.g., 2.0 in RHEL 5 and Fedora Core 6 or later), the legacy CMAN implementation was replaced by an open implementation based on the Application Interface Specification, AIS. The OpenAIS framework [2] is modular and provides a userspace daemon, `aisexec`, that uses various modules to access other subsystems. For example, a `totem` subsystem provides the messaging system for the cluster manager. The Red Hat CMAN code has now been modified to make CMAN just another module for the OpenAIS system. The only task handled by the CMAN module is that of providing a standardized API for existing applications – if the applications need this information from the cluster manager. Additionally, the module is responsible for communications with the quorum daemon, which is used optionally where a quorum must be calculated for a cluster. A specific algorithm is used, and the quorum daemon can be an optional part of this.

On the network front, OpenAIS relies on UDP multicast/unicast. If the cluster manager configuration doesn't define a multicast address, the address is generated dynamically. The address will then start with 239.192, with the last two octets created on the basis of the cluster ID.

Version 3.0 of the cluster suite (in RHEL 6 and Fedora 10 or later), replaces OpenAIS with Corosync [3]. Viewed superficially, not too many changes are seen between the two cluster managers, but the code is vastly different in some parts. The legacy OpenAIS modules are still available in part, but new modules have been added, and they are now called by Corosync, not by `aisexec`.

OpenAIS or Corosync

When OpenAIS or Corosync is used with the CMAN module in the Red Hat Cluster Suite, the cluster manager configuration is not handled in the typical way in the `/etc/ais/openais.conf` or `/etc/corosync/corosync.conf` configuration files. Instead, it is handled by an XML file named `/etc/cluster/cluster.conf`.

As of Cluster Suite version 3.0, an LDAP server can be used as the configuration repository. The `/etc/sysconfig/cman` file then contains a `CONFIG_LOADER` configuration parameter, which contains a value of either `xmlconfig` or `ldapconfig`. When launching CMAN via

```
/etc/init.d/cman start
```

the corresponding CMAN module loads the configuration options either from the XML file or from the LDAP server into the Corosync object database. Listing 1 shows a minimal XML configuration file.

XML Sections

The configuration file comprises several sections. Global parameters, such

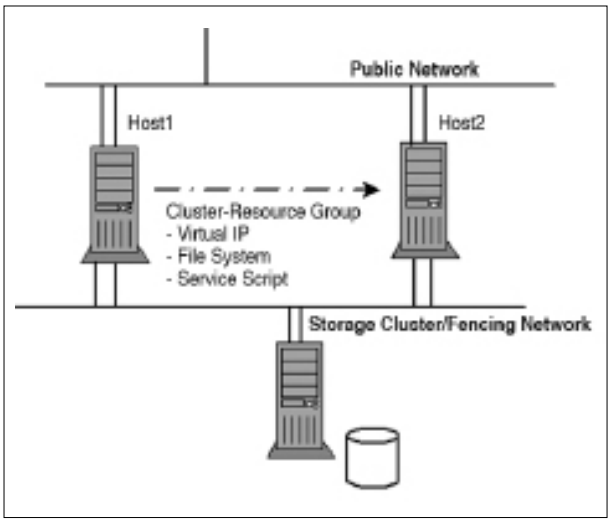

Figure 1: In a failover case, the Resource Manager launches a cluster resource group on another host.

as the cluster name are located in the `cluster` section; instructions for the CMAN plugin are in `cman`. Listing 1

Listing 1: XML Configuration File

```
01 # cat /etc/cluster/cluster.conf
02 <?xml version="1.0"?>
03 <cluster name="iscsicluster" config_version="5">
04   <cman two_node="1" expected_votes="1"/>
05   <clusternodes>
06     <clusternode name="iscsi1" votes="1" nodeid="1">
07     <fence/>
08     </clusternode>
09     <clusternode name="iscsi2" votes="1" nodeid="2">
10     <fence/>
11     </clusternode>
12   </clusternodes>
13   <fencedevices/>
14   <rm/>
15 </cluster>
```

Listing 2: Detailed Logging

```
01 <logging to_syslog="yes" to_logfile="yes" syslog_facility="daemon"
02                 syslog_priority="info" logfile_priority="info">
03       <logging_daemon name="qdiskd"
04                 logfile="/var/log/cluster/qdiskd.log"/>
05       <logging_daemon name="fenced"
06                 logfile="/var/log/cluster/fenced.log"/>
07       <logging_daemon name="dlm_controld"
08                 logfile="/var/log/cluster/dlm_controld.log"/>
09       <logging_daemon name="gfs_controld"
10                 logfile="/var/log/cluster/gfs_controld.log"/>
11       <logging_daemon name="rgmanager"
12                 logfile="/var/log/cluster/rgmanager.log"/>
13       <logging_daemon name="corosync"
14                 logfile="/var/log/cluster/corosync.log"/>
15 </logging>
```

Listing 3: At Least One Fencing Device per Node

```
01 <clusternodes>
02     <clusternode name="iscsi1" votes="1" nodeid="1">
03     <fence>
04         <method name="1">
05             <device name="nps" port="1"/>
06         </method>
07     </fence>
08     </clusternode>
09     <clusternode name="iscsi2" votes="1" nodeid="2">
10     <fence>
11         <method name="1">
12             <device name="nps" port="1"/>
13         </method>
14     </fence>
15 <clusternodes>
16 <fencedevices>
17     <fencedevice name="nps" agent="fence_nps" ipaddr="1.2.3.4"
                   passwd="redhat123"/>
18 </fencedevices>
```

contains two instructions, `two_node` and `expected_votes`, for this. The first instruction tells the cluster manager that the current cluster comprises only two nodes and that only a single vote is needed to calculate the quorum.

When clusters have more than two nodes, at least half the votes from the existing cluster nodes plus one are needed ($n/2+1$). A two-node cluster is thus an exception to the rule, as evidenced by the `expected_votes` parameter. The value here is typically identical to the number of cluster nodes, but this is not the case in a two-node cluster.

Besides the parameters mentioned so far, you also can define a multicast address, an alternative UDP port (default: 5405) for the CMAN reception socket, or the cluster ID here. If no cluster ID is defined, the cluster will define one. This can be an issue if you have multiple clusters on the same network and they accidentally generate the same ID.

CMAN generates a key based on the cluster name to encrypt the data traffic. If you prefer to use a different key, you can do so by adding a `keyfile` instruction to this section. Because CMAN is only one module of many in the Corosync Framework, you can add instructions for the other modules to the `cluster.conf` configuration file. For example, the Totem protocol gives you the option of defining a timeout for the Totem token (`<totem token="30000"/>`). The token travels back and forth between the individual cluster nodes. If the `timeout` period elapses on sending the token, the node is assumed to be down and is removed from the cluster.

The `logging` instruction is also very useful, because it lets administrators write the logs for all the subsystems involved with the cluster to a logfile, send them to syslog, or output them onscreen. Every subsystem can have its own logfile. Listing 2 provides an example.

For more Corosync-specific parameters, refer to the help page for `corosync.conf`.

Cluster Nodes

The next section, `clusternodes`, describes the individual nodes in the cluster and specifies their properties. Each cluster node is assigned a name, a vote, and an ID. Communication between cluster nodes is handled by the network, where the node names specified can be resolved. If you want to separate your data traffic from the cluster traffic, make sure you use appropriate DNS names.

If a cluster node has more than one vote, the quorum rule referred to previously no longer applies because the number of available votes and not the number of cluster nodes is the basis for quorum calculations. For example, in a cluster with three nodes, at least two computers should be online to achieve a quorum ($3/2+1=2$); otherwise, the cluster is not *quorate* and cannot provide HA service. But, if one node is given two votes rather than one, it is sufficient for this node to be online on its own for the cluster to be quorate, even if the other two machines are offline.

As another property, you can assign at least one reference to a fencing device to each node. The device itself must be defined in the `fence` section. This configuration is shown in Listing 3.

The fencing subsystem configuration is very important. If the cluster manager is unable to drop a machine that is down from the cluster (i.e., to receive a positive response to a

fencing event from the fencing daemon), the resource manager for the HA service that might be running on this node can't be launched on any other node. It is conceivable that the computer has just frozen temporarily and will wake up again after a certain amount of time and want to access its resources again while a second machine tries to do the same thing. A situation like this could lead to corrupted data.

If you now copy the `cluster.conf` file to another cluster node and then launch the cluster manager by typing `/etc/init.d/cman start`, you should see output similar to Listing 4 when you query the `cman_tool status`. Note that the current cluster version 3.0 no longer contains a cluster configuration system. Thus, the option of transferring changes to the configuration to the cluster manager via `ccs_tool update` no longer exists. Instead, you can introduce the cluster to a new version of the cluster configuration by entering `cman_tool version -r` *version_number* and transfer the configuration to the other nodes. This step completes the configuration for the cluster manager. Using the `corosync-objctl` tool

```
# corosync-objctl | grep cluster.name
cluster.name=iscsicluster
```

confirms that the Corosync database now knows the individual instructions and parameters.

LDAP Instead of XML

If you prefer to complete the configuration on an LDAP server, you can convert an existing XML config to LDIF format (Listing 5) then use `ldapadd` with the resulting LDIF file to import it to the LDAP server. First you must introduce your LDAP server to the matching LDAP schema file in the `/usr/share/doc/cman-version/` folder; otherwise, the LDAP server will not recognize the object classes and attributes and the LDIF import will fail. To ensure that every node in the cluster can access the configuration, you must add the LDAP server and the `BaseDN` to your `/etc/sysconfig/cman` configuration file:

```
# grep -i ldap /etc/sysconfig/cman
CONFIG_LOADER=ldapconfig
COROSYNC_LDAP_URL=ldap://ldap.tuxgeek.de
```

After CMAN is restarted, it should be able to populate the Corosync object database with entries from the LDAP server. To add new objects such as fencing devices or cluster resources to the LDAP database, add them to a dummy XML file first and generate the matching LDIF file from it. If you enjoy speaking LDIF, you can generate the objects directly in your LDAP tree. Managing the cluster configuration in an LDAP tree is still experimental and should not be used in production environments.

Listing 4: cman_tool status

```
01 # cman_tool status
02 Version: 6.2.0
03 Config Version: 5
04 Cluster Name: iscsicluster
05 Cluster Id: 46516
06 Cluster Member: Yes
07 Cluster Generation: 748
08 Membership state: Cluster-Member
09 Nodes: 2
10 Expected votes: 1
11 Total votes: 2
12 Node votes: 1
13 Quorum: 1
14 Active subsystems: 8
15 Flags: 2node
16 Ports Bound: 0
17 Node name: iscsi1
18 Node ID: 1
19 Multicast addresses: 239.192.181.106
20 Node addresses: 192.168.122.171
```

Resource Manager

As I mentioned previously, the resource manager in an HA cluster is

Listing 5: Converting the XML Config to LDIF

```
01 # confdb2ldif dc=tuxgeek,dc=de > cluster.ldif
02 # This file was generated by confdb2ldif,
03 # from an existing cluster configuration
04 #
05
06 dn: name=cluster,dc=tuxgeek,dc=de
07 name: iscsicluster
08 rhcsConfig-version: 5
09 objectclass: rhcsCluster
10
11 dn: cn=cman,name=cluster,dc=tuxgeek,dc=de
12 rhcsTwo-node: 1
13 rhcsExpected-votes: 1
14 rhcsNodename: iscsi1
15 rhcsCluster-id: 46516
16 cn: cman
17 objectclass: rhcsCman
18
19 dn: cn=clusternodes,name=cluster,dc=tuxgeek,
      dc=de
20 cn: clusternodes
21 objectclass: nsContainer
22
23 dn: cn=clusternode,cn=clusternodes,name=clust
   er,dc=tuxgeek,dc=de
24 cn: clusternode
25 objectclass: nsContainer
26
27 dn:
28 name=iscsi1,cn=clusternode,cn=clusternodes,na
   me=cluster,dc=tuxgeek,dc=de
29 name: iscsi1
30 rhcsVotes: 1
31 rhcsNodeid: 1
32 objectclass: rhcsClusternode
33
34 dn:
35 cn=fence,name=iscsi1,cn=clusternode,
   cn=clusternodes,name=cluster,dc=tuxgeek,dc=de
36 cn: fence
37 objectclass: nsContainer
38
39 dn: cn=clusternode,cn=clusternodes,name=clust
   er,dc=tuxgeek,dc=de
40 cn: clusternode
41 objectclass: nsContainer
42
43 dn:
44 name=iscsi2,cn=clusternode,cn=clusternodes,na
   me=cluster,dc=tuxgeek,dc=de
45 name: iscsi2
46 rhcsVotes: 1
47 rhcsNodeid: 2
48 objectclass: rhcsClusternode
49
50 dn:
51 cn=fence,name=iscsi2,cn=clusternode,cn=cluste
   rnodes,name=cluster,dc=tuxgeek,dc=de
52 cn: fence
53 objectclass: nsContainer
54
55 dn: cn=fencedevices,name=cluster,dc=tuxgeek,
      dc=de
56 cn: fencedevices
57 objectclass: nsContainer
58
59 dn: cn=rm,name=cluster,dc=tuxgeek,dc=de
60 cn: rm
61 objectclass: nsContainer
```

responsible for providing and managing the cluster services – also known as resource groups. These duties include manually and automatically starting and stopping the services and switching to other cluster nodes if the active cluster node fails.

In cluster versions 1.0 and 2.0, `rgmanager` was the only master of the Red Hat Cluster, but as of cluster version 3.0, the `pacemaker` tool is now included with RHEL 6 and Fedora 12 or newer.

The configuration in `rgmanager` also relies on the `cluster.conf` XML file or uses LDAP. The `pacemaker` tool has its own XML-based configuration file – also known as the Cluster Information Base (CIB). Manual editing of the file is not advisable; it makes far more sense to use the `crm` tool. In this section, I will be referring to the legacy `rgmanager`.

In the `cluster.conf` XML file, all configuration instructions for `rgmanager` reside in the `rm` section. The first step is to create a failover domain for the high-availability cluster services, which involves restricting the cluster nodes on which a service can run. This step is very practical if you want to restrict your heavyweight Oracle database to the more powerful machines in your cluster. The configuration for a failover domain is given in Listing 6.

Listing 6: Failover Domain

```
01 <failoverdomains>
02     <failoverdomain name="www-domain" ordered="0"
                      restricted="0" nofailback="1">
03         <failoverdomainnode name="node1"
                               priority="2"/>
04         <failoverdomainnode name="node2"
                               priority="1"/>
05     </failoverdomain>
06     <failoverdomain name="oracle-domain" ordered="0"
                      restricted="0" nofailback="1">
07         <failoverdomainnode name="node3"
                               priority="1"/>
08     </failoverdomain>
09 </failoverdomains>
```

Listing 7: Cluster Services per rgmanager

```
01 <resources>
02     <script file="/etc/init.d/httpd" name="httpd"/>
03     <fs device="/dev/vdb1" force_fsck="0"
        force_umount="1" self_fence="1" fsytpe="ext3"
        mountpoint="/var/www/html/" name="docroot"/>
04     <ip address="192.168.0.60" monitor_link="1"/>
05 </resource>
06 <service autostart="1" domain="www-domain"
           name="www">
07     <ip ref="192.168.122.20"/>
08     <fs ref="docroot">
09         <script ref="httpd"/>
10     </fs>
11 </service>
```

The `ordered`, `restricted`, and `nofailback` instructions allow you to specify whether certain nodes in a domain are given preferential treatment, whether the service is allowed to run on nodes outside the failover domain, and whether a preferred node should be used on failover when it again becomes available in a failover domain (e.g., following a fence event).

A resource group groups the individual cluster resources. The group can be defined either separately in a `resources` block, or it can follow a service definition. The first variant lets you use resources multiple times by referencing them. The resource group itself is then defined within a `service` block and points to the resources I just mentioned.

If you want to bind the service to a failover domain you set up previously, just specify it with the `domain` parameter. The default service policy is `restart`. In other words, if the service fails, `rgmanager` attempts to restart it on the same node.

If this process fails, the resource manager starts the service on another node in the specified failover domain. Listing 7 shows an example for a high-availability web server configuration.

The individual cluster resources are monitored by `rgmanager` by means of resource scripts. They are OCF- and LSB-compatible [4] [5] scripts in the `/usr/share/cluster/` directory. The start order of the individual resources is defined in the `service.sh` file; however, you can also define dependencies between individual resources simply by indenting (Listing 7). The timeouts for starting and stopping and the interval for checking the resources are defined in the resource scripts themselves. After configuring the cluster and resource manager, you can start the cluster services that you set up. You can use the `clusvcadm` tool:

```
# clusvcadm -e service:www
```

at the command line to do so.

Conclusions

In the course of time, much has happened behind the scenes of the Red Hat Cluster Suite. From an in-house development with a kernel-based cluster manager, the framework has mutated into a completely open cluster manager based on Corosync.

The old CMAN now plays a fairly insignificant role and mainly supports legacy functions. The advantage is that the configuration itself has not changed much; it still uses the `cluster.conf` XML file.

The `rgmanager` tool still is used for cluster resources; however, Pacemaker, by the popular Heartbeat project, now provides an alternative. Although complete integration still might take some time, the current Fedora 14 version, or RHEL 6, which includes Pacemaker as a technology preview, will give you an initial impression. For up-to-date information on developments in the cluster field, the Red Hat cluster pages are always a useful resource [6]. ■

Info

[1] GFS2: [http://sources.redhat.com/cluster/gfs/]

[2] OpenAIS: [http://www.openais.org/doku.php]

[3] Corosync: [http://www.corosync.org/doku.php]

[4] Open Cluster Framework: [http://opencf.org/]

[5] Linux Standard Base: [http://www.linuxfoundation.org/collaborate/workgroups/lsb]

[6] Red Hat cluster pages: [http://www.sourceware.org/cluster/wiki/]

The Author

Thorsten Scherf is a Senior Consultant for Red Hat EMEA. You can meet him as a speaker at conferences. He is also a keen marathon runner whenever time permits.