Pacemaker Administration

Release 2.1.5

the Pacemaker project contributors

Dec 12, 2022

CONTENTS

Managing Pacemaker Clusters

CHAPTER

ONE

ABSTRACT

This document has instructions and tips for system administrators who manage high-availability clusters using Pacemaker.

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Introduction

2.1.1 The Scope of this Document

The purpose of this document is to help system administrators learn how to manage a Pacemaker cluster.

System administrators may be interested in other parts of the Pacemaker documentation set such as *Clusters from Scratch*, a step-by-step guide to setting up an example cluster, and *Pacemaker Explained*, an exhaustive reference for cluster configuration.

Multiple higher-level tools (both command-line and GUI) are available to simplify cluster management. However, this document focuses on the lower-level command-line tools that come with Pacemaker itself. The concepts are applicable to the higher-level tools, though the syntax would differ.

2.1.2 What Is Pacemaker?

Pacemaker is a high-availability *cluster resource manager* – software that runs on a set of hosts (a *cluster* of *nodes*) in order to preserve integrity and minimize downtime of desired services (*resources*).[1] It is maintained by the ClusterLabs community.

Pacemaker's key features include:

- Detection of and recovery from node- and service-level failures
- Ability to ensure data integrity by fencing faulty nodes
- Support for one or more nodes per cluster
- Support for multiple resource interface standards (anything that can be scripted can be clustered)
- Support (but no requirement) for shared storage
- Support for practically any redundancy configuration (active/passive, N+1, etc.)
- Automatically replicated configuration that can be updated from any node
- Ability to specify cluster-wide relationships between services, such as ordering, colocation, and anti-colocation
- Support for advanced service types, such as *clones* (services that need to be active on multiple nodes), *promotable clones* (clones that can run in one of two roles), and containerized services
- Unified, scriptable cluster management tools

[1] *Cluster* is sometimes used in other contexts to refer to hosts grouped together for other purposes, such as high-performance computing (HPC), but Pacemaker is not intended for those purposes.

Note: Fencing

Fencing, also known as *STONITH* (an acronym for Shoot The Other Node In The Head), is the ability to ensure that it is not possible for a node to be running a service. This is accomplished via *fence devices* such as intelligent power switches that cut power to the target, or intelligent network switches that cut the target's access to the local network.

Pacemaker represents fence devices as a special class of resource.

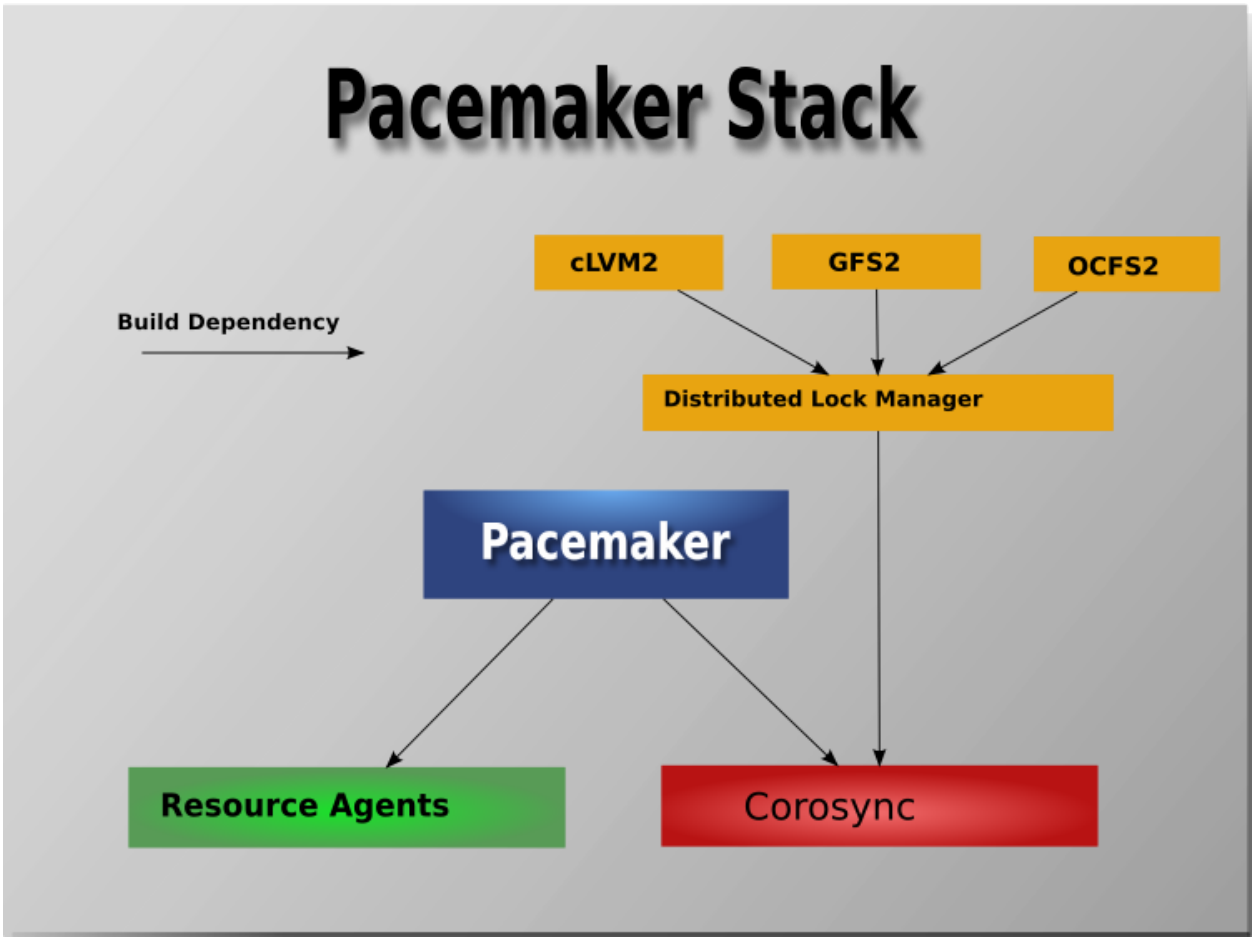
A cluster cannot safely recover from certain failure conditions, such as an unresponsive node, without fencing.

Cluster Architecture

At a high level, a cluster can be viewed as having these parts (which together are often referred to as the *cluster stack*):

- **Resources:** These are the reason for the cluster's being – the services that need to be kept highly available.
- **Resource agents:** These are scripts or operating system components that start, stop, and monitor resources, given a set of resource parameters. These provide a uniform interface between Pacemaker and the managed services.
- **Fence agents:** These are scripts that execute node fencing actions, given a target and fence device parameters.
- **Cluster membership layer:** This component provides reliable messaging, membership, and quorum information about the cluster. Currently, Pacemaker supports Corosync as this layer.
- **Cluster resource manager:** Pacemaker provides the brain that processes and reacts to events that occur in the cluster. These events may include nodes joining or leaving the cluster; resource events caused by failures, maintenance, or scheduled activities; and other administrative actions. To achieve the desired availability, Pacemaker may start and stop resources and fence nodes.
- **Cluster tools:** These provide an interface for users to interact with the cluster. Various command-line and graphical (GUI) interfaces are available.

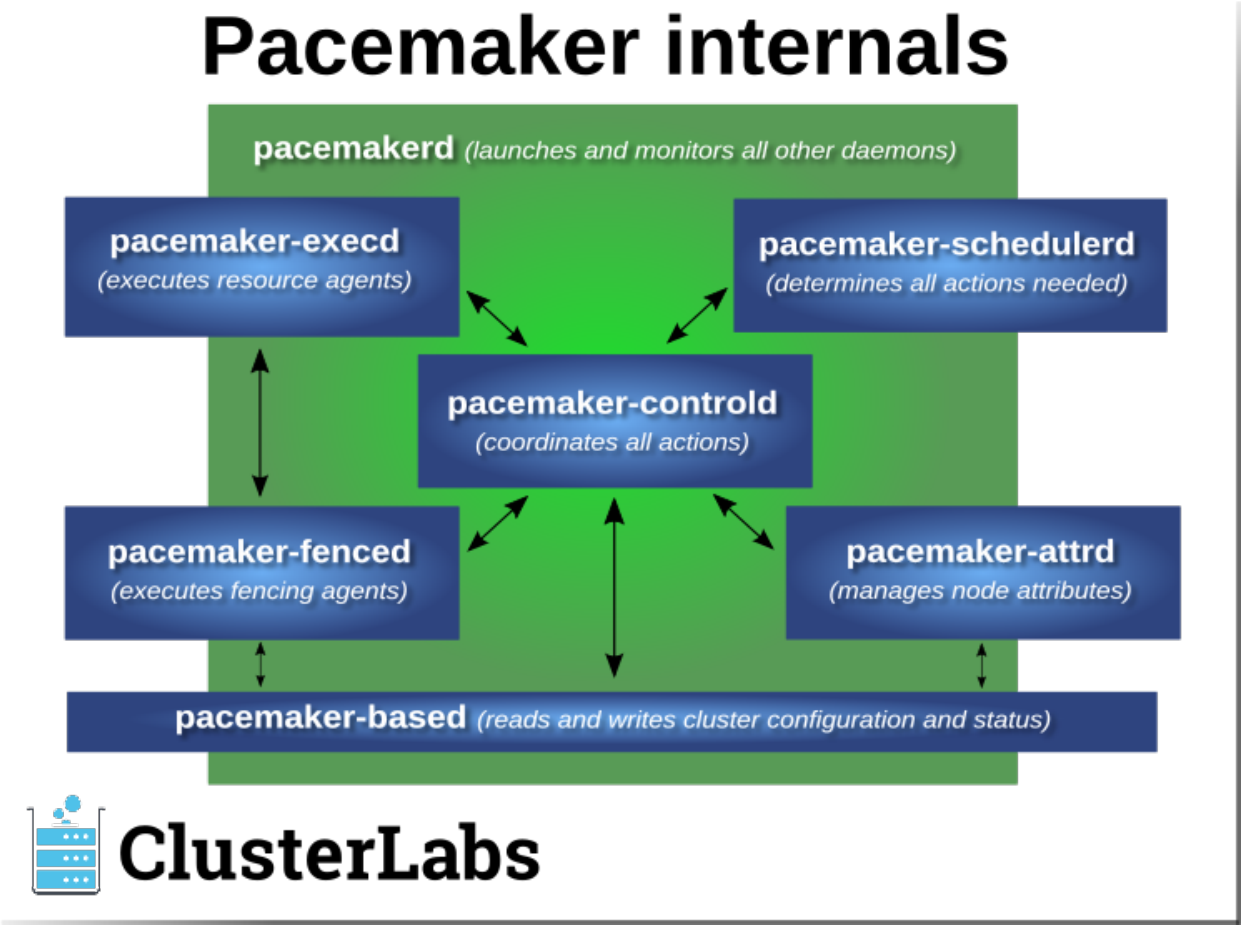
Most managed services are not, themselves, cluster-aware. However, many popular open-source cluster filesystems make use of a common *Distributed Lock Manager* (DLM), which makes direct use of Corosync for its messaging and membership capabilities and Pacemaker for the ability to fence nodes.



Pacemaker Architecture

Pacemaker itself is composed of multiple daemons that work together:

- `pacemakerd`
- `pacemaker-attrd`
- `pacemaker-based`
- `pacemaker-controld`
- `pacemaker-execd`
- `pacemaker-fenced`
- `pacemaker-schedulerd`



Pacemaker's main process (`pacemakerd`) spawns all the other daemons, and respawns them if they unexpectedly exit.

The *Cluster Information Base* (CIB) is an XML representation of the cluster's configuration and the state of all nodes and resources. The *CIB manager* (`pacemaker-based`) keeps the CIB synchronized across the cluster, and handles requests to modify it.

The *attribute manager* (`pacemaker-attrd`) maintains a database of attributes for all nodes, keeps it synchronized across the cluster, and handles requests to modify them. These attributes are usually recorded in the CIB.

Given a snapshot of the CIB as input, the *scheduler* (`pacemaker-schedulerd`) determines what actions are necessary to achieve the desired state of the cluster.

The *local executor* (`pacemaker-execd`) handles requests to execute resource agents on the local cluster node, and returns the result.

The *fencer* (`pacemaker-fenced`) handles requests to fence nodes. Given a target node, the fencer decides which cluster node(s) should execute which fencing device(s), and calls the necessary fencing agents (either directly, or via requests to the fencer peers on other nodes), and returns the result.

The *controller* (`pacemaker-controld`) is Pacemaker's coordinator, maintaining a consistent view of the cluster membership and orchestrating all the other components.

Pacemaker centralizes cluster decision-making by electing one of the controller instances as the *Designated Controller* (*DC*). Should the elected DC process (or the node it is on) fail, a new one is quickly established. The DC responds to cluster events by taking a current snapshot of the CIB, feeding it to the scheduler, then

asking the executors (either directly on the local node, or via requests to controller peers on other nodes) and the fencer to execute any necessary actions.

Note: Old daemon names

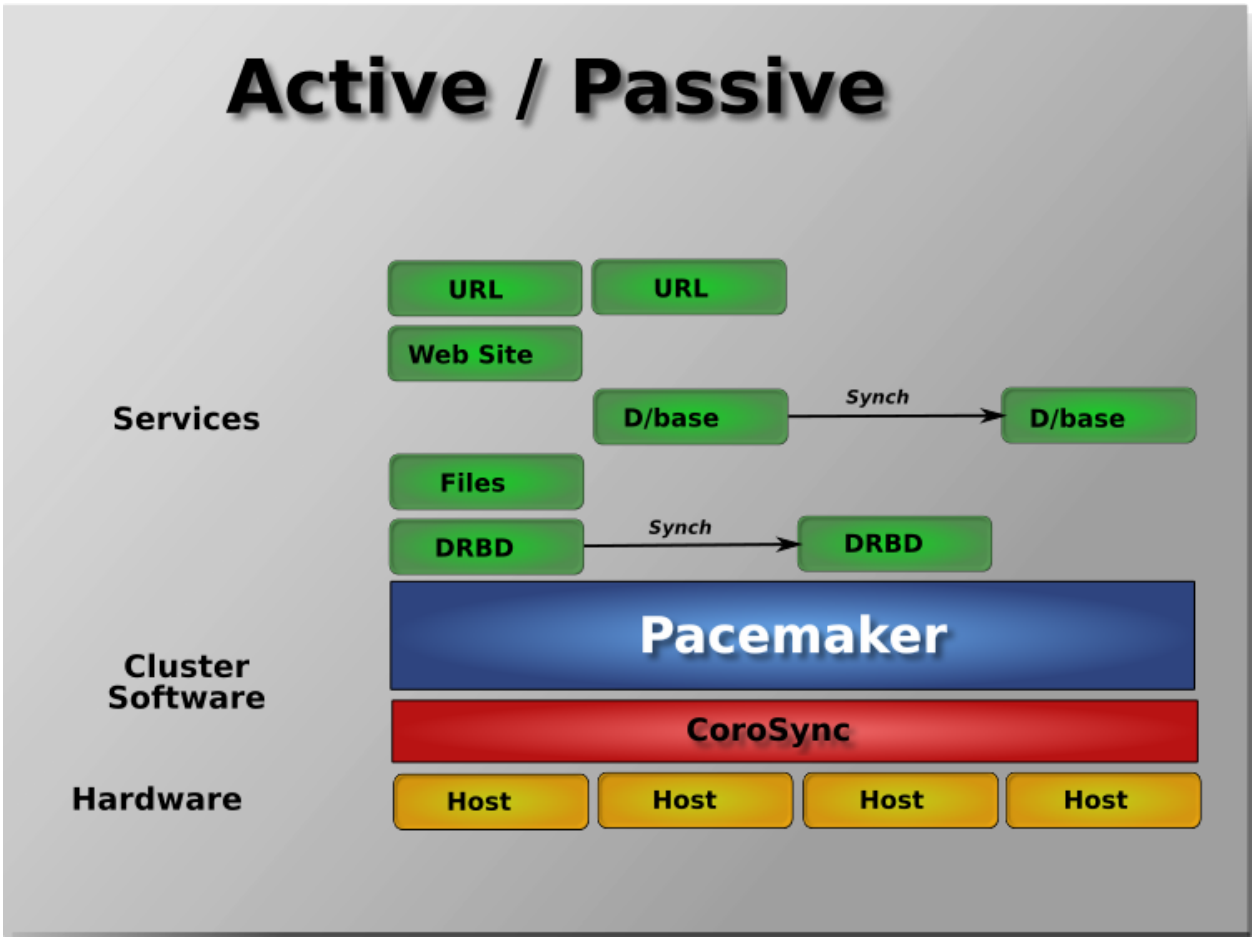
The Pacemaker daemons were renamed in version 2.0. You may still find references to the old names, especially in documentation targeted to version 1.1.

Old name	New name
`attrd`	`pacemaker-attrd`
`cib`	`pacemaker-based`
`crmd`	`pacemaker-controld`
`lrmd`	`pacemaker-execd`
`stonithd`	`pacemaker-fenced`
`pacemaker_remoted`	`pacemaker-remoted`

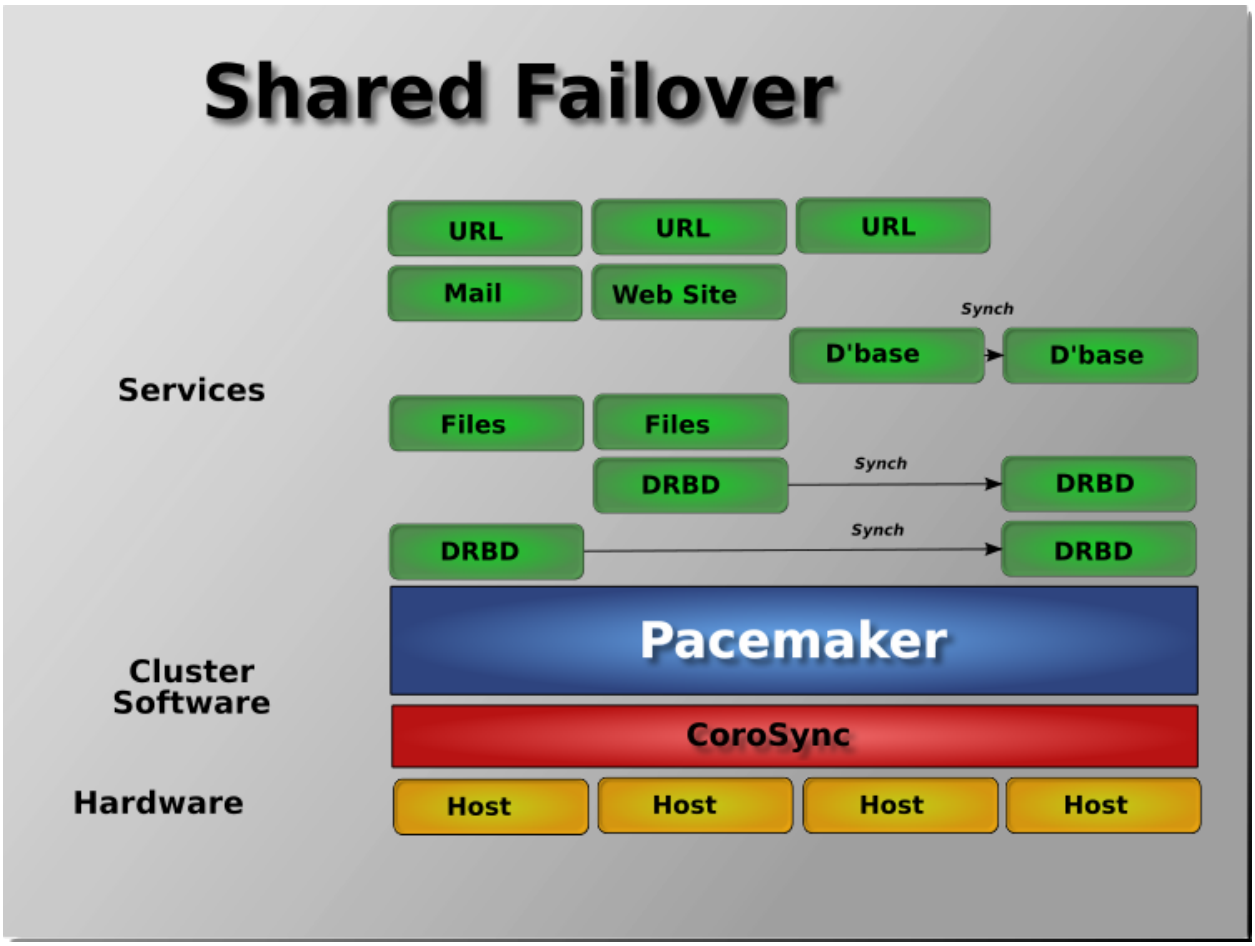
Node Redundancy Designs

Pacemaker supports practically any node redundancy configuration including *Active/Active*, *Active/Passive*, *N+1*, *N+M*, *N-to-1*, and *N-to-N*.

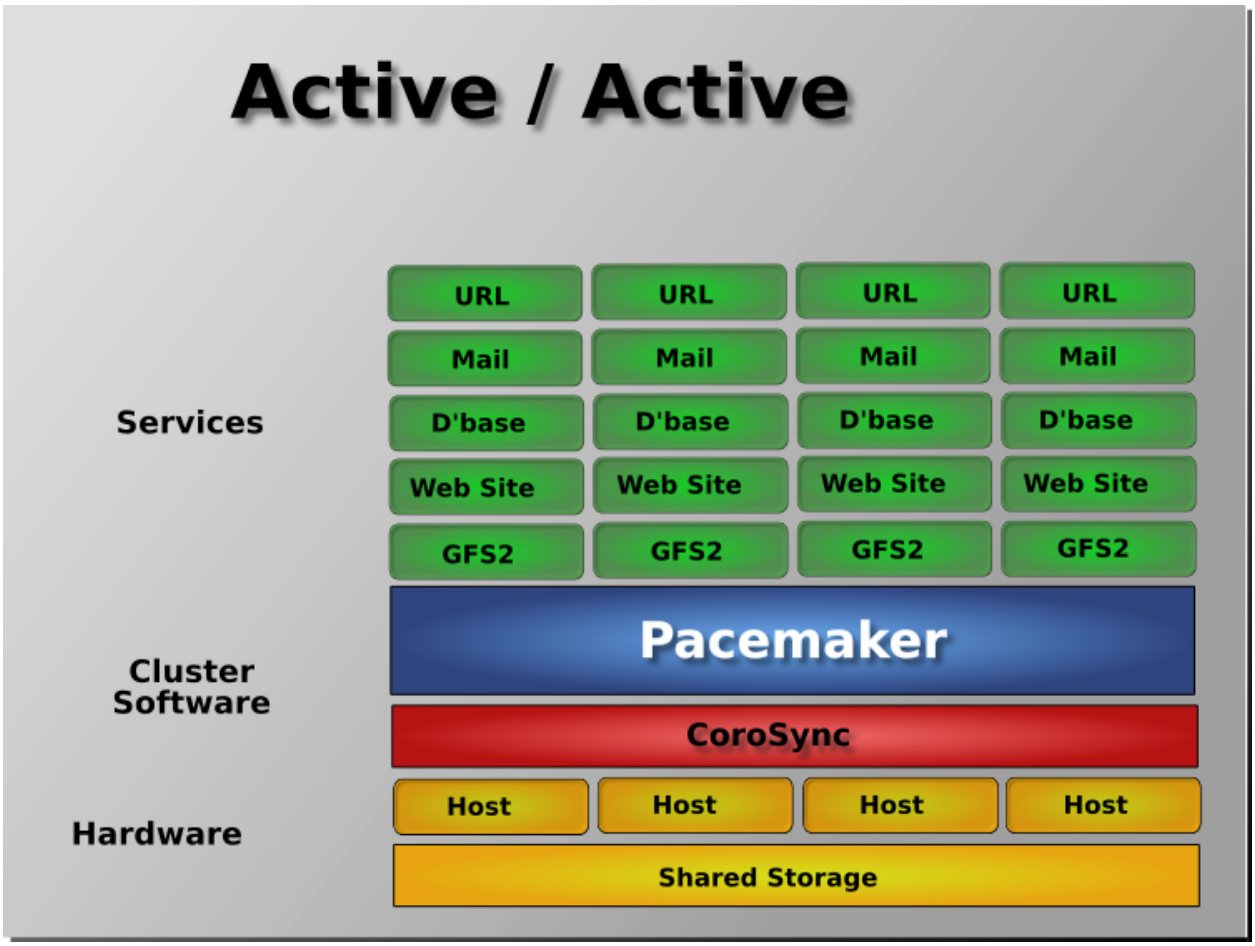
Active/passive clusters with two (or more) nodes using Pacemaker and DRBD are a cost-effective high-availability solution for many situations. One of the nodes provides the desired services, and if it fails, the other node takes over.



Pacemaker also supports multiple nodes in a shared-failover design, reducing hardware costs by allowing several active/passive clusters to be combined and share a common backup node.



When shared storage is available, every node can potentially be used for failover. Pacemaker can even run multiple copies of services to spread out the workload. This is sometimes called N-to-N redundancy.



2.2 Installing Cluster Software

Most major Linux distributions have pacemaker packages in their standard package repositories, or the software can be built from source code. See the Install wiki page for details.

2.3 The Cluster Layer

Pacemaker utilizes an underlying cluster layer for two purposes:

- obtaining quorum
- messaging between nodes

Currently, only Corosync 2 and later is supported for this layer.

This document assumes you have configured the cluster nodes in Corosync already. High-level cluster management tools are available that can configure Corosync for you. If you want the lower-level details, see the Corosync documentation.

2.4 Configuring Pacemaker

Pacemaker's configuration, the CIB, is stored in XML format. Cluster administrators have multiple options for modifying the configuration either via the XML, or at a more abstract (and easier for humans to understand) level.

Pacemaker reacts to configuration changes as soon as they are saved. Pacemaker's command-line tools and most higher-level tools provide the ability to batch changes together and commit them at once, rather than make a series of small changes, which could cause avoid unnecessary actions as Pacemaker responds to each change individually.

Pacemaker tracks revisions to the configuration and will reject any update older than the current revision. Thus, it is a good idea to serialize all changes to the configuration. Avoid attempting simultaneous changes, whether on the same node or different nodes, and whether manually or using some automated configuration tool.

Note: It is not necessary to update the configuration on all cluster nodes. Pacemaker immediately synchronizes changes to all active members of the cluster. To reduce bandwidth, the cluster only broadcasts the incremental updates that result from your changes and uses checksums to ensure that each copy is consistent.

2.4.1 Configuration Using Higher-level Tools

Most users will benefit from using higher-level tools provided by projects separate from Pacemaker. Some of the most commonly used include the crm shell, hawk, and pcs.[1]

See those projects' documentation for details on how to configure Pacemaker using them.

2.4.2 Configuration Using Pacemaker's Command-Line Tools

Pacemaker provides lower-level, command-line tools to manage the cluster. Most configuration tasks can be performed with these tools, without needing any XML knowledge.

To enable STONITH for example, one could run:

```
# crm_attribute --name stonith-enabled --update 1
```

Or, to check whether **node1** is allowed to run resources, there is:

```
# crm_standby --query --node node1
```

Or, to change the failure threshold of **my-test-rsc**, one can use:

```
# crm_resource -r my-test-rsc --set-parameter migration-threshold --parameter-value 3 --meta
```

Examples of using these tools for specific cases will be given throughout this document where appropriate. See the man pages for further details.

See *Edit the CIB XML with cibadmin* for how to edit the CIB using XML.

See *Batch Configuration Changes with crm_shadow* for a way to make a series of changes, then commit them all at once to the live cluster.

[1] For a list, see "Configuration Tools" at https://clusterlabs.org/components.html

Working with CIB Properties

Although these fields can be written to by the user, in most cases the cluster will overwrite any values specified by the user with the "correct" ones.

To change the ones that can be specified by the user, for example `admin_epoch`, one should use:

```
# cibadmin --modify --xml-text '<cib admin_epoch="42"/>'
```

A complete set of CIB properties will look something like this:

XML attributes set for a cib element

```
<cib crm_feature_set="3.0.7" validate-with="pacemaker-1.2"
   admin_epoch="42" epoch="116" num_updates="1"
   cib-last-written="Mon Jan 12 15:46:39 2015" update-origin="rhel7-1"
   update-client="crm_attribute" have-quorum="1" dc-uuid="1">
```

Querying and Setting Cluster Options

Cluster options can be queried and modified using the `crm_attribute` tool. To get the current value of `cluster-delay`, you can run:

```
# crm_attribute --query --name cluster-delay
```

which is more simply written as

```
# crm_attribute -G -n cluster-delay
```

If a value is found, you'll see a result like this:

```
# crm_attribute -G -n cluster-delay
scope=crm_config name=cluster-delay value=60s
```

If no value is found, the tool will display an error:

```
# crm_attribute -G -n clusta-deway
scope=crm_config name=clusta-deway value=(null)
Error performing operation: No such device or address
```

To use a different value (for example, 30 seconds), simply run:

```
# crm_attribute --name cluster-delay --update 30s
```

To go back to the cluster's default value, you can delete the value, for example:

```
# crm_attribute --name cluster-delay --delete
Deleted crm_config option: id=cib-bootstrap-options-cluster-delay name=cluster-delay
```

When Options are Listed More Than Once

If you ever see something like the following, it means that the option you're modifying is present more than once.

Deleting an option that is listed twice

```
# crm_attribute --name batch-limit --delete

Please choose from one of the matches below and supply the 'id' with --id
Multiple attributes match name=batch-limit in crm_config:
Value: 50          (set=cib-bootstrap-options, id=cib-bootstrap-options-batch-limit)
Value: 100         (set=custom, id=custom-batch-limit)
```

In such cases, follow the on-screen instructions to perform the requested action. To determine which value is currently being used by the cluster, refer to the "Rules" chapter of *Pacemaker Explained*.

2.4.3 Connecting from a Remote Machine

Provided Pacemaker is installed on a machine, it is possible to connect to the cluster even if the machine itself is not in the same cluster. To do this, one simply sets up a number of environment variables and runs the same commands as when working on a cluster node.

Table 1: **Environment Variables Used to Connect to Remote Instances of the CIB**

Environment Variable	Default	Description
CIB_user	$USER	The user to connect as. Needs to be part of the `haclient` group on the target host.
CIB_passwd		The user's password. Read from the command line if unset.
CIB_server	localhost	The host to contact
CIB_port		The port on which to contact the server; required.
CIB_encrypted	TRUE	Whether to encrypt network traffic

So, if **c001n01** is an active cluster node and is listening on port 1234 for connections, and **someuser** is a member of the **haclient** group, then the following would prompt for **someuser**'s password and return the cluster's current configuration:

```
# export CIB_port=1234; export CIB_server=c001n01; export CIB_user=someuser;
# cibadmin -Q
```

For security reasons, the cluster does not listen for remote connections by default. If you wish to allow remote access, you need to set the `remote-tls-port` (encrypted) or `remote-clear-port` (unencrypted) CIB properties (i.e., those kept in the `cib` tag, like `num_updates` and `epoch`).

Table 2: **Extra top-level CIB properties for remote access**

CIB Property	Default	Description
remote-tls-port		Listen for encrypted remote connections on this port.
remote-clear-port		Listen for plaintext remote connections on this port.

Important: The Pacemaker version on the administration host must be the same or greater than the version(s) on the cluster nodes. Otherwise, it may not have the schema files necessary to validate the CIB.

2.5 Using Pacemaker Command-Line Tools

2.5.1 Controlling Command Line Output

Some of the pacemaker command line utilities have been converted to a new output system. Among these tools are `crm_mon` and `stonith_admin`. This is an ongoing project, and more tools will be converted over time. This system lets you control the formatting of output with `--output-as=` and the destination of output with `--output-to=`.

The available formats vary by tool, but at least plain text and XML are supported by all tools that use the new system. The default format is plain text. The default destination is stdout but can be redirected to any file. Some formats support command line options for changing the style of the output. For instance:

```
# crm_mon --help-output
Usage:
  crm_mon [OPTION?]

Provides a summary of cluster's current state.

Outputs varying levels of detail in a number of different formats.

Output Options:
  --output-as=FORMAT                Specify output format as one of: console (default), html, text,
↪ xml
  --output-to=DEST                  Specify file name for output (or "-" for stdout)
  --html-cgi                        Add text needed to use output in a CGI program
  --html-stylesheet=URI             Link to an external CSS stylesheet
  --html-title=TITLE                Page title
  --text-fancy                      Use more highly formatted output
```

2.5.2 Monitor a Cluster with crm_mon

The `crm_mon` utility displays the current state of an active cluster. It can show the cluster status organized by node or by resource, and can be used in either single-shot or dynamically updating mode. It can also display operations performed and information about failures.

Using this tool, you can examine the state of the cluster for irregularities, and see how it responds when you cause or simulate failures.

See the manual page or the output of `crm_mon --help` for a full description of its many options.

Sample output from crm_mon -1

```
Cluster Summary:
  * Stack: corosync
  * Current DC: node2 (version 2.0.0-1) - partition with quorum
  * Last updated: Mon Jan 29 12:18:42 2018
  * Last change:  Mon Jan 29 12:18:40 2018 by root via crm_attribute      on node3
  * 5 nodes configured
  * 2 resources configured

Node List:
  * Online: [ node1 node2 node3 node4 node5 ]

* Active resources:
  * Fencing (stonith:fence_xvm):    Started node1
  * IP    (ocf:heartbeat:IPaddr2):        Started node2
```

Sample output from crm_mon -n -1

```
Cluster Summary:
  * Stack: corosync
  * Current DC: node2 (version 2.0.0-1) - partition with quorum
  * Last updated: Mon Jan 29 12:21:48 2018
  * Last change:  Mon Jan 29 12:18:40 2018 by root via crm_attribute      on node3
  * 5 nodes configured
  * 2 resources configured

* Node List:
  * Node node1: online
    * Fencing (stonith:fence_xvm):    Started
  * Node node2: online
    * IP  (ocf:heartbeat:IPaddr2):        Started
  * Node node3: online
  * Node node4: online
  * Node node5: online
```

As mentioned in an earlier chapter, the DC is the node is where decisions are made. The cluster elects a node to be DC as needed. The only significance of the choice of DC to an administrator is the fact that its logs will have the most information about why decisions were made.

Styling crm_mon HTML output

Various parts of `crm_mon`'s HTML output have a CSS class associated with them. Not everything does, but some of the most interesting portions do. In the following example, the status of each node has an `online` class and the details of each resource have an `rsc-ok` class.

```
<h2>Node List</h2>
<ul>
<li>
<span>Node: cluster01</span><span class="online"> online</span>
</li>
<li><ul><li><span class="rsc-ok">ping   (ocf::pacemaker:ping):   Started</span></li></ul></li>
<li>
<span>Node: cluster02</span><span class="online"> online</span>
```

(continues on next page)

(continued from previous page)

```
</li>
<li><ul><li><span class="rsc-ok">ping   (ocf::pacemaker:ping):   Started</span></li></ul></li>
</ul>
```

By default, a stylesheet for styling these classes is included in the head of the HTML output. The relevant portions of this stylesheet that would be used in the above example is:

```
<style>
.online { color: green }
.rsc-ok { color: green }
</style>
```

If you want to override some or all of the styling, simply create your own stylesheet, place it on a web server, and pass `--html-stylesheet=<URL>` to `crm_mon`. The link is added after the default stylesheet, so your changes take precedence. You don't need to duplicate the entire default. Only include what you want to change.

2.5.3 Edit the CIB XML with cibadmin

The most flexible tool for modifying the configuration is Pacemaker's `cibadmin` command. With `cibadmin`, you can query, add, remove, update or replace any part of the configuration. All changes take effect immediately, so there is no need to perform a reload-like operation.

The simplest way of using `cibadmin` is to use it to save the current configuration to a temporary file, edit that file with your favorite text or XML editor, and then upload the revised configuration.

Safely using an editor to modify the cluster configuration

```
# cibadmin --query > tmp.xml
# vi tmp.xml
# cibadmin --replace --xml-file tmp.xml
```

Some of the better XML editors can make use of a RELAX NG schema to help make sure any changes you make are valid. The schema describing the configuration can be found in `pacemaker.rng`, which may be deployed in a location such as `/usr/share/pacemaker` depending on your operating system distribution and how you installed the software.

If you want to modify just one section of the configuration, you can query and replace just that section to avoid modifying any others.

Safely using an editor to modify only the resources section

```
# cibadmin --query --scope resources > tmp.xml
# vi tmp.xml
# cibadmin --replace --scope resources --xml-file tmp.xml
```

To quickly delete a part of the configuration, identify the object you wish to delete by XML tag and id. For example, you might search the CIB for all STONITH-related configuration:

Searching for STONITH-related configuration items

```
# cibadmin --query | grep stonith
 <nvpair id="cib-bootstrap-options-stonith-action" name="stonith-action" value="reboot"/>
 <nvpair id="cib-bootstrap-options-stonith-enabled" name="stonith-enabled" value="1"/>
 <primitive id="child_DoFencing" class="stonith" type="external/vmware">
 <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:1" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:2" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
 <lrm_resource id="child_DoFencing:3" type="external/vmware" class="stonith">
```

If you wanted to delete the `primitive` tag with id `child_DoFencing`, you would run:

```
# cibadmin --delete --xml-text '<primitive id="child_DoFencing"/>'
```

See the cibadmin man page for more options.

Warning: Never edit the live `cib.xml` file directly. Pacemaker will detect such changes and refuse to use the configuration.

2.5.4 Batch Configuration Changes with crm_shadow

Often, it is desirable to preview the effects of a series of configuration changes before updating the live configuration all at once. For this purpose, `crm_shadow` creates a "shadow" copy of the configuration and arranges for all the command-line tools to use it.

To begin, simply invoke `crm_shadow --create` with a name of your choice, and follow the simple on-screen instructions. Shadow copies are identified with a name to make it possible to have more than one.

Warning: Read this section and the on-screen instructions carefully; failure to do so could result in destroying the cluster's active configuration!

Creating and displaying the active sandbox

```
# crm_shadow --create test
Setting up shadow instance
Type Ctrl-D to exit the crm_shadow shell
shadow[test]:
shadow[test] # crm_shadow --which
test
```

From this point on, all cluster commands will automatically use the shadow copy instead of talking to the cluster's active configuration. Once you have finished experimenting, you can either make the changes active via the `--commit` option, or discard them using the `--delete` option. Again, be sure to follow the on-screen instructions carefully!

For a full list of `crm_shadow` options and commands, invoke it with the `--help` option.

Use sandbox to make multiple changes all at once, discard them, and verify real configuration is untouched

```
shadow[test] # crm_failcount -r rsc_c001n01 -G
scope=status  name=fail-count-rsc_c001n01 value=0
shadow[test] # crm_standby --node c001n02 -v on
shadow[test] # crm_standby --node c001n02 -G
scope=nodes  name=standby value=on

shadow[test] # cibadmin --erase --force
shadow[test] # cibadmin --query
<cib crm_feature_set="3.0.14" validate-with="pacemaker-3.0" epoch="112" num_updates="2" admin_
↪epoch="0" cib-last-written="Mon Jan  8 23:26:47 2018" update-origin="rhel7-1" update-client=
↪"crm_node" update-user="root" have-quorum="1" dc-uuid="1">
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
shadow[test] # crm_shadow --delete test --force
Now type Ctrl-D to exit the crm_shadow shell
shadow[test] # exit
# crm_shadow --which
No active shadow configuration defined
# cibadmin -Q
<cib crm_feature_set="3.0.14" validate-with="pacemaker-3.0" epoch="110" num_updates="2" admin_
↪epoch="0" cib-last-written="Mon Jan  8 23:26:47 2018" update-origin="rhel7-1" update-client=
↪"crm_node" update-user="root" have-quorum="1">
   <configuration>
      <crm_config>
         <cluster_property_set id="cib-bootstrap-options">
            <nvpair id="cib-bootstrap-1" name="stonith-enabled" value="1"/>
            <nvpair id="cib-bootstrap-2" name="pe-input-series-max" value="30000"/>
```

See the next section, *Simulate Cluster Activity with crm_simulate*, for how to test your changes before committing them to the live cluster.

2.5.5 Simulate Cluster Activity with crm_simulate

The command-line tool *crm_simulate* shows the results of the same logic the cluster itself uses to respond to a particular cluster configuration and status.

As always, the man page is the primary documentation, and should be consulted for further details. This section aims for a better conceptual explanation and practical examples.

Replaying cluster decision-making logic

At any given time, one node in a Pacemaker cluster will be elected DC, and that node will run Pacemaker's scheduler to make decisions.

Each time decisions need to be made (a "transition"), the DC will have log messages like "Calculated

transition … saving inputs in …" with a file name. You can grab the named file and replay the cluster logic to see why particular decisions were made. The file contains the live cluster configuration at that moment, so you can also look at it directly to see the value of node attributes, etc., at that time.

The simplest usage is (replacing $FILENAME with the actual file name):

Simulate cluster response to a given CIB

```
# crm_simulate --simulate --xml-file $FILENAME
```

That will show the cluster state when the process started, the actions that need to be taken ("Transition Summary"), and the resulting cluster state if the actions succeed. Most actions will have a brief description of why they were required.

The transition inputs may be compressed. `crm_simulate` can handle these compressed files directly, though if you want to edit the file, you'll need to uncompress it first.

You can do the same simulation for the live cluster configuration at the current moment. This is useful mainly when using `crm_shadow` to create a sandbox version of the CIB; the `--live-check` option will use the shadow CIB if one is in effect.

Simulate cluster response to current live CIB or shadow CIB

```
# crm_simulate --simulate --live-check
```

Why decisions were made

To get further insight into the "why", it gets user-unfriendly very quickly. If you add the `--show-scores` option, you will also see all the scores that went into the decision-making. The node with the highest cumulative score for a resource will run it. You can look for `-INFINITY` scores in particular to see where complete bans came into effect.

You can also add `-VVVV` to get more detailed messages about what's happening under the hood. You can add up to two more V's even, but that's usually useful only if you're a masochist or tracing through the source code.

Visualizing the action sequence

Another handy feature is the ability to generate a visual graph of the actions needed, using the `--save-dotfile` option. This relies on the separate Graphviz[1] project.

Generate a visual graph of cluster actions from a saved CIB

```
# crm_simulate --simulate --xml-file $FILENAME --save-dotfile $FILENAME.dot
# dot $FILENAME.dot -Tsvg > $FILENAME.svg
```

`$FILENAME.dot` will contain a GraphViz representation of the cluster's response to your changes, including all actions with their ordering dependencies.

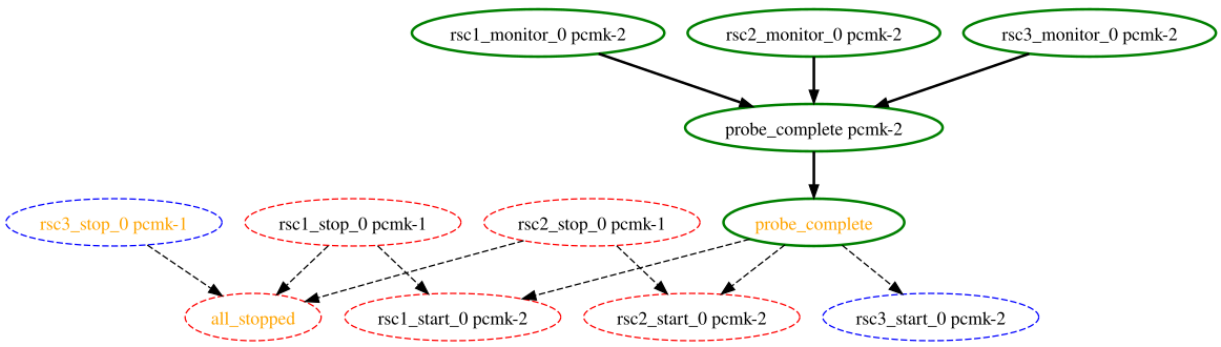
[1] Graph visualization software. See http://www.graphviz.org/ for details.

`$FILENAME.svg` will be the same information in a standard graphical format that you can view in your browser or other app of choice. You could, of course, use other `dot` options to generate other formats.

How to interpret the graphical output:

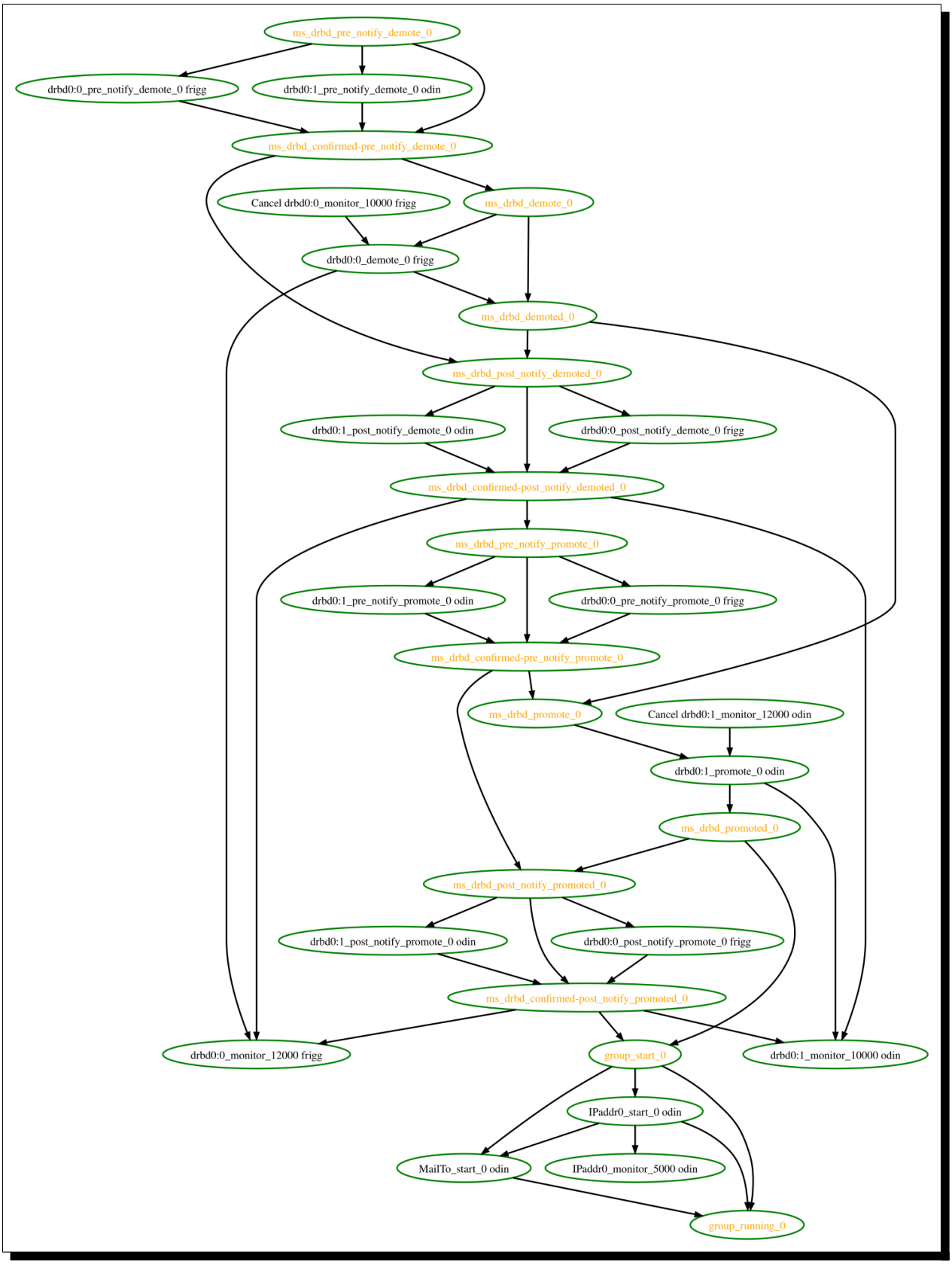
- Bubbles indicate actions, and arrows indicate ordering dependencies
- Resource actions have text of the form `<RESOURCE>_<ACTION>_<INTERVAL_IN_MS> <NODE>` indicating that the specified action will be executed for the specified resource on the specified node, once if interval is 0 or at specified recurring interval otherwise
- Actions with black text will be sent to the executor (that is, the appropriate agent will be invoked)
- Actions with orange text are "pseudo" actions that the cluster uses internally for ordering but require no real activity
- Actions with a solid green border are part of the transition (that is, the cluster will attempt to execute them in the given order – though a transition can be interrupted by action failure or new events)
- Dashed arrows indicate dependencies that are not present in the transition graph
- Actions with a dashed border will not be executed. If the dashed border is blue, the cluster does not feel the action needs to be executed. If the dashed border is red, the cluster would like to execute the action but cannot. Any actions depending on an action with a dashed border will not be able to execute.
- Loops should not happen, and should be reported as a bug if found.

Small Cluster Transition



In the above example, it appears that a new node, `pcmk-2`, has come online and that the cluster is checking to make sure `rsc1`, `rsc2` and `rsc3` are not already running there (indicated by the `rscN_monitor_0` entries). Once it did that, and assuming the resources were not active there, it would have liked to stop `rsc1` and `rsc2` on `pcmk-1` and move them to `pcmk-2`. However, there appears to be some problem and the cluster cannot or is not permitted to perform the stop actions which implies it also cannot perform the start actions. For some reason, the cluster does not want to start `rsc3` anywhere.

Complex Cluster Transition


ms_drbd_pre_notify_demote_0
drbd0:0_pre_notify_demote_0 frigg
drbd0:1_pre_notify_demote_0 odin
ms_drbd_confirmed-pre_notify_demote_0
Cancel drbd0:0_monitor_10000 frigg
ms_drbd_demote_0
drbd0:0_demote_0 frigg
ms_drbd_demoted_0
ms_drbd_post_notify_demoted_0
drbd0:1_post_notify_demote_0 odin
drbd0:0_post_notify_demote_0 frigg
ms_drbd_confirmed-post_notify_demoted_0
ms_drbd_pre_notify_promote_0
drbd0:1_pre_notify_promote_0 odin
drbd0:0_pre_notify_promote_0 frigg
ms_drbd_confirmed-pre_notify_promote_0
ms_drbd_promote_0
Cancel drbd0:1_monitor_12000 odin
drbd0:1_promote_0 odin
ms_drbd_promoted_0
ms_drbd_post_notify_promoted_0
drbd0:1_post_notify_promote_0 odin
drbd0:0_post_notify_promote_0 frigg
ms_drbd_confirmed-post_notify_promoted_0
drbd0:0_monitor_12000 frigg
group_start_0
drbd0:1_monitor_10000 odin
IPaddr0_start_0 odin
MailTo_start_0 odin
IPaddr0_monitor_5000 odin
group_running_0

What-if scenarios

You can make changes to the saved or shadow CIB and simulate it again, to see how Pacemaker would react differently. You can edit the XML by hand, use command-line tools such as `cibadmin` with either a shadow CIB or the `CIB_file` environment variable set to the filename, or use higher-level tool support (see the man pages of the specific tool you're using for how to perform actions on a saved CIB file rather than the live CIB).

You can also inject node failures and/or action failures into the simulation; see the `crm_simulate` man page for more details.

This capability is useful when using a shadow CIB to edit the configuration. Before committing the changes to the live cluster with `crm_shadow --commit`, you can use `crm_simulate` to see how the cluster will react to the changes.

2.5.6 Manage Node Attributes, Cluster Options and Defaults with crm_attribute and attrd_updater

`crm_attribute` and `attrd_updater` are confusingly similar tools with subtle differences.

`attrd_updater` can query and update node attributes. `crm_attribute` can query and update not only node attributes, but also cluster options, resource defaults, and operation defaults.

To understand the differences, it helps to understand the various types of node attribute.

Table 3: **Types of Node Attributes**

Type	Recorded in CIB?	Recorded in attribute manager memory?	Survive full cluster restart?	Manageable by crm_attribute?	Manageable by attrd_updater?
permanent	yes	no	yes	yes	no
transient	yes	yes	no	yes	yes
private	no	yes	no	no	yes

As you can see from the table above, `crm_attribute` can manage permanent and transient node attributes, while `attrd_updater` can manage transient and private node attributes.

The difference between the two tools lies mainly in *how* they update node attributes: `attrd_updater` always contacts the Pacemaker attribute manager directly, while `crm_attribute` will contact the attribute manager only for transient node attributes, and will instead modify the CIB directly for permanent node attributes (and for transient node attributes when unable to contact the attribute manager).

By contacting the attribute manager directly, `attrd_updater` can change an attribute's "dampening" (whether changes are immediately flushed to the CIB or after a specified amount of time, to minimize disk writes for frequent changes), set private node attributes (which are never written to the CIB), and set attributes for nodes that don't yet exist.

By modifying the CIB directly, `crm_attribute` can set permanent node attributes (which are only in the CIB and not managed by the attribute manager), and can be used with saved CIB files and shadow CIBs.

However a transient node attribute is set, it is synchronized between the CIB and the attribute manager, on all nodes.

2.5.7 Other Commonly Used Tools

Other command-line tools include:

- `crm_failcount`: query or delete resource fail counts
- `crm_node`: manage cluster nodes
- `crm_report`: generate a detailed cluster report for bug submissions
- `crm_resource`: manage cluster resources
- `crm_standby`: manage standby status of nodes
- `crm_verify`: validate a CIB
- `stonith_admin`: manage fencing devices

See the manual pages for details.

2.6 Troubleshooting Cluster Problems

2.6.1 Logging

Pacemaker by default logs messages of notice severity and higher to the system log, and messages of info severity and higher to the detail log, which by default is `/var/log/pacemaker/pacemaker.log`.

Logging options can be controlled via environment variables at Pacemaker start-up. Where these are set varies by operating system (often `/etc/sysconfig/pacemaker` or `/etc/default/pacemaker`).

Because cluster problems are often highly complex, involving multiple machines, cluster daemons, and managed services, Pacemaker logs rather verbosely to provide as much context as possible. It is an ongoing priority to make these logs more user-friendly, but by necessity there is a lot of obscure, low-level information that can make them difficult to follow.

The default log rotation configuration shipped with Pacemaker (typically installed in `/etc/logrotate.d/pacemaker`) rotates the log when it reaches 100MB in size, or weekly, whichever comes first.

If you configure debug or (Heaven forbid) trace-level logging, the logs can grow enormous quite quickly. Because rotated logs are by default named with the year, month, and day only, this can cause name collisions if your logs exceed 100MB in a single day. You can add `dateformat -%Y%m%d-%H` to the rotation configuration to avoid this.

2.6.2 Transitions

A key concept in understanding how a Pacemaker cluster functions is a *transition*. A transition is a set of actions that need to be taken to bring the cluster from its current state to the desired state (as expressed by the configuration).

Whenever a relevant event happens (a node joining or leaving the cluster, a resource failing, etc.), the controller will ask the scheduler to recalculate the status of the cluster, which generates a new transition. The controller then performs the actions in the transition in the proper order.

Each transition can be identified in the logs by a line like:

The file listed as the "inputs" is a snapshot of the cluster configuration and state at that moment (the CIB). This file can help determine why particular actions were scheduled. The `crm_simulate` command, described in *Simulate Cluster Activity with crm_simulate*, can be used to replay the file.

2.6.3 Further Information About Troubleshooting

Andrew Beekhof wrote a series of articles about troubleshooting in his blog, The Cluster Guy:

- Debugging Pacemaker
- Debugging the Policy Engine
- Pacemaker Logging

The articles were written for an earlier version of Pacemaker, so many of the specific names and log messages to look for have changed, but the concepts are still valid.

2.7 Upgrading a Pacemaker Cluster

2.7.1 Pacemaker Versioning

Pacemaker has an overall release version, plus separate version numbers for certain internal components.

- **Pacemaker release version:** This version consists of three numbers (*x.y.z*).

  The major version number (the *x* in *x.y.z*) increases when at least some rolling upgrades are not possible from the previous major version. For example, a rolling upgrade from 1.0.8 to 1.1.15 should always be supported, but a rolling upgrade from 1.0.8 to 2.0.0 may not be possible.

  The minor version (the *y* in *x.y.z*) increases when there are significant changes in cluster default behavior, tool behavior, and/or the API interface (for software that utilizes Pacemaker libraries). The main benefit is to alert you to pay closer attention to the release notes, to see if you might be affected.

  The release counter (the *z* in *x.y.z*) is increased with all public releases of Pacemaker, which typically include both bug fixes and new features.

- **CRM feature set:** This version number applies to the communication between full cluster nodes, and is used to avoid problems in mixed-version clusters.

  The major version number increases when nodes with different versions would not work (rolling upgrades are not allowed). The minor version number increases when mixed-version clusters are allowed only during rolling upgrades. The minor-minor version number is ignored, but allows resource agents to detect cluster support for various features.[1]

  Pacemaker ensures that the longest-running node is the cluster's DC. This ensures new features are not enabled until all nodes are upgraded to support them.

- **Pacemaker Remote protocol version:** This version applies to communication between a Pacemaker Remote node and the cluster. It increases when an older cluster node would have problems hosting the connection to a newer Pacemaker Remote node. To avoid these problems, Pacemaker Remote nodes will accept connections only from cluster nodes with the same or newer Pacemaker Remote protocol version.

  Unlike with CRM feature set differences between full cluster nodes, mixed Pacemaker Remote protocol versions between Pacemaker Remote nodes and full cluster nodes are fine, as long as the Pacemaker Remote nodes have the older version. This can be useful, for example, to host a legacy application in an older operating system version used as a Pacemaker Remote node.

- **XML schema version:** Pacemaker's configuration syntax — what's allowed in the Configuration Information Base (CIB) — has its own version. This allows the configuration syntax to evolve over time while still allowing clusters with older configurations to work without change.

[1] Before CRM feature set 3.1.0 (Pacemaker 2.0.0), the minor-minor version number was treated the same as the minor version.

2.7.2 Upgrading Cluster Software

There are three approaches to upgrading a cluster, each with advantages and disadvantages.

Table 4: **Upgrade Methods**

Method	Available between all versions	Can be used with Pacemaker Remote nodes	Service outage during upgrade	Service recovery during upgrade	Exercises failover logic	Allows change of messaging layer[2]
Complete cluster shutdown	yes	yes	always	N/A	no	yes
Rolling (node by node)	no	yes	always[3]	yes	yes	no
Detach and reattach	yes	no	only due to failure	no	no	yes

Complete Cluster Shutdown

In this scenario, one shuts down all cluster nodes and resources, then upgrades all the nodes before restarting the cluster.

1. On each node:
   (a) Shutdown the cluster software (pacemaker and the messaging layer).
   (b) Upgrade the Pacemaker software. This may also include upgrading the messaging layer and/or the underlying operating system.
   (c) Check the configuration with the `crm_verify` tool.
2. On each node:
   (a) Start the cluster software.

Currently, only Corosync version 2 and greater is supported as the cluster layer, but if another stack is supported in the future, the stack does not need to be the same one before the upgrade.

One variation of this approach is to build a new cluster on new hosts. This allows the new version to be tested beforehand, and minimizes downtime by having the new nodes ready to be placed in production as soon as the old nodes are shut down.

Rolling (node by node)

In this scenario, each node is removed from the cluster, upgraded, and then brought back online, until all nodes are running the newest version.

Special considerations when planning a rolling upgrade:

- If you plan to upgrade other cluster software – such as the messaging layer – at the same time, consult that software's documentation for its compatibility with a rolling upgrade.

[2] Currently, Corosync version 2 and greater is the only supported cluster stack, but other stacks have been supported by past versions, and may be supported by future versions.

[3] Any active resources will be moved off the node being upgraded, so there will be at least a brief outage unless all resources can be migrated "live".

- If the major version number is changing in the Pacemaker version you are upgrading to, a rolling upgrade may not be possible. Read the new version's release notes (as well the information here) for what limitations may exist.
- If the CRM feature set is changing in the Pacemaker version you are upgrading to, you should run a mixed-version cluster only during a small rolling upgrade window. If one of the older nodes drops out of the cluster for any reason, it will not be able to rejoin until it is upgraded.
- If the Pacemaker Remote protocol version is changing, all cluster nodes should be upgraded before upgrading any Pacemaker Remote nodes.

See the ClusterLabs wiki's release calendar to figure out whether the CRM feature set and/or Pacemaker Remote protocol version changed between the the Pacemaker release versions in your rolling upgrade.

To perform a rolling upgrade, on each node in turn:

1. Put the node into standby mode, and wait for any active resources to be moved cleanly to another node. (This step is optional, but allows you to deal with any resource issues before the upgrade.)
2. Shutdown the cluster software (pacemaker and the messaging layer) on the node.
3. Upgrade the Pacemaker software. This may also include upgrading the messaging layer and/or the underlying operating system.
4. If this is the first node to be upgraded, check the configuration with the `crm_verify` tool.
5. Start the messaging layer. This must be the same messaging layer (currently only Corosync version 2 and greater is supported) that the rest of the cluster is using.

Note: Even if a rolling upgrade from the current version of the cluster to the newest version is not directly possible, it may be possible to perform a rolling upgrade in multiple steps, by upgrading to an intermediate version first.

Table 5: **Version Compatibility Table**

Version being Installed	Oldest Compatible Version
Pacemaker 2.y.z	Pacemaker 1.1.11[4]
Pacemaker 1.y.z	Pacemaker 1.0.0
Pacemaker 0.7.z	Pacemaker 0.6.z

Detach and Reattach

The reattach method is a variant of a complete cluster shutdown, where the resources are left active and get re-detected when the cluster is restarted.

This method may not be used if the cluster contains any Pacemaker Remote nodes.

1. Tell the cluster to stop managing services. This is required to allow the services to remain active after the cluster shuts down.

```
# crm_attribute --name maintenance-mode --update true
```

2. On each node, shutdown the cluster software (pacemaker and the messaging layer), and upgrade the Pacemaker software. This may also include upgrading the messaging layer. While the underlying

[4] Rolling upgrades from Pacemaker 1.1.z to 2.y.z are possible only if the cluster uses corosync version 2 or greater as its messaging layer, and the Cluster Information Base (CIB) uses schema 1.0 or higher in its `validate-with` property.

operating system may be upgraded at the same time, that will be more likely to cause outages in the detached services (certainly, if a reboot is required).

3. Check the configuration with the `crm_verify` tool.
4. On each node, start the cluster software. Currently, only Corosync version 2 and greater is supported as the cluster layer, but if another stack is supported in the future, the stack does not need to be the same one before the upgrade.
5. Verify that the cluster re-detected all resources correctly.
6. Allow the cluster to resume managing resources again:

```
# crm_attribute --name maintenance-mode --delete
```

Note: While the goal of the detach-and-reattach method is to avoid disturbing running services, resources may still move after the upgrade if any resource's location is governed by a rule based on transient node attributes. Transient node attributes are erased when the node leaves the cluster. A common example is using the `ocf:pacemaker:ping` resource to set a node attribute used to locate other resources.

2.7.3 Upgrading the Configuration

The CIB schema version can change from one Pacemaker version to another.

After cluster software is upgraded, the cluster will continue to use the older schema version that it was previously using. This can be useful, for example, when administrators have written tools that modify the configuration, and are based on the older syntax.[5]

However, when using an older syntax, new features may be unavailable, and there is a performance impact, since the cluster must do a non-persistent configuration upgrade before each transition. So while using the old syntax is possible, it is not advisable to continue using it indefinitely.

Even if you wish to continue using the old syntax, it is a good idea to follow the upgrade procedure outlined below, except for the last step, to ensure that the new software has no problems with your existing configuration (since it will perform much the same task internally).

If you are brave, it is sufficient simply to run `cibadmin --upgrade`.

A more cautious approach would proceed like this:

1. Create a shadow copy of the configuration. The later commands will automatically operate on this copy, rather than the live configuration.

```
# crm_shadow --create shadow
```

1. Verify the configuration is valid with the new software (which may be stricter about syntax mistakes, or may have dropped support for deprecated features):

```
# crm_verify --live-check
```

2. Fix any errors or warnings.
3. Perform the upgrade:

[5] As of Pacemaker 2.0.0, only schema versions pacemaker-1.0 and higher are supported (excluding pacemaker-1.1, which was a special case).

```
# cibadmin --upgrade
```

4. If this step fails, there are three main possibilities:

   (a) The configuration was not valid to start with (did you do steps 2 and 3?).

   (b) The transformation failed; report a bug.

   (c) The transformation was successful but produced an invalid result.

   If the result of the transformation is invalid, you may see a number of errors from the validation library. If these are not helpful, visit the Validation FAQ wiki page and/or try the manual upgrade procedure described below.

5. Check the changes:

   ```
   # crm_shadow --diff
   ```

   If at this point there is anything about the upgrade that you wish to fine-tune (for example, to change some of the automatic IDs), now is the time to do so:

   ```
   # crm_shadow --edit
   ```

   This will open the configuration in your favorite editor (whichever is specified by the standard `$EDITOR` environment variable).

6. Preview how the cluster will react:

   ```
   # crm_simulate --live-check --save-dotfile shadow.dot -S
   # dot -Tsvg shadow.dot -o shadow.svg
   ```

   You can then view shadow.svg with any compatible image viewer or web browser. Verify that either no resource actions will occur or that you are happy with any that are scheduled. If the output contains actions you do not expect (possibly due to changes to the score calculations), you may need to make further manual changes. See *Simulate Cluster Activity with crm_simulate* for further details on how to interpret the output of `crm_simulate` and `dot`.

7. Upload the changes:

   ```
   # crm_shadow --commit shadow --force
   ```

   In the unlikely event this step fails, please report a bug.

Note: It is also possible to perform the configuration upgrade steps manually:

1. Locate the `upgrade*.xsl` conversion scripts provided with the source code. These will often be installed in a location such as `/usr/share/pacemaker`, or may be obtained from the source repository.

2. Run the conversion scripts that apply to your older version, for example:

   ```
   # xsltproc /path/to/upgrade06.xsl config06.xml > config10.xml
   ```

3. Locate the `pacemaker.rng` script (from the same location as the xsl files).

4. Check the XML validity:

   ```
   # xmllint --relaxng /path/to/pacemaker.rng config10.xml
   ```

The advantage of this method is that it can be performed without the cluster running, and any validation errors are often more informative.

2.7.4 What Changed in 2.1

The Pacemaker 2.1 release is fully backward-compatible in both the CIB XML and the C API. Highlights:

- Pacemaker now supports the **OCF Resource Agent API version 1.1**. Most notably, the `Master` and `Slave` role names have been renamed to `Promoted` and `Unpromoted`.
- Pacemaker now supports colocations where the dependent resource does not affect the primary resource's placement (via a new `influence` colocation constraint option and `critical` resource meta-attribute). This is intended for cases where a less-important resource must be colocated with an essential resource, but it is preferred to leave the less-important resource stopped if it fails, rather than move both resources.
- If Pacemaker is built with libqb 2.0 or later, the detail log will use **millisecond-resolution timestamps**.
- In addition to crm_mon and stonith_admin, the crmadmin, crm_resource, crm_simulate, and crm_verify commands now support the `--output-as` and `--output-to` options, including **XML output** (which scripts and higher-level tools are strongly recommended to use instead of trying to parse the text output, which may change from release to release).

For a detailed list of changes, see the release notes and the Pacemaker 2.1 Changes page on the ClusterLabs wiki.

2.7.5 What Changed in 2.0

The main goal of the 2.0 release was to remove support for deprecated syntax, along with some small changes in default configuration behavior and tool behavior. Highlights:

- Only Corosync version 2 and greater is now supported as the underlying cluster layer. Support for Heartbeat and Corosync 1 (including CMAN) is removed.
- The Pacemaker detail log file is now stored in `/var/log/pacemaker/pacemaker.log` by default.
- The record-pending cluster property now defaults to true, which allows status tools such as crm_mon to show operations that are in progress.
- Support for a number of deprecated build options, environment variables, and configuration settings has been removed.
- The `master` tag has been deprecated in favor of using the `clone` tag with the new `promotable` meta-attribute set to `true`. "Master/slave" clone resources are now referred to as "promotable" clone resources.
- The public API for Pacemaker libraries that software applications can use has changed significantly.

For a detailed list of changes, see the release notes and the Pacemaker 2.0 Changes page on the ClusterLabs wiki.

2.7.6 What Changed in 1.0

New

- Failure timeouts.
- New section for resource and operation defaults.
- Tool for making offline configuration changes.
- `Rules`, `instance_attributes`, `meta_attributes` and sets of operations can be defined once and referenced in multiple places.
- The CIB now accepts XPath-based create/modify/delete operations. See `cibadmin --help`.
- Multi-dimensional colocation and ordering constraints.
- The ability to connect to the CIB from non-cluster machines.
- Allow recurring actions to be triggered at known times.

Changed

- Syntax
  - All resource and cluster options now use dashes (-) instead of underscores (_)
  - `master_slave` was renamed to `master`
  - The `attributes` container tag was removed
  - The operation field `pre-req` has been renamed `requires`
  - All operations must have an `interval`, `start`/`stop` must have it set to zero
- The `stonith-enabled` option now defaults to true.
- The cluster will refuse to start resources if `stonith-enabled` is true (or unset) and no STONITH resources have been defined
- The attributes of colocation and ordering constraints were renamed for clarity.
- `resource-failure-stickiness` has been replaced by `migration-threshold`.
- The parameters for command-line tools have been made consistent
- Switched to 'RelaxNG' schema validation and 'libxml2' parser
  - id fields are now XML IDs which have the following limitations:
    - id's cannot contain colons (:)
    - id's cannot begin with a number
    - id's must be globally unique (not just unique for that tag)
  - Some fields (such as those in constraints that refer to resources) are IDREFs.

    This means that they must reference existing resources or objects in order for the configuration to be valid. Removing an object which is referenced elsewhere will therefore fail.
  - The CIB representation, from which a MD5 digest is calculated to verify CIBs on the nodes, has changed.

    This means that every CIB update will require a full refresh on any upgraded nodes until the cluster is fully upgraded to 1.0. This will result in significant performance degradation and it is therefore highly inadvisable to run a mixed 1.0/0.6 cluster for any longer than absolutely necessary.

- Ping node information no longer needs to be added to `ha.cf`. Simply include the lists of hosts in your ping resource(s).

Removed

- Syntax
  - It is no longer possible to set resource meta options as top-level attributes. Use meta-attributes instead.
  - Resource and operation defaults are no longer read from `crm_config`.

2.8 Resource Agents

2.8.1 Action Completion

If one resource depends on another resource via constraints, the cluster will interpret an expected result as sufficient to continue with dependent actions. This may cause timing issues if the resource agent start returns before the service is not only launched but fully ready to perform its function, or if the resource agent stop returns before the service has fully released all its claims on system resources. At a minimum, the start or stop should not return before a status command would return the expected (started or stopped) result.

2.8.2 OCF Resource Agents

Location of Custom Scripts

OCF Resource Agents are found in `/usr/lib/ocf/resource.d/$PROVIDER`

When creating your own agents, you are encouraged to create a new directory under `/usr/lib/ocf/resource.d/` so that they are not confused with (or overwritten by) the agents shipped by existing providers.

So, for example, if you choose the provider name of big-corp and want a new resource named big-app, you would create a resource agent called `/usr/lib/ocf/resource.d/big-corp/big-app` and define a resource:

Actions

All OCF resource agents are required to implement the following actions.

Table 6: **Required Actions for OCF Agents**

Action	Description	Instructions
start	Start the resource	Return 0 on success and an appropriate error code otherwise. Must not report success until the resource is fully active.
stop	Stop the resource	Return 0 on success and an appropriate error code otherwise. Must not report success until the resource is fully stopped.
monitor	Check the resource's state	Exit 0 if the resource is running, 7 if it is stopped, and any other OCF exit code if it is failed. NOTE: The monitor script should test the state of the resource on the local machine only.
metadata	Describe the resource	Provide information about this resource in the XML format defined by the OCF standard. Exit with 0. NOTE: This is *not* required to be performed as root.

OCF resource agents may optionally implement additional actions. Some are used only with advanced resource types such as clones.

Table 7: **Optional Actions for OCF Resource Agents**

Action	Description	Instructions
validate-all	This should validate the instance parameters provided.	Return 0 if parameters are valid, 2 if not valid, and 6 if resource is not configured.
promote	Bring the local instance of a promotable clone resource to the promoted role.	Return 0 on success
demote	Bring the local instance of a promotable clone resource to the unpromoted role.	Return 0 on success
notify	Used by the cluster to send the agent pre- and post- notification events telling the resource what has happened and will happen.	Must not fail. Must exit with 0
reload	Reload the service's own config.	Not used by Pacemaker
reload-agent	Make effective any changes in instance parameters marked as reloadable in the agent's metadata.	This is used when the agent can handle a change in some of its parameters more efficiently than stopping and starting the resource.
recover	Restart the service.	Not used by Pacemaker

Important: If you create a new OCF resource agent, use *ocf-tester* to verify that the agent complies with the OCF standard properly.

How are OCF Return Codes Interpreted?

The first thing the cluster does is to check the return code against the expected result. If the result does not match the expected value, then the operation is considered to have failed, and recovery action is initiated.

There are three types of failure recovery:

Table 8: **Types of recovery performed by the cluster**

Type	Description	Action Taken by the Cluster
soft	A transient error occurred	Restart the resource or move it to a new location
hard	A non-transient error that may be specific to the current node	Move the resource elsewhere and prevent it from being retried on the current node
fatal	A non-transient error that will be common to all cluster nodes (e.g. a bad configuration was specified)	Stop the resource and prevent it from being started on any cluster node

OCF Return Codes

The following table outlines the different OCF return codes and the type of recovery the cluster will initiate when a failure code is received. Although counterintuitive, even actions that return 0 (aka. `OCF_SUCCESS`) can be considered to have failed, if 0 was not the expected return value.

Table 9: **OCF Exit Codes and their Recovery Types**

Exit Code	OCF Alias	Description	Recovery
0	OCF_SUCCESS	Success. The command completed successfully. This is the expected result for all start, stop, promote and demote commands.	soft
1	OCF_ERR_GENERIC	Generic "there was a problem" error code.	soft
2	OCF_ERR_ARGS	The resource's parameter values are not valid on this machine (for example, a value refers to a file not found on the local host).	hard
3	OCF_ERR_UNIMPLEMENTED	The requested action is not implemented.	hard
4	OCF_ERR_PERM	The resource agent does not have sufficient privileges to complete the task.	hard
5	OCF_ERR_INSTALLED	The tools required by the resource are not installed on this machine.	hard
6	OCF_ERR_CONFIGURED	The resource's parameter values are inherently invalid (for example, a required parameter was not given).	fatal
7	OCF_NOT_RUNNING	The resource is safely stopped. This should only be returned by monitor actions, not stop actions.	N/A
8	OCF_RUNNING_PROMOTED	The resource is running in the promoted role.	soft
9	OCF_FAILED_PROMOTED	The resource is (or might be) in the promoted role but has failed. The resource will be demoted, stopped and then started (and possibly promoted) again.	soft
190	OCF_DEGRADED	The resource is properly active, but in such a condition that future failures are more likely.	none
191	OCF_DEGRADED_PROMOTED	The resource is properly active in the promoted role, but in such a condition that future failures are more likely.	none
other	*none*	Custom error code.	soft

Exceptions to the recovery handling described above:

- Probes (non-recurring monitor actions) that find a resource active (or in the promoted role) will not result in recovery action unless it is also found active elsewhere.
- The recovery action taken when a resource is found active more than once is determined by the resource's `multiple-active` property.
- Recurring actions that return `OCF_ERR_UNIMPLEMENTED` do not cause any type of recovery.
- Actions that return one of the "degraded" codes will be treated the same as if they had returned success, but status output will indicate that the resource is degraded.

2.8.3 LSB Resource Agents (Init Scripts)

LSB Compliance

The relevant part of the LSB specifications includes a description of all the return codes listed here.

Assuming *some_service* is configured correctly and currently inactive, the following sequence will help you determine if it is LSB-compatible:

1. Start (stopped):

```
# /etc/init.d/some_service start ; echo "result: $?"
```

   - Did the service start?
   - Did the echo command print `result: 0` (in addition to the init script's usual output)?

2. Status (running):

```
# /etc/init.d/some_service status ; echo "result: $?"
```

   - Did the script accept the command?
   - Did the script indicate the service was running?
   - Did the echo command print `result: 0` (in addition to the init script's usual output)?

3. Start (running):

```
# /etc/init.d/some_service start ; echo "result: $?"
```

   - Is the service still running?
   - **Did the echo command print `result: 0` (in addition to the init** script's usual output)?

4. Stop (running):

```
# /etc/init.d/some_service stop ; echo "result: $?"
```

   - Was the service stopped?
   - Did the echo command print `result: 0` (in addition to the init script's usual output)?

5. Status (stopped):

```
# /etc/init.d/some_service status ; echo "result: $?"
```

   - Did the script accept the command?
   - Did the script indicate the service was not running?

- Did the echo command print `result: 3` (in addition to the init script's usual output)?

6. Stop (stopped):

```
# /etc/init.d/some_service stop ; echo "result: $?"
```

   - Is the service still stopped?
   - Did the echo command print `result: 0` (in addition to the init script's usual output)?

7. Status (failed):

   This step is not readily testable and relies on manual inspection of the script.

   The script can use one of the error codes (other than 3) listed in the LSB spec to indicate that it is active but failed. This tells the cluster that before moving the resource to another node, it needs to stop it on the existing one first.

If the answer to any of the above questions is no, then the script is not LSB-compliant. Your options are then to either fix the script or write an OCF agent based on the existing script.

2.9 Quick Comparison of pcs and crm shell

`pcs` and `crm shell` are two popular higher-level command-line interfaces to Pacemaker. Each has its own syntax; this chapter gives a quick comparion of how to accomplish the same tasks using either one. Some examples also show the equivalent command using low-level Pacmaker command-line tools.

These examples show the simplest syntax; see the respective man pages for all possible options.

2.9.1 Show Cluster Configuration and Status

Show Configuration (Raw XML)

```
crmsh     # crm configure show xml
pcs       # pcs cluster cib
pacemaker # cibadmin -Q
```

Show Configuration (Human-friendly)

```
crmsh # crm configure show
pcs   # pcs config
```

Show Cluster Status

```
crmsh     # crm status
pcs       # pcs status
pacemaker # crm_mon -1
```

2.9.2 Manage Nodes

Put node "pcmk-1" in standby mode

```
crmsh     # crm node standby pcmk-1
pcs-0.9   # pcs cluster standby pcmk-1
pcs-0.10  # pcs node standby pcmk-1
pacemaker # crm_standby -N pcmk-1 -v on
```

Remove node "pcmk-1" from standby mode

```
crmsh     # crm node online pcmk-1
pcs-0.9   # pcs cluster unstandby pcmk-1
pcs-0.10  # pcs node unstandby pcmk-1
pacemaker # crm_standby -N pcmk-1 -v off
```

2.9.3 Manage Cluster Properties

Set the "stonith-enabled" cluster property to "false"

```
crmsh     # crm configure property stonith-enabled=false
pcs       # pcs property set stonith-enabled=false
pacemaker # crm_attribute -n stonith-enabled -v false
```

2.9.4 Show Resource Agent Information

List Resource Agent (RA) Classes

```
crmsh    # crm ra classes
pcs      # pcs resource standards
pacmaker # crm_resource --list-standards
```

List Available Resource Agents (RAs) by Standard

```
crmsh     # crm ra list ocf
pcs       # pcs resource agents ocf
pacemaker # crm_resource --list-agents ocf
```

List Available Resource Agents (RAs) by OCF Provider

```
crmsh     # crm ra list ocf pacemaker
pcs       # pcs resource agents ocf:pacemaker
pacemaker # crm_resource --list-agents ocf:pacemaker
```

List Available Resource Agent Parameters

```
crmsh     # crm ra info IPaddr2
pcs       # pcs resource describe IPaddr2
pacemaker # crm_resource --show-metadata ocf:heartbeat:IPaddr2
```

You can also use the full `class:provider:type` format with crmsh and pcs if multiple RAs with the same name are available.

Show Available Fence Agent Parameters

```
crmsh # crm ra info stonith:fence_ipmilan
pcs   # pcs stonith describe fence_ipmilan
```

2.9.5 Manage Resources

Create a Resource

```
crmsh # crm configure primitive ClusterIP ocf:heartbeat:IPaddr2 \
        params ip=192.168.122.120 cidr_netmask=24 \
        op monitor interval=30s
pcs   # pcs resource create ClusterIP IPaddr2 ip=192.168.122.120 cidr_netmask=24
```

pcs determines the standard and provider (`ocf:heartbeat`) automatically since `IPaddr2` is unique, and automatically creates operations (including monitor) based on the agent's meta-data.

Show Configuration of All Resources

```
crmsh    # crm configure show
pcs-0.9  # pcs resource show --full
pcs-0.10 # pcs resource config
```

Show Configuration of One Resource

```
crmsh    # crm configure show ClusterIP
pcs-0.9  # pcs resource show ClusterIP
pcs-0.10 # pcs resource config ClusterIP
```

Show Configuration of Fencing Resources

```
crmsh    # crm resource status
pcs-0.9  # pcs stonith show --full
pcs-0.10 # pcs stonith config
```

Start a Resource

```
crmsh     # crm resource start ClusterIP
pcs       # pcs resource enable ClusterIP
pacemaker # crm_resource -r ClusterIP --set-parameter target-role --meta -v Started
```

Stop a Resource

```
crmsh     # crm resource stop ClusterIP
pcs       # pcs resource disable ClusterIP
pacemaker # crm_resource -r ClusterIP --set-parameter target-role --meta -v Stopped
```

Remove a Resource

```
crmsh # crm configure delete ClusterIP
pcs   # pcs resource delete ClusterIP
```

Modify a Resource's Instance Parameters

```
crmsh     # crm resource param ClusterIP set clusterip_hash=sourceip
pcs       # pcs resource update ClusterIP clusterip_hash=sourceip
pacemaker # crm_resource -r ClusterIP --set-parameter clusterip_hash -v sourceip
```

crmsh also has an *edit* command which edits the simplified CIB syntax (same commands as the command line) via a configurable text editor.

Modify a Resource's Instance Parameters Interactively

```
crmsh # crm configure edit ClusterIP
```

Using the interactive shell mode of crmsh, multiple changes can be edited and verified before committing to the live configuration:

Make Multiple Configuration Changes Interactively

```
crmsh # crm configure
crmsh # edit
crmsh # verify
crmsh # commit
```

Delete a Resource's Instance Parameters

```
crmsh     # crm resource param ClusterIP delete nic
pcs       # pcs resource update ClusterIP nic=
pacemaker # crm_resource -r ClusterIP --delete-parameter nic
```

List Current Resource Defaults

```
crmsh     # crm configure show type:rsc_defaults
pcs       # pcs resource defaults
pacemaker # cibadmin -Q --scope rsc_defaults
```

Set Resource Defaults

```
crmsh # crm configure rsc_defaults resource-stickiness=100
pcs   # pcs resource defaults resource-stickiness=100
```

List Current Operation Defaults

```
crmsh     # crm configure show type:op_defaults
pcs       # pcs resource op defaults
pacemaker # cibadmin -Q --scope op_defaults
```

Set Operation Defaults

```
crmsh # crm configure op_defaults timeout=240s
pcs   # pcs resource op defaults timeout=240s
```

Enable Resource Agent Tracing for a Resource

```
crmsh # crm resource trace Website
```

Clear Fail Counts for a Resource

```
crmsh     # crm resource cleanup Website
pcs       # pcs resource cleanup Website
pacemaker # crm_resource --cleanup -r Website
```

Create a Clone Resource

```
crmsh # crm configure clone WebIP ClusterIP meta globally-unique=true clone-max=2 clone-node-
↪max=2
pcs   # pcs resource clone ClusterIP globally-unique=true clone-max=2 clone-node-max=2
```

Create a Promotable Clone Resource

```
crmsh    # crm configure ms WebDataClone WebData \
           meta master-max=1 master-node-max=1 \
           clone-max=2 clone-node-max=1 notify=true
pcs-0.9  # pcs resource master WebDataClone WebData \
           master-max=1 master-node-max=1 \
           clone-max=2 clone-node-max=1 notify=true
pcs-0.10 # pcs resource promotable WebData WebDataClone \
           promoted-max=1 promoted-node-max=1 \
           clone-max=2 clone-node-max=1 notify=true
```

pcs will generate the clone name automatically if it is omitted from the command line.

2.9.6 Manage Constraints

Create a Colocation Constraint

```
crmsh # crm configure colocation website-with-ip INFINITY: WebSite ClusterIP
pcs   # pcs constraint colocation add ClusterIP with WebSite INFINITY
```

Create a Colocation Constraint Based on Role

```
crmsh # crm configure colocation another-ip-with-website inf: AnotherIP WebSite:Master
pcs   # pcs constraint colocation add Started AnotherIP with Promoted WebSite INFINITY
```

Create an Ordering Constraint

```
crmsh # crm configure order apache-after-ip mandatory: ClusterIP WebSite
pcs   # pcs constraint order ClusterIP then WebSite
```

Create an Ordering Constraint Based on Role

```
crmsh # crm configure order ip-after-website Mandatory: WebSite:Master AnotherIP
pcs   # pcs constraint order promote WebSite then start AnotherIP
```

Create a Location Constraint

```
crmsh # crm configure location prefer-pcmk-1 WebSite 50: pcmk-1
pcs   # pcs constraint location WebSite prefers pcmk-1=50
```

Create a Location Constraint Based on Role

```
crmsh # crm configure location prefer-pcmk-1 WebSite rule role=Master 50: \#uname eq pcmk-1
pcs   # pcs constraint location WebSite rule role=Promoted 50 \#uname eq pcmk-1
```

Move a Resource to a Specific Node (by Creating a Location Constraint)

```
crmsh     # crm resource move WebSite pcmk-1
pcs       # pcs resource move WebSite pcmk-1
pacemaker # crm_resource -r WebSite --move -N pcmk-1
```

Move a Resource Away from Its Current Node (by Creating a Location Constraint)

```
crmsh     # crm resource ban Website pcmk-2
pcs       # pcs resource ban Website pcmk-2
pacemaker # crm_resource -r WebSite --move
```

Remove any Constraints Created by Moving a Resource

```
crmsh     # crm resource unmove WebSite
pcs       # pcs resource clear WebSite
pacemaker # crm_resource -r WebSite --clear
```

2.9.7 Advanced Configuration

Manipulate Configuration Elements by Type

List Constraints with IDs

```
pcs   # pcs constraint list --full
```

Remove Constraint by ID

```
pcs   # pcs constraint remove cli-ban-Website-on-pcmk-1
crmsh # crm configure remove cli-ban-Website-on-pcmk-1
```

crmsh's *show* and *edit* commands can be used to manage resources and constraints by type:

Show Configuration Elements

```
crmsh # crm configure show type:primitive
crmsh # crm configure edit type:colocation
```

Batch Changes

Make Multiple Changes and Apply Together

```
crmsh # crm
crmsh # cib new drbd_cfg
crmsh # configure primitive WebData ocf:linbit:drbd params drbd_resource=wwwdata \
        op monitor interval=60s
crmsh # configure ms WebDataClone WebData meta master-max=1 master-node-max=1 \
        clone-max=2 clone-node-max=1 notify=true
crmsh # cib commit drbd_cfg
crmsh # quit

pcs      # pcs cluster cib drbd_cfg
pcs      # pcs -f drbd_cfg resource create WebData ocf:linbit:drbd drbd_resource=wwwdata \
           op monitor interval=60s
pcs-0.9  # pcs -f drbd_cfg resource master WebDataClone WebData \
           master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true
pcs-0.10 # pcs -f drbd_cfg resource promotable WebData WebDataClone \
           promoted-max=1 promoted-node-max=1 clone-max=2 clone-node-max=1 notify=true
pcs      # pcs cluster cib-push drbd_cfg
```

Template Creation

Create Resource Template Based on Existing Primitives of Same Type

```
crmsh # crm configure assist template ClusterIP AdminIP
```

Log Analysis

Show Information About Recent Cluster Events

```
crmsh # crm history
crmsh # peinputs
crmsh # transition pe-input-10
crmsh # transition log pe-input-10
```

Configuration Scripts

Script Multiple-step Cluster Configurations

```
crmsh # crm script show apache
crmsh # crm script run apache \
        id=WebSite \
        install=true \
        virtual-ip:ip=192.168.0.15 \
        database:id=WebData \
        database:install=true
```

CHAPTER

THREE

INDEX

- genindex
- search

INDEX

V