

Pacemaker 1.1

Configuration Explained

An A-Z guide to Pacemaker's Configuration Options



Andrew Beekhof

Pacemaker 1.1 Configuration Explained

An A-Z guide to Pacemaker's Configuration Options

Edition 7

Author	Andrew Beekhof	andrew@beekhof.net
Translator	Dan Frîncu	df.cluster@gmail.com
	Philipp Marek	philipp.marek@linbit.com
	Tanja Roth	taroth@suse.com
	Lars Marowsky-Bree	lmb@suse.com
	Yan Gao	ygao@suse.com
	Thomas Schraitle	toms@suse.com
	Dejan Muhamedagic	dmuhamedagic@suse.com

Copyright © 2009-2015 Andrew Beekhof.

The text of and illustrations in this document are licensed under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA")¹.

In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

In addition to the requirements of this license, the following activities are looked upon favorably:

1. If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.
2. All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.
3. Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy or CD-ROM expression of the author(s) work.

The purpose of this document is to definitively explain the concepts used to configure Pacemaker. To achieve this, it will focus exclusively on the XML syntax used to configure Pacemaker's Cluster Information Base (CIB).

¹ An explanation of CC-BY-SA is available at <http://creativecommons.org/licenses/by-sa/3.0/>

Table of Contents

Preface	xvii
1. Document Conventions	xvii
1.1. Typographic Conventions	xvii
1.2. Pull-quote Conventions	xviii
1.3. Notes and Warnings	xix
2. We Need Feedback!	xix
1. Read-Me-First	1
1.1. The Scope of this Document	1
1.2. What Is <i>Pacemaker</i> ?	1
1.3. Pacemaker Architecture	2
1.3.1. Internal Components	3
1.4. Types of Pacemaker Clusters	4
2. Configuration Basics	7
2.1. Configuration Layout	7
2.2. The Current State of the Cluster	8
2.3. How Should the Configuration be Updated?	9
2.3.1. Editing the CIB Using XML	9
2.3.2. Quickly Deleting Part of the Configuration	10
2.3.3. Updating the Configuration Without Using XML	10
2.4. Making Configuration Changes in a Sandbox	11
2.5. Testing Your Configuration Changes	12
2.5.1. Small Cluster Transition	13
2.5.2. Complex Cluster Transition	14
2.6. Do I Need to Update the Configuration on All Cluster Nodes?	14
3. Cluster-Wide Configuration	17
3.1. CIB Properties	17
3.1.1. Working with CIB Properties	18
3.2. Cluster Options	18
3.2.1. Querying and Setting Cluster Options	21
3.2.2. When Options are Listed More Than Once	21
4. Cluster Nodes	23
4.1. Defining a Cluster Node	23
4.2. Where Pacemaker Gets the Node Name	23
4.3. Node Attributes	24
4.4. Managing Nodes in a Corosync-Based Cluster	24
4.4.1. Adding a New Corosync Node	24
4.4.2. Removing a Corosync Node	25
4.4.3. Replacing a Corosync Node	25
4.5. Managing Nodes in a Heartbeat-based Cluster	25
4.5.1. Adding a New Heartbeat Node	25
4.5.2. Removing a Heartbeat Node	26
4.5.3. Replacing a Heartbeat Node	26
5. Cluster Resources	27
5.1. What is a Cluster Resource?	27
5.2. Resource Classes	27
5.2.1. Open Cluster Framework	28
5.2.2. Linux Standard Base	28
5.2.3. Systemd	29
5.2.4. Upstart	29

5.2.5. System Services	30
5.2.6. STONITH	30
5.2.7. Nagios Plugins	30
5.3. Resource Properties	31
5.4. Resource Options	31
5.4.1. Resource Meta-Attributes	31
5.4.2. Setting Global Defaults for Resource Meta-Attributes	34
5.4.3. Resource Instance Attributes	34
5.5. Resource Operations	36
5.5.1. Monitoring Resources for Failure	37
5.5.2. Monitoring Resources When Administration is Disabled	37
5.5.3. Setting Global Defaults for Operations	38
5.5.4. When Implicit Operations Take a Long Time	38
5.5.5. Multiple Monitor Operations	38
5.5.6. Disabling a Monitor Operation	39
6. Resource Constraints	41
6.1. Scores	41
6.1.1. Infinity Math	41
6.2. Deciding Which Nodes a Resource Can Run On	42
6.2.1. Location Properties	42
6.2.2. Asymmetrical "Opt-In" Clusters	43
6.2.3. Symmetrical "Opt-Out" Clusters	44
6.2.4. What if Two Nodes Have the Same Score	44
6.3. Specifying the Order in which Resources Should Start/Stop	44
6.3.1. Ordering Properties	45
6.3.2. Optional and mandatory ordering	45
6.4. Placing Resources Relative to other Resources	46
6.4.1. Colocation Properties	46
6.4.2. Mandatory Placement	47
6.4.3. Advisory Placement	47
6.5. Resource Sets	48
6.6. Ordering Sets of Resources	48
6.6.1. Ordered Set	49
6.6.2. Ordering Multiple Sets	49
6.6.3. Resource Set OR Logic	51
6.7. Colocating Sets of Resources	51
7. Alerts	55
7.1. Alert Agents	55
7.2. Alert Recipients	55
7.3. Alert Meta-Attributes	56
7.4. Alert Instance Attributes	56
7.5. Using the Sample Alert Agents	57
7.6. Writing an Alert Agent	58
8. Rules	61
8.1. Rule Properties	61
8.2. Node Attribute Expressions	62
8.3. Time- and Date-Based Expressions	63
8.3.1. Date Specifications	63
8.3.2. Durations	64
8.3.3. Sample Time-Based Expressions	64
8.4. Using Rules to Determine Resource Location	66
8.4.1. Location Rules Based on Other Node Properties	66

8.4.2. Using score-attribute Instead of score	67
8.5. Using Rules to Control Resource Options	67
8.6. Using Rules to Control Cluster Options	68
8.7. Ensuring Time-Based Rules Take Effect	68
9. Advanced Configuration	69
9.1. Connecting from a Remote Machine	69
9.2. Specifying When Recurring Actions are Performed	70
9.3. Moving Resources	70
9.3.1. Moving Resources Manually	70
9.3.2. Moving Resources Due to Failure	72
9.3.3. Moving Resources Due to Connectivity Changes	73
9.3.4. Migrating Resources	75
9.4. Reusing Rules, Options and Sets of Operations	76
9.5. Reloading Services After a Definition Change	77
10. Advanced Resource Types	81
10.1. Groups - A Syntactic Shortcut	81
10.1.1. Group Properties	82
10.1.2. Group Options	82
10.1.3. Group Instance Attributes	82
10.1.4. Group Contents	82
10.1.5. Group Constraints	83
10.1.6. Group Stickiness	83
10.2. Clones - Resources That Get Active on Multiple Hosts	83
10.2.1. Clone Properties	84
10.2.2. Clone Options	84
10.2.3. Clone Instance Attributes	84
10.2.4. Clone Contents	84
10.2.5. Clone Constraints	85
10.2.6. Clone Stickiness	85
10.2.7. Clone Resource Agent Requirements	85
10.3. Multi-state - Resources That Have Multiple Modes	87
10.3.1. Multi-state Properties	88
10.3.2. Multi-state Options	88
10.3.3. Multi-state Instance Attributes	88
10.3.4. Multi-state Contents	88
10.3.5. Monitoring Multi-State Resources	88
10.3.6. Multi-state Constraints	89
10.3.7. Multi-state Stickiness	91
10.3.8. Which Resource Instance is Promoted	91
10.3.9. Requirements for Multi-state Resource Agents	91
11. Utilization and Placement Strategy	97
11.1. Utilization attributes	97
11.2. Placement Strategy	98
11.3. Allocation Details	99
11.3.1. Which node is preferred to get consumed first when allocating resources?	99
11.3.2. Which node has more free capacity?	99
11.3.3. Which resource is preferred to be assigned first?	99
11.4. Limitations and Workarounds	100
12. Resource Templates	101
12.1. Configuring Resources with Templates	101
12.2. Referencing Templates in Constraints	103
12.2.1. Referencing Resource Templates in Sequential Resource Sets	103

12.2.2. Referencing Resource Templates in Parallel Resource Sets	104
13. STONITH	105
13.1. What Is STONITH?	105
13.2. What STONITH Device Should You Use?	105
13.3. Special Treatment of STONITH Resources	105
13.4. Configuring STONITH	110
13.4.1. Example STONITH Configuration	111
13.5. Advanced STONITH Configurations	113
13.5.1. Example Dual-Layer, Dual-Device Fencing Topologies	114
13.6. Remapping Reboots	119
14. Status—Here be dragons	121
14.1. Node Status	121
14.2. Transient Node Attributes	122
14.3. Operation History	122
14.3.1. Simple Operation History Example	124
14.3.2. Complex Operation History Example	125
15. Multi-Site Clusters and Tickets	127
15.1. Challenges for Multi-Site Clusters	127
15.2. Conceptual Overview	127
15.2.1. Ticket	127
15.2.2. Dead Man Dependency	128
15.2.3. Cluster Ticket Registry	128
15.2.4. Configuration Replication	128
15.3. Configuring Ticket Dependencies	129
15.4. Managing Multi-Site Clusters	130
15.4.1. Granting and Revoking Tickets Manually	130
15.4.2. Granting and Revoking Tickets via a Cluster Ticket Registry	130
15.4.3. General Management of Tickets	131
15.5. For more information	132
A. FAQ	133
Frequently Asked Questions	133
B. More About OCF Resource Agents	135
B.1. Location of Custom Scripts	135
B.2. Actions	135
B.3. How are OCF Return Codes Interpreted?	136
B.4. OCF Return Codes	136
C. What Changed in 1.0	139
C.1. New	139
C.2. Changed	139
C.3. Removed	140
D. Installing	141
D.1. Installing the Software	141
D.2. Enabling Pacemaker	141
D.2.1. Enabling Pacemaker For Corosync 2.x	141
D.2.2. Enabling Pacemaker For Corosync 1.x	142
D.2.3. Enabling Pacemaker For Heartbeat	143
E. Upgrading Cluster Software	145
E.1. Complete Cluster Shutdown	145
E.2. Rolling (node by node)	146
E.3. Disconnect and Reattach	146

F. Upgrading the Configuration	149
F.1. Perform the upgrade	149
F.1.1. Upgrade the software	149
F.1.2. Upgrade the Configuration	149
F.1.3. Manually Upgrading the Configuration	150
G. Init Script LSB Compliance	153
H. Sample Configurations	155
H.1. Empty	155
H.2. Simple	155
H.3. Advanced Configuration	156
I. Further Reading	159
J. Revision History	161
Index	163

List of Figures

1.1. The Pacemaker Stack	3
1.2. Internal Components	3
1.3. Active/Passive Redundancy	4
1.4. Shared Failover	5
1.5. N to N Redundancy	5
6.1. Visual representation of the four resources' start order for the above constraints	48
6.2. Visual representation of the start order for two ordered sets of unordered resources	50
6.3. Visual representation of the start order for the three sets defined above	50
6.4. Visual representation of a colocation chain where the members of the middle set have no inter-dependencies	53

List of Tables

3.1. CIB Properties	17
3.2. Cluster Options	18
5.1. Properties of a Primitive Resource	31
5.2. Meta-attributes of a Primitive Resource	32
5.3. Properties of an Operation	36
6.1. Properties of a <code>rsc_location</code> Constraint	42
6.2. Properties of a <code>rsc_order</code> Constraint	45
6.3. Properties of a <code>rsc_colocation</code> Constraint	46
6.4. Properties of a <code>resource_set</code>	48
7.1. Meta-Attributes of an Alert	56
7.2. Environment variables passed to alert agents	58
8.1. Properties of a Rule	61
8.2. Properties of an Expression	62
8.3. Built-in node attributes	62
8.4. Properties of a Date Expression	63
8.5. Properties of a Date Specification	63
9.1. Environment Variables Used to Connect to Remote Instances of the CIB	69
9.2. Extra top-level CIB properties for remote access	69
9.3. Common Options for a <i>ping</i> Resource	73
10.1. Properties of a Group Resource	82
10.2. Properties of a Clone Resource	84
10.3. Clone-specific configuration options	84
10.4. Environment variables supplied with Clone notify actions	86
10.5. Properties of a Multi-State Resource	88
10.6. Multi-state-specific resource configuration options	88
10.7. Additional colocation constraint options for multi-state resources	89
10.8. Additional colocation set options relevant to multi-state resources	90
10.9. Additional ordered set options relevant to multi-state resources	90
10.10. Role implications of OCF return codes	91
10.11. Environment variables supplied with multi-state notify actions	92
13.1. Properties of Fencing Resources	106
13.2. Properties of Fencing Levels	113
14.1. Authoritative Sources for State Information	121
14.2. Node Status Fields	121
14.3. Contents of an <code>lrm_rsc_op</code> job	123
B.1. Required Actions for OCF Agents	135
B.2. Optional Actions for OCF Resource Agents	136
B.3. Types of recovery performed by the cluster	136
B.4. OCF Return Codes and their Recovery Types	136
E.1. Upgrade Methods	145
E.2. Version Compatibility Table	146

List of Examples

2.1. An empty configuration	7
2.2. Sample output from <code>crm_mon</code>	8
2.3. Sample output from <code>crm_mon -n</code>	8
2.4. Safely using an editor to modify the cluster configuration	9
2.5. Safely using an editor to modify only the resources section	10
2.6. Searching for STONITH-related configuration items	10
2.7. Creating and displaying the active sandbox	11
2.8. Using a sandbox to make multiple changes atomically, discard them and verify the real configuration is untouched	11
3.1. Attributes set for a cib object	18
3.2. Deleting an option that is listed twice	21
4.1. Example Heartbeat cluster node entry	23
4.2. Example Corosync cluster node entry	23
4.3. Result of using <code>crm_attribute</code> to specify which kernel <code>pcmk-1</code> is running	24
5.1. A system resource definition	31
5.2. An OCF resource definition	31
5.3. An LSB resource with cluster options	34
5.4. An example OCF resource with instance attributes	34
5.5. Displaying the metadata for the Dummy resource agent template	35
5.6. An OCF resource with a recurring health check	36
5.7. An OCF resource with custom timeouts for its implicit actions	38
5.8. An OCF resource with two recurring health checks, performing different levels of checks specified via OCF_CHECK_LEVEL	39
5.9. Example of an OCF resource with a disabled health check	39
6.1. Opt-in location constraints for two resources	43
6.2. Opt-out location constraints for two resources	44
6.3. Constraints where a resource prefers two nodes equally	44
6.4. Optional and mandatory ordering constraints	46
6.5. Mandatory colocation constraint for two resources	47
6.6. Mandatory anti-colocation constraint for two resources	47
6.7. Advisory colocation constraint for two resources	47
6.8. A set of 3 resources	48
6.9. A chain of ordered resources	48
6.10. A chain of ordered resources expressed as a set	49
6.11. Ordered sets of unordered resources	49
6.12. Advanced use of set ordering - Three ordered sets, two of which are internally unordered	50
6.13. Resource Set "OR" logic: Three ordered sets, where the first set is internally unordered with "OR" logic	51
6.14. Chain of colocated resources	52
6.15. Equivalent colocation chain expressed using resource_set	52
6.16. Using colocated sets to specify a common peer	52
6.17. A colocation chain where the members of the middle set have no interdependencies and the last has master status.	53
7.1. Simple alert configuration	55
7.2. Alert configuration with recipient	55
7.3. Alert configuration with meta-attributes	56
7.4. Alert configuration with instance attributes	57
7.5. Sending cluster events as SNMP traps	57
7.6. Sending cluster events as e-mails	57
8.1. True if now is any time in the year 2005	64
8.2. Equivalent expression	64

8.3. 9am-5pm Monday-Friday	64
8.4. 9am-6pm Monday through Friday or anytime Saturday	65
8.5. 9am-5pm or 9pm-12am Monday through Friday	65
8.6. Mondays in March 2005	65
8.7. A full moon on Friday the 13th	65
8.8. Prevent myApacheRsc from running on c001n03	66
8.9. Prevent myApacheRsc from running on c001n03 - expanded version	66
8.10. A sample nodes section for use with score-attribute	66
8.11. Defining different resource options based on the node name	67
8.12. Change resource-stickiness during working hours	68
9.1. Specifying a Base for Recurring Action Intervals	70
9.2. An example ping cluster resource that checks node connectivity once every minute	73
9.3. Don't run a resource on unconnected nodes	74
9.4. Run only on nodes connected to three or more ping targets.	74
9.5. Prefer the node with the most connected ping nodes	75
9.6. How the cluster translates the above location constraint	75
9.7. A more complex example of choosing a location based on connectivity	75
9.8. Referencing rules from other constraints	76
9.9. Referencing attributes, options, and operations from other resources	77
9.10. The DRBD agent's logic for supporting reload	77
9.11. The DRBD Agent Advertising Support for the reload Operation	78
9.12. Parameter that can be changed using reload	78
10.1. A group of two primitive resources	81
10.2. How the cluster sees a group resource	82
10.3. Some constraints involving groups	83
10.4. A clone of an LSB resource	83
10.5. Some constraints involving clones	85
10.6. Notification variables	86
10.7. Monitoring both states of a multi-state resource	88
10.8. Constraints involving multi-state resources	89
10.9. Colocate C and D with A's and B's master instances	90
10.10. Start C and D after first promoting A and B	90
10.11. Explicitly preferring node1 to be promoted to master	91
11.1. Specifying CPU and RAM capacities of two nodes	97
11.2. Specifying CPU and RAM consumed by several resources	98
12.1. Resource template for a migratable Xen virtual machine	101
12.2. Xen primitive resource using a resource template	101
12.3. Equivalent Xen primitive resource not using a resource template	102
12.4. Xen resource overriding template values	102
13.1. Obtaining a list of STONITH Parameters	111
13.2. An IPMI-based STONITH Resource	112
13.3. Fencing topology with different devices for different nodes	114
14.1. A bare-bones status entry for a healthy node cl-virt-1	121
14.2. A set of transient node attributes for node cl-virt-1	122
14.3. A record of the apcstonith resource	123
14.4. A monitor operation (determines current state of the apcstonith resource)	124
14.5. Resource history of a pingd clone with multiple jobs	125
15.1. Constraint that fences node if ticketA is revoked	129
15.2. Constraint that demotes rsc1 if ticketA is revoked	129
15.3. Ticket constraint for multiple resources	129
D.1. Corosync 2.x configuration file for two nodes myhost1 and myhost2	141
D.2. Corosync 2.x configuration file for three nodes myhost1 , myhost2 and myhost3	142
D.3. Corosync 1.x configuration file for a cluster with all nodes on the 192.0.2.0/24 network	142
D.4. Corosync 1.x configuration fragment to enable Pacemaker plugin	143

D.5. Heartbeat configuration fragment to enable Pacemaker	144
H.1. An Empty Configuration	155
H.2. A simple configuration with two nodes, some cluster options and a resource	155
H.3. An advanced configuration with groups, clones and STONITH	156

Preface

Table of Contents

1. Document Conventions	xvii
1.1. Typographic Conventions	xvii
1.2. Pull-quote Conventions	xviii
1.3. Notes and Warnings	xix
2. We Need Feedback!	xix

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the [Liberation Fonts](https://fedorahosted.org/liberation-fonts/)¹ set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later include the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keys and key combinations. For example:

To see the contents of the file **my_next_bestselling_novel** in your current working directory, enter the **cat my_next_bestselling_novel** command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

Press **Enter** to execute the command.

Press **Ctrl+Alt+F2** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **mono-spaced bold**. For example:

¹ <https://fedorahosted.org/liberation-fonts/>

File-related classes include **filesystem** for file systems, **file** for files, and **dir** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, select the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find...** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh *john@example.com***.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount */home***.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — *username*, *domain.name*, *file-system*, *package*, *version* and *release*. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

books	Desktop	documentation	drafts	mss	photos	stuff	svn
books_tests	Desktop1	downloads	images	notes	scripts	svgs	

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
    public static void main(String args[])
        throws Exception
    {
        InitialContext iniCtx = new InitialContext();
        Object          ref    = iniCtx.lookup("EchoBean");
        EchoHome        home   = (EchoHome) ref;
        Echo            echo   = home.create();

        System.out.println("Created Echo");

        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
    }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.



Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla² against the product **Pacemaker**.

When submitting a bug report, be sure to mention the manual's identifier: *Pacemaker_Explained*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

² <http://bugs.clusterlabs.org>

Read-Me-First

Table of Contents

1.1. The Scope of this Document	1
1.2. What Is <i>Pacemaker</i> ?	1
1.3. Pacemaker Architecture	2
1.3.1. Internal Components	3
1.4. Types of Pacemaker Clusters	4

1.1. The Scope of this Document

The purpose of this document is to definitively explain the concepts used to configure Pacemaker. To achieve this, it will focus exclusively on the XML syntax used to configure the CIB.

For those that are allergic to XML, there exist several unified shells and GUIs for Pacemaker. However these tools will not be covered at all in this document ¹, precisely because they hide the XML.

Additionally, this document is NOT a step-by-step how-to guide for configuring a specific clustering scenario.

Although such guides exist, ² the purpose of this document is to provide an understanding of the building blocks that can be used to construct any type of Pacemaker cluster.

1.2. What Is *Pacemaker*?

Pacemaker is a *cluster resource manager*, that is, a logic responsible for a life-cycle of deployed software — indirectly perhaps even whole systems or their interconnections — under its control within a set of computers (a.k.a. *nodes*) and driven by prescribed rules.

It achieves maximum availability for your cluster services (a.k.a. *resources*) by detecting and recovering from node- and resource-level failures by making use of the messaging and membership capabilities provided by your preferred cluster infrastructure (either *Corosync*³ or *Heartbeat*⁴), and possibly by utilizing other parts of the overall cluster stack.



Note

For **the goal of minimal downtime** a term *high availability* was coined and together with its acronym, *HA*, is well-established in the sector. To differentiate this sort of clusters from high performance computing (*HPC*) ones, should a context require it (apparently, not the case in this document), using *HA cluster* is an option.

Pacemaker's key features include:

¹ I hope, however, that the concepts explained here make the functionality of these tools more easily understood.

² For example, see the *Clusters from Scratch* [<http://www.clusterlabs.org/doc/>] guide.

³ <http://www.corosync.org/>

⁴ <http://linux-ha.org/wiki/Heartbeat>

- Detection and recovery of node and service-level failures
- Storage agnostic, no requirement for shared storage
- Resource agnostic, anything that can be scripted can be clustered
- Supports *fencing* (also referred to as the *STONITH* acronym, *deciphered* later on) for ensuring data integrity
- Supports large and small clusters
- Supports both quorate and resource-driven clusters
- Supports practically any redundancy configuration
- Automatically replicated configuration that can be updated from any node
- Ability to specify cluster-wide service ordering, colocation and anti-colocation
- Support for advanced service types
 - Clones: for services which need to be active on multiple nodes
 - Multi-state: for services with multiple modes (e.g. master/slave, primary/secondary)
- Unified, scriptable cluster management tools

1.3. Pacemaker Architecture

At the highest level, the cluster is made up of three pieces:

- **Non-cluster-aware components.** These pieces include the resources themselves; scripts that start, stop and monitor them; and a local daemon that masks the differences between the different standards these scripts implement. Even though interactions of these resources when run as multiple instances can resemble a distributed system, they still lack the proper HA mechanisms and/or autonomous cluster-wide governance as subsumed in the following item.
- **Resource management.** Pacemaker provides the brain that processes and reacts to events regarding the cluster. These events include nodes joining or leaving the cluster; resource events caused by failures, maintenance and scheduled activities; and other administrative actions. Pacemaker will compute the ideal state of the cluster and plot a path to achieve it after any of these events. This may include moving resources, stopping nodes and even forcing them offline with remote power switches.
- **Low-level infrastructure.** Projects like *Corosync*, *CMAN* and *Heartbeat* provide reliable messaging, membership and quorum information about the cluster.

When combined with Corosync, Pacemaker also supports popular open source cluster filesystems.⁵

Due to past standardization within the cluster filesystem community, cluster filesystems make use of a common *distributed lock manager*, which makes use of Corosync for its messaging and membership capabilities (which nodes are up/down) and Pacemaker for fencing services.

⁵ Even though Pacemaker also supports Heartbeat, the filesystems need to use the stack for messaging and membership, and Corosync seems to be what they're standardizing on. Technically, it would be possible for them to support Heartbeat as well, but there seems little interest in this.

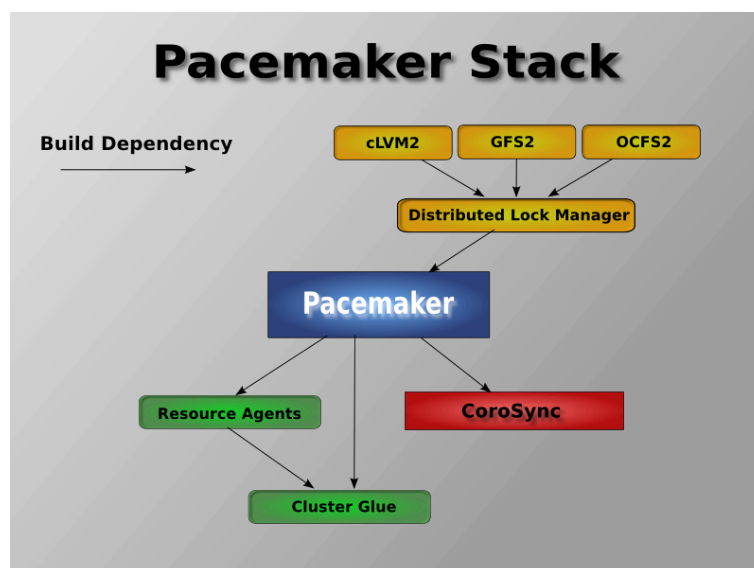


Figure 1.1. The Pacemaker Stack

1.3.1. Internal Components

Pacemaker itself is composed of five key components:

- *Cluster Information Base (CIB)*
- *Cluster Resource Management daemon (CRMd)*
- *Local Resource Management daemon (LRMd)*
- *Policy Engine (PEngine or PE)*
- *Fencing daemon (STONITHd)*

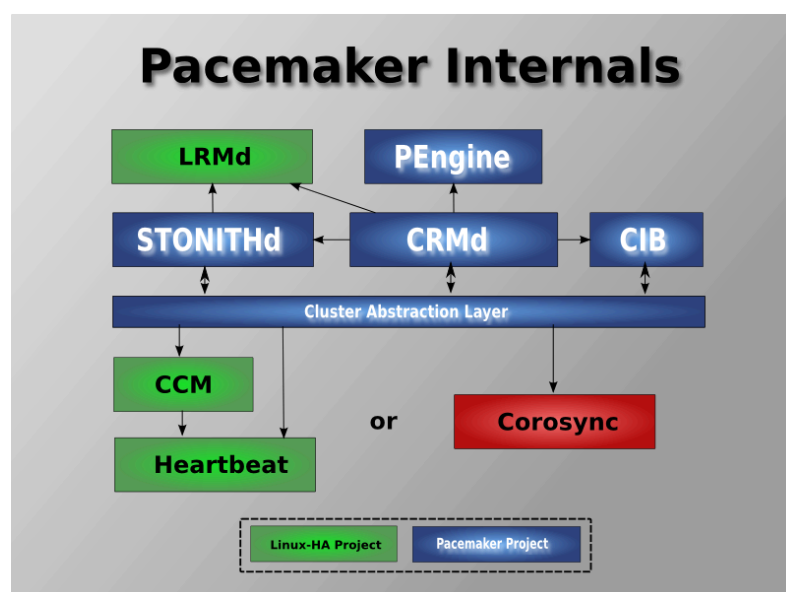


Figure 1.2. Internal Components

The CIB uses XML to represent both the cluster's configuration and current state of all resources in the cluster. The contents of the CIB are automatically kept in sync across the entire cluster and are used by the PEngine to compute the ideal state of the cluster and how it should be achieved.

This list of instructions is then fed to the *Designated Controller (DC)*. Pacemaker centralizes all cluster decision making by electing one of the CRMD instances to act as a master. Should the elected CRMD process (or the node it is on) fail, a new one is quickly established.

The DC carries out the PEngine's instructions in the required order by passing them to either the Local Resource Management daemon (LRMD) or CRMD peers on other nodes via the cluster messaging infrastructure (which in turn passes them on to their LRMD process).

The peer nodes all report the results of their operations back to the DC and, based on the expected and actual results, will either execute any actions that needed to wait for the previous one to complete, or abort processing and ask the PEngine to recalculate the ideal cluster state based on the unexpected results.

In some cases, it may be necessary to power off nodes in order to protect shared data or complete resource recovery. For this, Pacemaker comes with STONITHd.



Note

STONITH is an acronym for *Shoot-The-Other-Node-In-The-Head*, a recommended practice that misbehaving node is best to be promptly *fenced* (shut off, cut from shared resources or otherwise immobilized), and is usually implemented with a remote power switch.

In Pacemaker, STONITH devices are modeled as resources (and configured in the CIB) to enable them to be easily monitored for failure, however STONITHd takes care of understanding the STONITH topology such that its clients simply request a node be fenced, and it does the rest.

1.4. Types of Pacemaker Clusters

Pacemaker makes no assumptions about your environment. This allows it to support practically any *redundancy configuration*⁶ including *Active/Active*, *Active/Passive*, *N+1*, *N+M*, *N-to-1* and *N-to-N*.

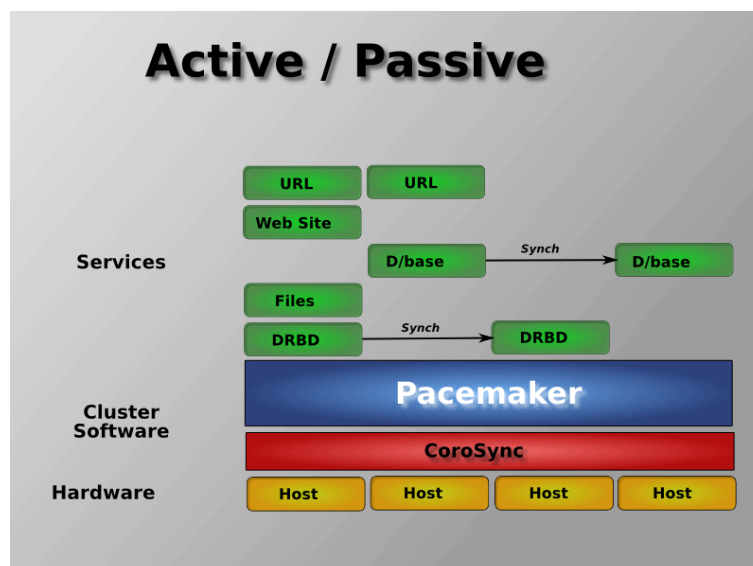


Figure 1.3. Active/Passive Redundancy

⁶ http://en.wikipedia.org/wiki/High-availability_cluster#Node_configurations

Two-node Active/Passive clusters using Pacemaker and *DRBD* are a cost-effective solution for many High Availability situations.

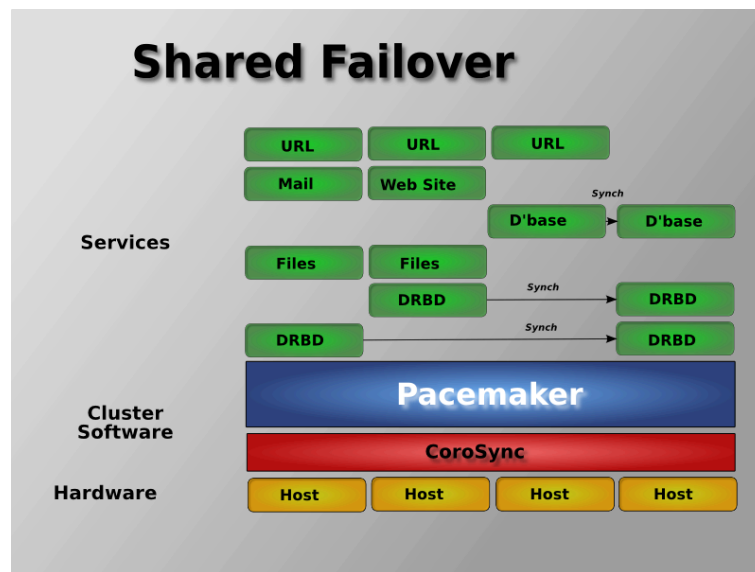


Figure 1.4. Shared Failover

By supporting many nodes, Pacemaker can dramatically reduce hardware costs by allowing several active/passive clusters to be combined and share a common backup node.

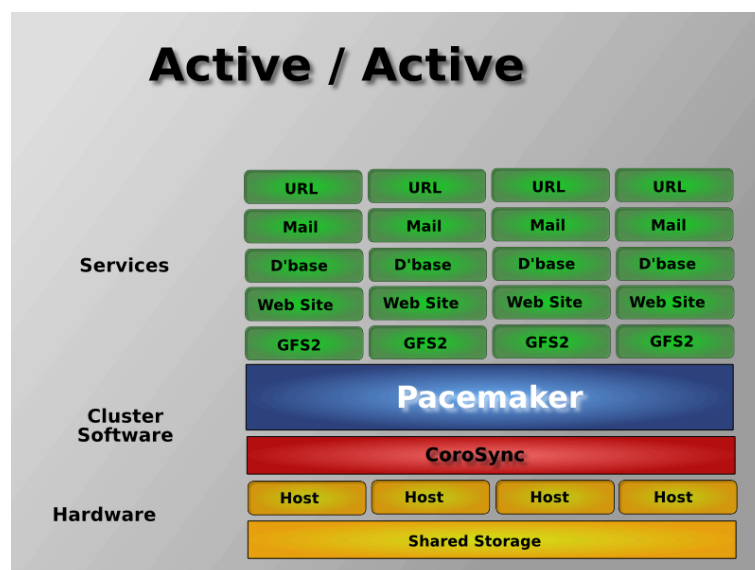


Figure 1.5. N to N Redundancy

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.

Configuration Basics

Table of Contents

2.1. Configuration Layout	7
2.2. The Current State of the Cluster	8
2.3. How Should the Configuration be Updated?	9
2.3.1. Editing the CIB Using XML	9
2.3.2. Quickly Deleting Part of the Configuration	10
2.3.3. Updating the Configuration Without Using XML	10
2.4. Making Configuration Changes in a Sandbox	11
2.5. Testing Your Configuration Changes	12
2.5.1. Small Cluster Transition	13
2.5.2. Complex Cluster Transition	14
2.6. Do I Need to Update the Configuration on All Cluster Nodes?	14

2.1. Configuration Layout

The cluster is defined by the Cluster Information Base (CIB), which uses XML notation. The simplest CIB, an empty one, looks like this:

Example 2.1. An empty configuration

```
<cib crm_feature_set="3.0.7" validate-with="pacemaker-1.2" admin_epoch="1" epoch="0"
num_updates="0">
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
```

The empty configuration above contains the major sections that make up a CIB:

- **cib**: The entire CIB is enclosed with a **cib** tag. Certain fundamental settings are defined as attributes of this tag.
- **configuration**: This section — the primary focus of this document — contains traditional configuration information such as what resources the cluster serves and the relationships among them.
 - **crm_config**: cluster-wide configuration options
 - **nodes**: the machines that host the cluster
 - **resources**: the services run by the cluster
 - **constraints**: indications of how resources should be placed
- **status**: This section contains the history of each resource on each node. Based on this data, the cluster can construct the complete current state of the cluster. The authoritative source for this section is the local resource manager (lrmd process) on each cluster node, and the cluster

will occasionally repopulate the entire section. For this reason, it is never written to disk, and administrators are advised against modifying it in any way.

In this document, configuration settings will be described as *properties* or *options* based on how they are defined in the CIB:

- Properties are XML attributes of an XML element.
- Options are name-value pairs expressed as **nvpair** child elements of an XML element.

Normally you will use command-line tools that abstract the XML, so the distinction will be unimportant; both properties and options are cluster settings you can tweak.

2.2. The Current State of the Cluster

Before one starts to configure a cluster, it is worth explaining how to view the finished product. For this purpose we have created the **crm_mon** utility, which will display the current state of an active cluster. It can show the cluster status by node or by resource and can be used in either single-shot or dynamically-updating mode. There are also modes for displaying a list of the operations performed (grouped by node and resource) as well as 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.

Details on all the available options can be obtained using the **crm_mon --help** command.

Example 2.2. Sample output from **crm_mon**

```
=====
Last updated: Fri Nov 23 15:26:13 2007
Current DC: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec)
3 Nodes configured.
5 Resources configured.
=====

Node: sles-1 (1186dc9a-324d-425a-966e-d757e693dc86): online
  192.168.100.181 (heartbeat::ocf:IPaddr): Started sles-1
  192.168.100.182 (heartbeat:IPaddr): Started sles-1
  192.168.100.183 (heartbeat::ocf:IPaddr): Started sles-1
  rsc_sles-1 (heartbeat::ocf:IPaddr): Started sles-1
  child_DoFencing:2 (stonith:external/vmware): Started sles-1
Node: sles-2 (02fb99a8-e30e-482f-b3ad-0fb3ce27d088): standby
Node: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec): online
  rsc_sles-2 (heartbeat::ocf:IPaddr): Started sles-3
  rsc_sles-3 (heartbeat::ocf:IPaddr): Started sles-3
  child_DoFencing:0 (stonith:external/vmware): Started sles-3
```

Example 2.3. Sample output from **crm_mon -n**

```
=====
Last updated: Fri Nov 23 15:26:13 2007
Current DC: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec)
3 Nodes configured.
5 Resources configured.
=====

Node: sles-1 (1186dc9a-324d-425a-966e-d757e693dc86): online
Node: sles-2 (02fb99a8-e30e-482f-b3ad-0fb3ce27d088): standby
Node: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec): online
```

```

Resource Group: group-1
  192.168.100.181 (heartbeat::ocf:IPAddr): Started sles-1
  192.168.100.182 (heartbeat:IPAddr): Started sles-1
  192.168.100.183 (heartbeat::ocf:IPAddr): Started sles-1
rsc_sles-1 (heartbeat::ocf:IPAddr): Started sles-1
rsc_sles-2 (heartbeat::ocf:IPAddr): Started sles-3
rsc_sles-3 (heartbeat::ocf:IPAddr): Started sles-3
Clone Set: DoFencing
  child_DoFencing:0 (stonith:external/vmware): Started sles-3
  child_DoFencing:1 (stonith:external/vmware): Stopped
  child_DoFencing:2 (stonith:external/vmware): Started sles-1

```

The DC (Designated Controller) node is where all the decisions are made, and if the current DC fails a new one is elected from the remaining cluster nodes. The choice of DC is of no significance to an administrator beyond the fact that its logs will generally be more interesting.

2.3. How Should the Configuration be Updated?

There are three basic rules for updating the cluster configuration:

- Rule 1 - Never edit the **cib.xml** file manually. Ever. I'm not making this up.
- Rule 2 - Read Rule 1 again.
- Rule 3 - The cluster will notice if you ignored rules 1 & 2 and refuse to use the configuration.

Now that it is clear how *not* to update the configuration, we can begin to explain how you *should*.

2.3.1. Editing the CIB Using XML

The most powerful tool for modifying the configuration is the **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.¹

Example 2.4. 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** or **/usr/lib/heartbeat** depending on your operating system 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.

¹ This process might appear to risk overwriting changes that happen after the initial **cibadmin** call, but **pacemaker** will reject any update that is "too old". If the CIB is updated in some other fashion after the initial **cibadmin**, the second **cibadmin** will be rejected because the version number will be too low.

Example 2.5. 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
```

2.3.2. Quickly Deleting 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:

Example 2.6. Searching for STONITH-related configuration items

```
# cibadmin -Q | 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">
<lrmd_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
<lrmd_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
<lrmd_resource id="child_DoFencing:1" type="external/vmware" class="stonith">
<lrmd_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
<lrmd_resource id="child_DoFencing:2" type="external/vmware" class="stonith">
<lrmd_resource id="child_DoFencing:0" type="external/vmware" class="stonith">
<lrmd_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"/>'
```

2.3.3. Updating the Configuration Without Using XML

Most tasks can be performed with one of the other command-line tools provided with pacemaker, avoiding the need to read or edit XML.

To enable STONITH for example, one could run:

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

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

```
# crm_standby --get-value --node somenode
```

Or, to find the current location of **my-test-rsc**, one can use:

```
# crm_resource --locate --resource my-test-rsc
```

Examples of using these tools for specific cases will be given throughout this document where appropriate.

**Note**

Old versions of pacemaker (1.0.3 and earlier) had different command-line tool syntax. If you are using an older version, check your installed manual pages for the proper syntax to use.

2.4. Making Configuration Changes in a Sandbox

Often it is desirable to preview the effects of a series of changes before updating the configuration atomically. For this purpose we have created **crm_shadow** which 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 the name of a configuration to create², and follow the simple on-screen instructions.

**Warning**

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

Example 2.7. 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.

Example 2.8. Using a sandbox to make multiple changes atomically, discard them and verify the real configuration is untouched

```
shadow[test] # crm_failcount -G -r rsc_c001n01
name=fail-count-rsc_c001n01 value=0
shadow[test] # crm_standby -v on -N c001n02
shadow[test] # crm_standby -G -N c001n02
name=c001n02 scope=nodes value=on
shadow[test] # cibadmin --erase --force
shadow[test] # cibadmin --query
```

² Shadow copies are identified with a name, making it possible to have more than one.

```
<cib cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0"
crm_feature_set="3.0" have-quorum="1" epoch="112"
  dc-uuid="c001n01" num_updates="1" cib-last-written="Fri Jun 27 12:17:10 2008">
  <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 cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0"
crm_feature_set="3.0" have-quorum="1" epoch="110"
  dc-uuid="c001n01" num_updates="551">
  <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"/>
      </cluster_property_set>
    </crm_config>
  </configuration>
</cib>
```

2.5. Testing Your Configuration Changes

We saw previously how to make a series of changes to a "shadow" copy of the configuration. Before loading the changes back into the cluster (e.g. **crm_shadow --commit mytest --force**), it is often advisable to simulate the effect of the changes with **crm_simulate**. For example:

```
# crm_simulate --live-check -VVVV --save-graph tmp.graph --save-dotfile tmp.dot
```

This tool uses the same library as the live cluster to show what it would have done given the supplied input. Its output, in addition to a significant amount of logging, is stored in two files **tmp.graph** and **tmp.dot**. Both files are representations of the same thing: the cluster's response to your changes.

The graph file stores the complete transition from the existing cluster state to your desired new state, containing a list of all the actions, their parameters and their pre-requisites. Because the transition graph is not terribly easy to read, the tool also generates a Graphviz³ dot-file representing the same information.

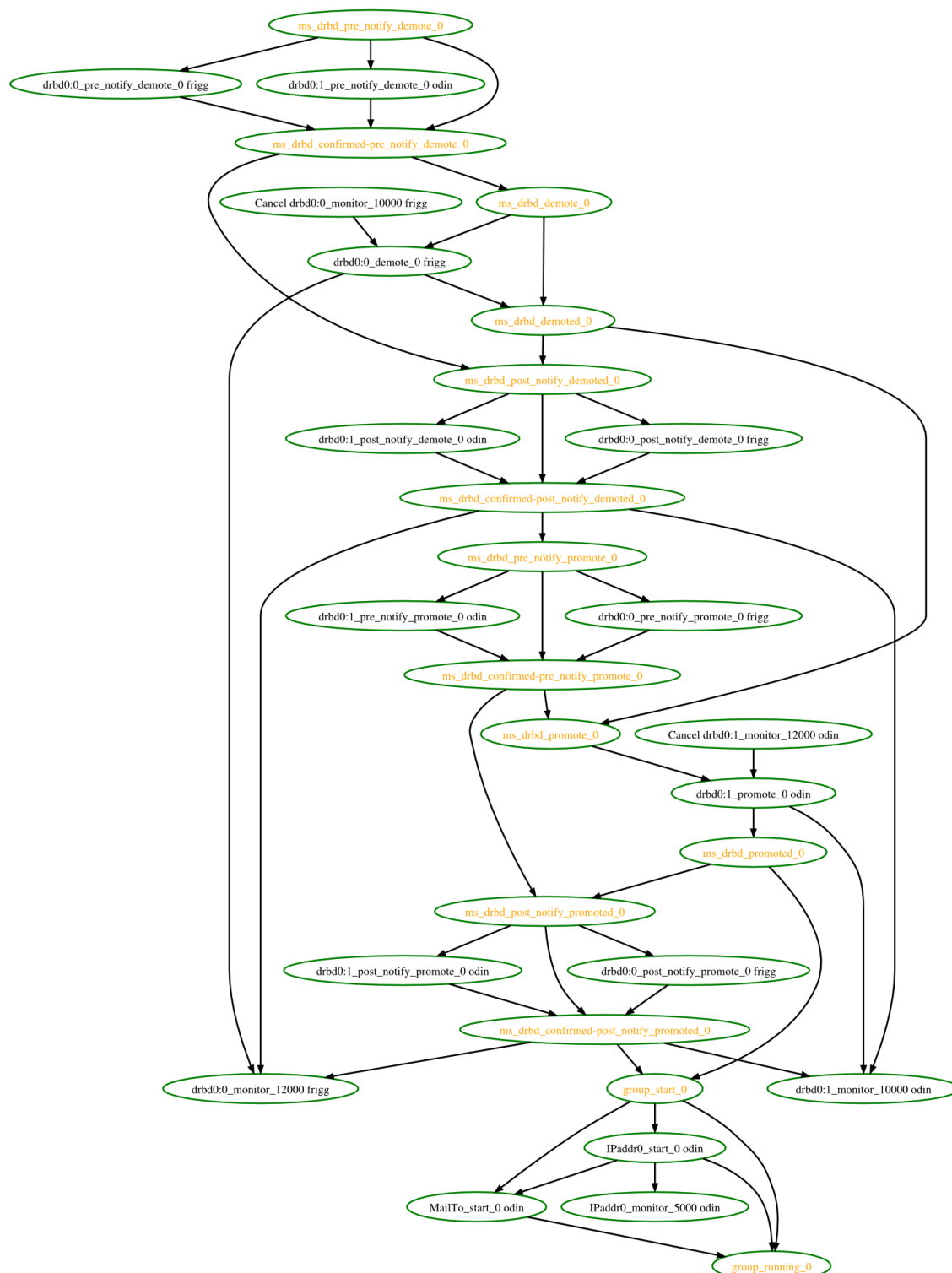
For information on the options supported by **crm_simulate**, use its **--help** option.

Interpreting the Graphviz output

- Arrows indicate ordering dependencies
- Dashed arrows indicate dependencies that are not present in the transition graph
- Actions with a dashed border of any color do not form part of the transition graph
- Actions with a green border form part of the transition graph
- Actions with a red border are ones the cluster would like to execute but cannot run
- Actions with a blue border are ones the cluster does not feel need to be executed

³ Graph visualization software. See <http://www.graphviz.org/> for details.

2.5.2. Complex Cluster Transition



2.6. Do I Need to Update the Configuration on All Cluster Nodes?

No. Any changes are immediately synchronized to the other active members of the cluster.

To reduce bandwidth, the cluster only broadcasts the incremental updates that result from your changes and uses MD5 checksums to ensure that each copy is completely consistent.

Cluster-Wide Configuration

Table of Contents

3.1. CIB Properties	17
3.1.1. Working with CIB Properties	18
3.2. Cluster Options	18
3.2.1. Querying and Setting Cluster Options	21
3.2.2. When Options are Listed More Than Once	21

3.1. CIB Properties

Certain settings are defined by CIB properties (that is, attributes of the **cib** tag) rather than with the rest of the cluster configuration in the **configuration** section.

The reason is simply a matter of parsing. These options are used by the configuration database which is, by design, mostly ignorant of the content it holds. So the decision was made to place them in an easy-to-find location.

Table 3.1. CIB Properties

Field	Description
admin_epoch	When a node joins the cluster, the cluster performs a check to see which node has the best configuration. It asks the node with the highest (admin_epoch , epoch , num_updates) tuple to replace the configuration on all the nodes — which makes setting them, and setting them correctly, very important. admin_epoch is never modified by the cluster; you can use this to make the configurations on any inactive nodes obsolete. <i>Never set this value to zero.</i> In such cases, the cluster cannot tell the difference between your configuration and the "empty" one used when nothing is found on disk.
epoch	The cluster increments this every time the configuration is updated (usually by the administrator).
num_updates	The cluster increments this every time the configuration or status is updated (usually by the cluster) and resets it to 0 when epoch changes.
validate-with	Determines the type of XML validation that will be done on the configuration. If set to none , the cluster will not verify that updates conform to the DTD (nor reject ones that don't). This option can be useful when operating a mixed-version cluster during an upgrade.
cib-last-written	Indicates when the configuration was last written to disk. Maintained by the cluster; for informational purposes only.
have-quorum	Indicates if the cluster has quorum. If false, this may mean that the cluster cannot start resources or fence other nodes (see no-quorum-policy below). Maintained by the cluster.
dc-uuid	Indicates which cluster node is the current leader. Used by the cluster when placing resources and determining the order of some events. Maintained by the cluster.

3.1.1. 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:

Example 3.1. Attributes set for a cib object

```
<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">
```

3.2. Cluster Options

Cluster options, as you might expect, control how the cluster behaves when confronted with certain situations.

They are grouped into sets within the **crm_config** section, and, in advanced configurations, there may be more than one set. (This will be described later in the section on [Chapter 8, Rules](#) where we will show how to have the cluster use different sets of options during working hours than during weekends.) For now, we will describe the simple case where each option is present at most once.

You can obtain an up-to-date list of cluster options, including their default values, by running the **man pengine** and **man crmd** commands.

Table 3.2. Cluster Options

Option	Default	Description
dc-version		Version of Pacemaker on the cluster's DC. Determined automatically by the cluster. Often includes the hash which identifies the exact Git changeset it was built from. Used for diagnostic purposes.
cluster-infrastructure		The messaging stack on which Pacemaker is currently running. Determined automatically by the cluster. Used for informational and diagnostic purposes.
expected-quorum-votes		The number of nodes expected to be in the cluster. Determined automatically by the cluster. Used to calculate quorum in clusters that use Corosync 1.x without CMAN as the messaging layer.
no-quorum-policy	stop	What to do when the cluster does not have quorum. Allowed values: <ul style="list-style-type: none"> • ignore: continue all resource management • freeze: continue resource management, but don't recover resources from nodes not in the affected partition • stop: stop all resources in the affected cluster partition

Option	Default	Description
		<ul style="list-style-type: none"> • suicide: fence all nodes in the affected cluster partition
batch-limit	30	The number of jobs that the Transition Engine (TE) is allowed to execute in parallel. The TE is the logic in pacemaker's CRMD that executes the actions determined by the Policy Engine (PE). The "correct" value will depend on the speed and load of your network and cluster nodes.
migration-limit	-1	The number of migration jobs that the TE is allowed to execute in parallel on a node. A value of -1 means unlimited.
symmetric-cluster	TRUE	Can all resources run on any node by default?
stop-all-resources	FALSE	Should the cluster stop all resources?
stop-orphan-resources	TRUE	Should deleted resources be stopped?
stop-orphan-actions	TRUE	Should deleted actions be cancelled?
start-failure-is-fatal	TRUE	Should a failure to start a resource on a particular node prevent further start attempts on that node? If FALSE, the cluster will decide whether to try starting on the same node again based on the resource's current failure count and migration-threshold (see Section 9.3.2, "Moving Resources Due to Failure").
enable-startup-probes	TRUE	Should the cluster check for active resources during startup?
maintenance-mode	FALSE	Should the cluster refrain from monitoring, starting and stopping resources?
stonith-enabled	TRUE	<p>Should failed nodes and nodes with resources that can't be stopped be shot? If you value your data, set up a STONITH device and enable this.</p> <p>If true, or unset, the cluster will refuse to start resources unless one or more STONITH resources have been configured. If false, unresponsive nodes are immediately assumed to be running no resources, and resource takeover to online nodes starts without any further protection (which means <i>data loss</i> if the unresponsive node still accesses shared storage, for example). See also the requires meta-attribute in Section 5.4, "Resource Options".</p>
stonith-action	reboot	Action to send to STONITH device. Allowed values are reboot and off . The value poweroff is also allowed, but is only used for legacy devices.
stonith-timeout	60s	How long to wait for STONITH actions (reboot, on, off) to complete
concurrent-fencing	FALSE	Is the cluster allowed to initiate multiple fence actions concurrently?

Option	Default	Description
cluster-delay	60s	Estimated maximum round-trip delay over the network (excluding action execution). If the TE requires an action to be executed on another node, it will consider the action failed if it does not get a response from the other node in this time (after considering the action's own timeout). The "correct" value will depend on the speed and load of your network and cluster nodes.
dc-deadtime	20s	How long to wait for a response from other nodes during startup. The "correct" value will depend on the speed/load of your network and the type of switches used.
cluster-recheck-interval	15min	Polling interval for time-based changes to options, resource parameters and constraints. The Cluster is primarily event-driven, but your configuration can have elements that take effect based on the time of day. To ensure these changes take effect, we can optionally poll the cluster's status for changes. A value of 0 disables polling. Positive values are an interval (in seconds unless other SI units are specified, e.g. 5min).
pe-error-series-max	-1	The number of PE inputs resulting in ERRORS to save. Used when reporting problems. A value of -1 means unlimited (report all).
pe-warn-series-max	-1	The number of PE inputs resulting in WARNINGS to save. Used when reporting problems. A value of -1 means unlimited (report all).
pe-input-series-max	-1	The number of "normal" PE inputs to save. Used when reporting problems. A value of -1 means unlimited (report all).
remove-after-stop	FALSE	<i>Advanced Use Only:</i> Should the cluster remove resources from the LRM after they are stopped? Values other than the default are, at best, poorly tested and potentially dangerous.
startup-fencing	TRUE	<i>Advanced Use Only:</i> Should the cluster shoot unseen nodes? Not using the default is very unsafe!
election-timeout	2min	<i>Advanced Use Only:</i> If you need to adjust this value, it probably indicates the presence of a bug.
shutdown-escalation	20min	<i>Advanced Use Only:</i> If you need to adjust this value, it probably indicates the presence of a bug.
crmd-integration-timeout	3min	<i>Advanced Use Only:</i> If you need to adjust this value, it probably indicates the presence of a bug.
crmd-finalization-timeout	30min	<i>Advanced Use Only:</i> If you need to adjust this value, it probably indicates the presence of a bug.
crmd-transition-delay	0s	<i>Advanced Use Only:</i> Delay cluster recovery for the configured interval to allow for additional/related events to occur. Useful if your configuration is sensitive to the

Option	Default	Description
		order in which ping updates arrive. Enabling this option will slow down cluster recovery under all conditions.
default-resource-stickiness	0	<i>Deprecated:</i> See Section 5.4.2, “Setting Global Defaults for Resource Meta-Attributes” instead
is-managed-default	TRUE	<i>Deprecated:</i> See Section 5.4.2, “Setting Global Defaults for Resource Meta-Attributes” instead
default-action-timeout	20s	<i>Deprecated:</i> See Section 5.5.3, “Setting Global Defaults for Operations” instead

3.2.1. 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
```

3.2.2. 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.

Example 3.2. Deleting an option that is listed twice

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

```
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)  
Please choose from one of the matches above and supply the 'id' with --id
```

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 [Chapter 8, Rules](#).

Cluster Nodes

Table of Contents

4.1. Defining a Cluster Node	23
4.2. Where Pacemaker Gets the Node Name	23
4.3. Node Attributes	24
4.4. Managing Nodes in a Corosync-Based Cluster	24
4.4.1. Adding a New Corosync Node	24
4.4.2. Removing a Corosync Node	25
4.4.3. Replacing a Corosync Node	25
4.5. Managing Nodes in a Heartbeat-based Cluster	25
4.5.1. Adding a New Heartbeat Node	25
4.5.2. Removing a Heartbeat Node	26
4.5.3. Replacing a Heartbeat Node	26

4.1. Defining a Cluster Node

Each node in the cluster will have an entry in the nodes section containing its UUID, uname, and type.

Example 4.1. Example Heartbeat cluster node entry

```
<node id="1186dc9a-324d-425a-966e-d757e693dc86" uname="pcmk-1" type="normal"/>
```

Example 4.2. Example Corosync cluster node entry

```
<node id="101" uname="pcmk-1" type="normal"/>
```

In normal circumstances, the admin should let the cluster populate this information automatically from the communications and membership data. However for Heartbeat, one can use the **crm_uuid** tool to read an existing UUID or define a value before the cluster starts.

4.2. Where Pacemaker Gets the Node Name

Traditionally, Pacemaker required nodes to be referred to by the value returned by **uname -n**. This can be problematic for services that require the **uname -n** to be a specific value (e.g. for a licence file).

This requirement has been relaxed for clusters using Corosync 2.0 or later. The name Pacemaker uses is:

1. The value stored in **corosync.conf** under **ring0_addr** in the **nodelist**, if it does not contain an IP address; otherwise
2. The value stored in **corosync.conf** under **name** in the **nodelist**; otherwise
3. The value of **uname -n**

Pacemaker provides the **crm_node -n** command which displays the name used by a running cluster.

If a Corosync **nodelist** is used, **crm_node --name-for-id number** is also available to display the name used by the node with the corosync **nodeid** of *number*, for example: **crm_node --name-for-id 2**.

4.3. Node Attributes

Node attributes are a special type of option (name-value pair) that applies to a node object.

Beyond the basic definition of a node, the administrator can describe the node's attributes, such as how much RAM, disk, what OS or kernel version it has, perhaps even its physical location. This information can then be used by the cluster when deciding where to place resources. For more information on the use of node attributes, see [Chapter 8, Rules](#).

Node attributes can be specified ahead of time or populated later, when the cluster is running, using **crm_attribute**.

Below is what the node's definition would look like if the admin ran the command:

Example 4.3. Result of using **crm_attribute** to specify which kernel pcmk-1 is running

```
# crm_attribute --type nodes --node pcmk-1 --name kernel --update $(uname -r)

<node uname="pcmk-1" type="normal" id="101">
  <instance_attributes id="nodes-101">
    <nvpair id="nodes-101-kernel" name="kernel" value="3.10.0-123.13.2.el7.x86_64"/>
  </instance_attributes>
</node>
```

Rather than having to read the XML, a simpler way to determine the current value of an attribute is to use **crm_attribute** again:

```
# crm_attribute --type nodes --node pcmk-1 --name kernel --query
scope=nodes name=kernel value=3.10.0-123.13.2.el7.x86_64
```

By specifying **--type nodes** the admin tells the cluster that this attribute is persistent. There are also transient attributes which are kept in the status section which are "forgotten" whenever the node rejoins the cluster. The cluster uses this area to store a record of how many times a resource has failed on that node, but administrators can also read and write to this section by specifying **--type status**.

4.4. Managing Nodes in a Corosync-Based Cluster

4.4.1. Adding a New Corosync Node

To add a new node:

1. Install Corosync and Pacemaker on the new host.
2. Copy **/etc/corosync/corosync.conf** and **/etc/corosync/authkey** (if it exists) from an existing node. You may need to modify the **mcastaddr** option to match the new node's IP address.

3. Start the cluster software on the new host. If a log message containing "Invalid digest" appears from Corosync, the keys are not consistent between the machines.

4.4.2. Removing a Corosync Node

Because the messaging and membership layers are the authoritative source for cluster nodes, deleting them from the CIB is not a complete solution. First, one must arrange for corosync to forget about the node (**pcmk-1** in the example below).

1. Stop the cluster on the host to be removed. How to do this will vary with your operating system and installed versions of cluster software, for example, **pcs cluster stop** if you are using pcs for cluster management, or **service corosync stop** on a host using corosync 1.x with the pacemaker plugin.
2. From one of the remaining active cluster nodes, tell Pacemaker to forget about the removed host, which will also delete the node from the CIB:

```
# crm_node -R pcmk-1
```



Note

This procedure only works for pacemaker 1.1.8 and later.

4.4.3. Replacing a Corosync Node

To replace an existing cluster node:

1. Make sure the old node is completely stopped.
2. Give the new machine the same hostname and IP address as the old one.
3. Follow the procedure above for adding a node.

4.5. Managing Nodes in a Heartbeat-based Cluster

4.5.1. Adding a New Heartbeat Node

To add a new node:

1. Install heartbeat and pacemaker on the new host.
2. Copy **ha.cf** and **authkeys** from an existing node.
3. If you do not use **autojoin any** in **ha.cf**, run:

```
hb_addnode $(uname -n)
```

4. Start the cluster software on the new node.

4.5.2. Removing a Heartbeat Node

Because the messaging and membership layers are the authoritative source for cluster nodes, deleting them from the CIB is not a complete solution. First, one must arrange for Heartbeat to forget about the node (pcmk-1 in the example below).

1. On the host to be removed, stop the cluster:

```
service heartbeat stop
```

2. From one of the remaining active cluster nodes, tell Heartbeat the node should be removed:

```
hb_delnode pcmk-1
```

3. Tell Pacemaker to forget about the removed host:

```
crm_node -R pcmk-1
```



Note

This procedure only works for pacemaker versions after 1.1.8.

4.5.3. Replacing a Heartbeat Node

To replace an existing cluster node:

1. Make sure the old node is completely stopped.
2. Give the new machine the same hostname as the old one.
3. Go to an active cluster node and look up the UUID for the old node in **/var/lib/heartbeat/hostcache**.
4. Install the cluster software.
5. Copy **ha.cf** and **authkeys** to the new node.
6. On the new node, populate its UUID using **crm_uuid -w** and the UUID obtained earlier.
7. Start the new cluster node.

Cluster Resources

Table of Contents

5.1. What is a Cluster Resource?	27
5.2. Resource Classes	27
5.2.1. Open Cluster Framework	28
5.2.2. Linux Standard Base	28
5.2.3. Systemd	29
5.2.4. Upstart	29
5.2.5. System Services	30
5.2.6. STONITH	30
5.2.7. Nagios Plugins	30
5.3. Resource Properties	31
5.4. Resource Options	31
5.4.1. Resource Meta-Attributes	31
5.4.2. Setting Global Defaults for Resource Meta-Attributes	34
5.4.3. Resource Instance Attributes	34
5.5. Resource Operations	36
5.5.1. Monitoring Resources for Failure	37
5.5.2. Monitoring Resources When Administration is Disabled	37
5.5.3. Setting Global Defaults for Operations	38
5.5.4. When Implicit Operations Take a Long Time	38
5.5.5. Multiple Monitor Operations	38
5.5.6. Disabling a Monitor Operation	39

5.1. What is a Cluster Resource?

A resource is a service made highly available by a cluster. The simplest type of resource, a *primitive* resource, is described in this section. More complex forms, such as groups and clones, are described in later sections.

Every primitive resource has a *resource agent*. A resource agent is an external program that abstracts the service it provides and present a consistent view to the cluster.

This allows the cluster to be agnostic about the resources it manages. The cluster doesn't need to understand how the resource works because it relies on the resource agent to do the right thing when given a **start**, **stop** or **monitor** command. For this reason, it is crucial that resource agents are well-tested.

Typically, resource agents come in the form of shell scripts. However, they can be written using any technology (such as C, Python or Perl) that the author is comfortable with.

5.2. Resource Classes

Pacemaker supports several classes of agents:

- OCF

- LSB
- Upstart
- Systemd
- Service
- Fencing
- Nagios Plugins

5.2.1. Open Cluster Framework

The OCF standard ¹ is basically an extension of the Linux Standard Base conventions for init scripts to:

- support parameters,
- make them self-describing, and
- make them extensible

OCF specs have strict definitions of the exit codes that actions must return. ²

The cluster follows these specifications exactly, and giving the wrong exit code will cause the cluster to behave in ways you will likely find puzzling and annoying. In particular, the cluster needs to distinguish a completely stopped resource from one which is in some erroneous and indeterminate state.

Parameters are passed to the resource agent as environment variables, with the special prefix **OCF_RESKEY_**. So, a parameter which the user thinks of as **ip** will be passed to the resource agent as **OCF_RESKEY_ip**. The number and purpose of the parameters is left to the resource agent; however, the resource agent should use the **meta-data** command to advertise any that it supports.

The OCF class is the most preferred as it is an industry standard, highly flexible (allowing parameters to be passed to agents in a non-positional manner) and self-describing.

For more information, see the [reference](#) ³ and [Appendix B, More About OCF Resource Agents](#).

5.2.2. Linux Standard Base

LSB resource agents are those found in **/etc/init.d**.

Generally, they are provided by the OS distribution and, in order to be used with the cluster, they must conform to the LSB Spec. ⁴

¹ See <http://www.opencf.org/cgi-bin/viewcvs.cgi/specs/ra/resource-agent-api.txt?rev=HEAD> — at least as it relates to resource agents. The Pacemaker implementation has been somewhat extended from the OCF specs, but none of those changes are incompatible with the original OCF specification.

² The resource-agents source code includes the **ocf-tester** script, which can be useful in this regard.

³ http://www.linux-ha.org/wiki/OCF_Resource_Agents

⁴ See http://refspecs.linux-foundation.org/LSB_3.0.0/LSB-Core-generic/LSB-Core-generic/inisrptact.html for the LSB Spec as it relates to init scripts.



Warning

Many distributions claim LSB compliance but ship with broken init scripts. For details on how to check whether your init script is LSB-compatible, see [Appendix G, Init Script LSB Compliance](#). Common problematic violations of the LSB standard include:

- Not implementing the status operation at all
- Not observing the correct exit status codes for **start/stop/status** actions
- Starting a started resource returns an error
- Stopping a stopped resource returns an error



Important

Remember to make sure the computer is *not* configured to start any services at boot time — that should be controlled by the cluster.

5.2.3. Systemd

Some newer distributions have replaced the old "[SysV](#)"⁵ style of initialization daemons and scripts with an alternative called [Systemd](#)⁶.

Pacemaker is able to manage these services *if they are present*.

Instead of init scripts, systemd has *unit files*. Generally, the services (unit files) are provided by the OS distribution, but there are online guides for converting from init scripts.⁷



Important

Remember to make sure the computer is *not* configured to start any services at boot time — that should be controlled by the cluster.

5.2.4. Upstart

⁵ <http://en.wikipedia.org/wiki/Init#SysV-style>

⁶ <http://www.freedesktop.org/wiki/Software/systemd>

⁷ For example, <http://0pointer.de/blog/projects/systemd-for-admins-3.html>

Some newer distributions have replaced the old "SysV"⁸ style of initialization daemons (and scripts) with an alternative called *Upstart*⁹.

Pacemaker is able to manage these services *if they are present*.

Instead of init scripts, upstart has *jobs*. Generally, the services (jobs) are provided by the OS distribution.



Important

Remember to make sure the computer is *not* configured to start any services at boot time — that should be controlled by the cluster.

5.2.5. System Services

Since there are various types of system services (**systemd**, **upstart**, and **lsb**), Pacemaker supports a special **service** alias which intelligently figures out which one applies to a given cluster node.

This is particularly useful when the cluster contains a mix of **systemd**, **upstart**, and **lsb**.

In order, Pacemaker will try to find the named service as:

1. an LSB init script
2. a Systemd unit file
3. an Upstart job

5.2.6. STONITH

The STONITH class is used exclusively for fencing-related resources. This is discussed later in [Chapter 13, STONITH](#).

5.2.7. Nagios Plugins

Nagios Plugins¹⁰ allow us to monitor services on remote hosts.

Pacemaker is able to do remote monitoring with the plugins *if they are present*.

A common use case is to configure them as resources belonging to a resource container (usually a virtual machine), and the container will be restarted if any of them has failed. Another use is to configure them as ordinary resources to be used for monitoring hosts or services via the network.

⁸ <http://en.wikipedia.org/wiki/Init#SysV-style>

⁹ <http://upstart.ubuntu.com/>

¹⁰ The project has two independent forks, hosted at <https://www.nagios-plugins.org/> and <https://www.monitoring-plugins.org/>. Output from both projects' plugins is similar, so plugins from either project can be used with pacemaker.

The supported parameters are same as the long options of the plugin.

5.3. Resource Properties

These values tell the cluster which resource agent to use for the resource, where to find that resource agent and what standards it conforms to.

Table 5.1. Properties of a Primitive Resource

Field	Description
id	Your name for the resource
class	The standard the resource agent conforms to. Allowed values: lsb , nagios , ocf , service , stonith , systemd , upstart
type	The name of the Resource Agent you wish to use. E.g. IPaddr or Filesystem
provider	The OCF spec allows multiple vendors to supply the same resource agent. To use the OCF resource agents supplied by the Heartbeat project, you would specify heartbeat here.

The XML definition of a resource can be queried with the **crm_resource** tool. For example:

```
# crm_resource --resource Email --query-xml
```

might produce:

Example 5.1. A system resource definition

```
<primitive id="Email" class="service" type="exim"/>
```



Note

One of the main drawbacks to system services (LSB, systemd or Upstart) resources is that they do not allow any parameters!

Example 5.2. An OCF resource definition

```
<primitive id="Public-IP" class="ocf" type="IPaddr" provider="heartbeat">
  <instance_attributes id="Public-IP-params">
    <nvpair id="Public-IP-ip" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

5.4. Resource Options

Resources have two types of options: *meta-attributes* and *instance attributes*. Meta-attributes apply to any type of resource, while instance attributes are specific to each resource agent.

5.4.1. Resource Meta-Attributes

Meta-attributes are used by the cluster to decide how a resource should behave and can be easily set using the **--meta** option of the **crm_resource** command.

Table 5.2. Meta-attributes of a Primitive Resource

Field	Default	Description
priority	0	If not all resources can be active, the cluster will stop lower priority resources in order to keep higher priority ones active.
target-role	Started	<p>What state should the cluster attempt to keep this resource in? Allowed values:</p> <ul style="list-style-type: none"> • Stopped: Force the resource to be stopped • Started: Allow the resource to be started (and in the case of <i>multi-state</i> resources, promoted to master if appropriate) • Slave: Allow the resource to be started, but only in Slave mode if the resource is <i>multi-state</i> • Master: Equivalent to Started
is-managed	TRUE	Is the cluster allowed to start and stop the resource? Allowed values: true , false
resource-stickiness	value of resource-stickiness in the rsc_defaults section	How much does the resource prefer to stay where it is?
requires	fencing (unless stonith-enabled is false or class is stonith , in which case it defaults to quorum)	<p>Conditions under which the resource can be started (<i>since 1.1.8</i>) Allowed values:</p> <ul style="list-style-type: none"> • nothing: can always be started • quorum: The cluster can only start this resource if a majority of the configured nodes are active • fencing: The cluster can only start this resource if a majority of the configured nodes are active <i>and</i> any failed or unknown nodes have been powered off • unfencing: The cluster can only start this resource if a majority of the configured nodes are active <i>and</i> any failed or unknown nodes have been powered off <i>and</i> only on nodes that have been <i>unfenced</i> (<i>since 1.1.9</i>)
migration-threshold	INFINITY	How many failures may occur for this resource on a node, before this node is marked ineligible to host this resource. A value of 0 indicates that this feature is disabled (the node will never be marked ineligible); by contrast, the cluster treats INFINITY (the default) as a very large but finite number. This option has an effect only if the failed operation has

Field	Default	Description
		on-fail=restart (the default), and additionally for failed start operations, if the cluster property start-failure-is-fatal is false.
failure-timeout	0	How many seconds to wait before acting as if the failure had not occurred, and potentially allowing the resource back to the node on which it failed. A value of 0 indicates that this feature is disabled. As with any time-based actions, this is not guaranteed to be checked more frequently than the value of cluster-recheck-interval (see Section 3.2, "Cluster Options").
multiple-active	stop_start	What should the cluster do if it ever finds the resource active on more than one node? Allowed values: <ul style="list-style-type: none"> • block: mark the resource as unmanaged • stop_only: stop all active instances and leave them that way • stop_start: stop all active instances and start the resource in one location only
remote-node		The name of the remote-node this resource defines. This both enables the resource as a remote-node and defines the unique name used to identify the remote-node. If no other parameters are set, this value will also be assumed as the hostname to connect to at the port specified by remote-port . WARNING : This value cannot overlap with any resource or node IDs. If not specified, this feature is disabled.
remote-port	3121	Port to use for the guest connection to pacemaker_remote
remote-addr	value of remote-node	The IP address or hostname to connect to if remote-node's name is not the hostname of the guest.
remote-connect-timeout	60s	How long before a pending guest connection will time out.



Note

Support for remote nodes was added in pacemaker 1.1.10. If you are using an earlier version, options related to remote nodes will not be available.

As an example of setting resource options, if you performed the following commands on an LSB Email resource:

```
# crm_resource --meta --resource Email --set-parameter priority --parameter-value 100
# crm_resource -m -r Email -p multiple-active -v block
```

the resulting resource definition might be:

Example 5.3. An LSB resource with cluster options

```
<primitive id="Email" class="lsb" type="exim">
  <meta_attributes id="Email-meta_attributes">
    <nvpair id="Email-meta_attributes-priority" name="priority" value="100"/>
    <nvpair id="Email-meta_attributes-multiple-active" name="multiple-active"
value="block"/>
  </meta_attributes>
</primitive>
```

5.4.2. Setting Global Defaults for Resource Meta-Attributes

To set a default value for a resource option, add it to the **rsc_defaults** section with **crm_attribute**. For example,

```
# crm_attribute --type rsc_defaults --name is-managed --update false
```

would prevent the cluster from starting or stopping any of the resources in the configuration (unless of course the individual resources were specifically enabled by having their **is-managed** set to **true**).

5.4.3. Resource Instance Attributes

The resource agents of some resource classes (lsb, systemd and upstart *not* among them) can be given parameters which determine how they behave and which instance of a service they control.

If your resource agent supports parameters, you can add them with the **crm_resource** command. For example,

```
# crm_resource --resource Public-IP --set-parameter ip --parameter-value 192.0.2.2
```

would create an entry in the resource like this:

Example 5.4. An example OCF resource with instance attributes

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

For an OCF resource, the result would be an environment variable called **OCF_RESKEY_ip** with a value of **192.0.2.2**.

The list of instance attributes supported by an OCF resource agent can be found by calling the resource agent with the **meta-data** command. The output contains an XML description of all the supported attributes, their purpose and default values.

Example 5.5. Displaying the metadata for the Dummy resource agent template

```
# export OCF_ROOT=/usr/lib/ocf
# $OCF_ROOT/resource.d/pacemaker/Dummy meta-data
```

```
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="Dummy" version="1.0">
<version>1.0</version>

<longdesc>
This is a Dummy Resource Agent. It does absolutely nothing except
keep track of whether its running or not.
Its purpose in life is for testing and to serve as a template for RA writers.

NB: Please pay attention to the timeouts specified in the actions
section below. They should be meaningful for the kind of resource
the agent manages. They should be the minimum advised timeouts,
but they shouldn't/cannot cover _all_ possible resource
instances. So, try to be neither overly generous nor too stingy,
but moderate. The minimum timeouts should never be below 10 seconds.
</longdesc>
<shortdesc>Example stateless resource agent</shortdesc>

<parameters>
<parameter name="state" unique="1">
<longdesc>
Location to store the resource state in.
</longdesc>
<shortdesc>State file</shortdesc>
<content type="string" default="/var/run/Dummy-default.state" />
</parameter>

<parameter name="fake" unique="0">
<longdesc>
Fake attribute that can be changed to cause a reload
</longdesc>
<shortdesc>Fake attribute that can be changed to cause a reload</shortdesc>
<content type="string" default="dummy" />
</parameter>

<parameter name="op_sleep" unique="1">
<longdesc>
Number of seconds to sleep during operations. This can be used to test how
the cluster reacts to operation timeouts.
</longdesc>
<shortdesc>Operation sleep duration in seconds.</shortdesc>
<content type="string" default="0" />
</parameter>

</parameters>

<actions>
<action name="start"          timeout="20" />
<action name="stop"           timeout="20" />
<action name="monitor"        timeout="20" interval="10" depth="0"/>
<action name="reload"         timeout="20" />
<action name="migrate_to"     timeout="20" />
<action name="migrate_from"   timeout="20" />
<action name="validate-all"   timeout="20" />
<action name="meta-data"      timeout="5" />
</actions>
</resource-agent>
```

5.5. Resource Operations

Operations are actions the cluster can perform on a resource by calling the resource agent. Resource agents must support certain common operations such as start, stop and monitor, and may implement any others.

Some operations are generated by the cluster itself, for example, stopping and starting resources as needed.

You can configure operations in the cluster configuration. As an example, by default the cluster will *not* ensure your resources stay healthy once they are started.¹¹ To instruct the cluster to do this, you need to add a **monitor** operation to the resource's definition.

Example 5.6. An OCF resource with a recurring health check

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <operations>
    <op id="public-ip-check" name="monitor" interval="60s"/>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

Table 5.3. Properties of an Operation

Field	Default	Description
id		A unique name for the operation.
name		The action to perform. This can be any action supported by the agent; common values include monitor , start , and stop .
interval	0	How frequently (in seconds) to perform the operation. A value of 0 means never. A positive value defines a <i>recurring action</i> , which is typically used with <i>monitor</i> .
timeout		How long to wait before declaring the action has failed
on-fail	restart (except for stop operations, which default to fence when STONITH is enabled and block otherwise)	<p>The action to take if this action ever fails. Allowed values:</p> <ul style="list-style-type: none"> • ignore: Pretend the resource did not fail. • block: Don't perform any further operations on the resource. • stop: Stop the resource and do not start it elsewhere. • restart: Stop the resource and start it again (possibly on a different node).

¹¹ Currently, anyway. Automatic monitoring operations may be added in a future version of Pacemaker.

Field	Default	Description
		<ul style="list-style-type: none"> • fence: STONITH the node on which the resource failed. • standby: Move <i>all</i> resources away from the node on which the resource failed.
enabled	TRUE	If false , ignore this operation definition. This is typically used to pause a particular recurring monitor operation; for instance, it can complement the respective resource being unmanaged (is-managed=false), as this alone will <i>not block any configured monitoring</i> . Disabling the operation does not suppress all actions of the given type. Allowed values: true , false .
record-pending		If true , the intention to perform the operation is recorded so that GUIs and CLI tools can indicate that an operation is in progress. This is best set as an <i>operation default</i> (see next section). Allowed values: true , false .
role		Run the operation only on node(s) that the cluster thinks should be in the specified role. This only makes sense for recurring monitor operations. Allowed (case-sensitive) values: Stopped , Started , and in the case of <i>multi-state</i> resources, Slave and Master .

5.5.1. Monitoring Resources for Failure

When Pacemaker first starts a resource, it runs one-time monitor operations (referred to as *probes*) to ensure the resource is running where it's supposed to be, and not running where it's not supposed to be. (This behavior can be affected by the **resource-discovery** location constraint property.)

Other than those initial probes, Pacemaker will not (by default) check that the resource continues to stay healthy. As in the example above, you must configure monitor operations explicitly to perform these checks.

By default, a monitor operation will ensure that the resource is running where it is supposed to. The **target-role** property can be used for further checking.

For example, if a resource has one monitor operation with **interval=10** **role=Started** and a second monitor operation with **interval=11** **role=Stopped**, the cluster will run the first monitor on any nodes it thinks *should* be running the resource, and the second monitor on any nodes that it thinks *should not* be running the resource (for the truly paranoid, who want to know when an administrator manually starts a service by mistake).

5.5.2. Monitoring Resources When Administration is Disabled

Recurring monitor operations behave differently under various administrative settings:

- When a resource is unmanaged (by setting **is-managed=false**): No monitors will be stopped.

If the unmanaged resource is stopped on a node where the cluster thinks it should be running, the cluster will detect and report that it is not, but it will not consider the monitor failed, and will not try to start the resource until it is managed again.

Starting the unmanaged resource on a different node is strongly discouraged and will at least cause the cluster to consider the resource failed, and may require the resource's **target-role** to be set to **Stopped** then **Started** to be recovered.

- When a node is put into standby: All resources will be moved away from the node, and all monitor operations will be stopped on the node, except those with **role=Stopped**. Monitor operations with **role=Stopped** will be started on the node if appropriate.
- When the cluster is put into maintenance mode: All resources will be marked as unmanaged. All monitor operations will be stopped, except those with **role=Stopped**. As with single unmanaged resources, starting a resource on a node other than where the cluster expects it to be will cause problems.

5.5.3. Setting Global Defaults for Operations

You can change the global default values for operation properties in a given cluster. These are defined in an **op_defaults** section of the CIB's **configuration** section, and can be set with **crm_attribute**. For example,

```
# crm_attribute --type op_defaults --name timeout --update 20s
```

would default each operation's **timeout** to 20 seconds. If an operation's definition also includes a value for **timeout**, then that value would be used for that operation instead.

5.5.4. When Implicit Operations Take a Long Time

The cluster will always perform a number of implicit operations: **start**, **stop** and a non-recurring **monitor** operation used at startup to check whether the resource is already active. If one of these is taking too long, then you can create an entry for them and specify a longer timeout.

Example 5.7. An OCF resource with custom timeouts for its implicit actions

```
<primitive id="Public-IP" class="ocf" type="IPaddr" provider="heartbeat">
  <operations>
    <op id="public-ip-startup" name="monitor" interval="0" timeout="90s"/>
    <op id="public-ip-start" name="start" interval="0" timeout="180s"/>
    <op id="public-ip-stop" name="stop" interval="0" timeout="15min"/>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

5.5.5. Multiple Monitor Operations

Provided no two operations (for a single resource) have the same name and interval, you can have as many monitor operations as you like. In this way, you can do a superficial health check every minute and progressively more intense ones at higher intervals.

To tell the resource agent what kind of check to perform, you need to provide each monitor with a different value for a common parameter. The OCF standard creates a special parameter called

OCF_CHECK_LEVEL for this purpose and dictates that it is "made available to the resource agent without the normal **OCF_RESKEY** prefix".

Whatever name you choose, you can specify it by adding an **instance_attributes** block to the **op** tag. It is up to each resource agent to look for the parameter and decide how to use it.

Example 5.8. An OCF resource with two recurring health checks, performing different levels of checks specified via **OCF_CHECK_LEVEL**.

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <operations>
    <op id="public-ip-health-60" name="monitor" interval="60">
      <instance_attributes id="params-public-ip-depth-60">
        <nvpair id="public-ip-depth-60" name="OCF_CHECK_LEVEL" value="10"/>
      </instance_attributes>
    </op>
    <op id="public-ip-health-300" name="monitor" interval="300">
      <instance_attributes id="params-public-ip-depth-300">
        <nvpair id="public-ip-depth-300" name="OCF_CHECK_LEVEL" value="20"/>
      </instance_attributes>
    </op>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-level" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

5.5.6. Disabling a Monitor Operation

The easiest way to stop a recurring monitor is to just delete it. However, there can be times when you only want to disable it temporarily. In such cases, simply add **enabled="false"** to the operation's definition.

Example 5.9. Example of an OCF resource with a disabled health check

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <operations>
    <op id="public-ip-check" name="monitor" interval="60s" enabled="false"/>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
  </instance_attributes>
</primitive>
```

This can be achieved from the command line by executing:

```
# cibadmin --modify --xml-text '<op id="public-ip-check" enabled="false"/>'
```

Once you've done whatever you needed to do, you can then re-enable it with

```
# cibadmin --modify --xml-text '<op id="public-ip-check" enabled="true"/>'
```


Resource Constraints

Table of Contents

6.1. Scores	41
6.1.1. Infinity Math	41
6.2. Deciding Which Nodes a Resource Can Run On	42
6.2.1. Location Properties	42
6.2.2. Asymmetrical "Opt-In" Clusters	43
6.2.3. Symmetrical "Opt-Out" Clusters	44
6.2.4. What if Two Nodes Have the Same Score	44
6.3. Specifying the Order in which Resources Should Start/Stop	44
6.3.1. Ordering Properties	45
6.3.2. Optional and mandatory ordering	45
6.4. Placing Resources Relative to other Resources	46
6.4.1. Colocation Properties	46
6.4.2. Mandatory Placement	47
6.4.3. Advisory Placement	47
6.5. Resource Sets	48
6.6. Ordering Sets of Resources	48
6.6.1. Ordered Set	49
6.6.2. Ordering Multiple Sets	49
6.6.3. Resource Set OR Logic	51
6.7. Colocating Sets of Resources	51

6.1. Scores

Scores of all kinds are integral to how the cluster works. Practically everything from moving a resource to deciding which resource to stop in a degraded cluster is achieved by manipulating scores in some way.

Scores are calculated per resource and node. Any node with a negative score for a resource can't run that resource. The cluster places a resource on the node with the highest score for it.

6.1.1. Infinity Math

Pacemaker implements **INFINITY** (or equivalently, **+INFINITY**) internally as a score of 1,000,000. Addition and subtraction with it follow these three basic rules:

- Any value + **INFINITY** = **INFINITY**
- Any value - **INFINITY** = **-INFINITY**
- **INFINITY** - **INFINITY** = **-INFINITY**

**Note**

What if you want to use a score higher than 1,000,000? Typically this possibility arises when someone wants to base the score on some external metric that might go above 1,000,000.

The short answer is you can't.

The long answer is it is sometimes possible work around this limitation creatively. You may be able to set the score to some computed value based on the external metric rather than use the metric directly. For nodes, you can store the metric as a node attribute, and query the attribute when computing the score (possibly as part of a custom resource agent).

6.2. Deciding Which Nodes a Resource Can Run On

Location constraints tell the cluster which nodes a resource can run on.

There are two alternative strategies. One way is to say that, by default, resources can run anywhere, and then the location constraints specify nodes that are not allowed (an *opt-out* cluster). The other way is to start with nothing able to run anywhere, and use location constraints to selectively enable allowed nodes (an *opt-in* cluster).

Whether you should choose opt-in or opt-out depends on your personal preference and the make-up of your cluster. If most of your resources can run on most of the nodes, then an opt-out arrangement is likely to result in a simpler configuration. On the other-hand, if most resources can only run on a small subset of nodes, an opt-in configuration might be simpler.

6.2.1. Location Properties

Table 6.1. Properties of a `rsc_location` Constraint

Field	Default	Description
id		A unique name for the constraint
rsc		A resource name
node		A node's name
score		Positive values indicate the resource should run on this node. Negative values indicate the resource should not run on this node. Values of +/- INFINITY change "should"/"should not" to "must"/"must not".
resource-discovery	always	Whether Pacemaker should perform resource discovery (that is, check whether the resource is already running) for this resource on this node. This should normally be left as the default, so that rogue instances of a service can be stopped when they are running where they are not supposed to be. However, there are two situations where disabling resource discovery is a good idea: when a service is not installed on a node, discovery might return an error (properly written OCF agents will not, so this is usually only seen with other agent types); and when Pacemaker Remote is used to scale a cluster to hundreds

Field	Default	Description
		<p>of nodes, limiting resource discovery to allowed nodes can significantly boost performance. (<i>since 1.1.13</i>)</p> <ul style="list-style-type: none"> • always: Always perform resource discovery for the specified resource on this node. • never: Never perform resource discovery for the specified resource on this node. This option should generally be used with a <code>-INFINITY</code> score, although that is not strictly required. • exclusive: Perform resource discovery for the specified resource only on this node (and other nodes similarly marked as exclusive). Multiple location constraints using exclusive discovery for the same resource across different nodes creates a subset of nodes resource-discovery is exclusive to. If a resource is marked for exclusive discovery on one or more nodes, that resource is only allowed to be placed within that subset of nodes.



Warning

Setting resource-discovery to **never** or **exclusive** removes Pacemaker's ability to detect and stop unwanted instances of a service running where it's not supposed to be. It is up to the system administrator (you!) to make sure that the service can *never* be active on nodes without resource-discovery (such as by leaving the relevant software uninstalled).

6.2.2. Asymmetrical "Opt-In" Clusters

To create an opt-in cluster, start by preventing resources from running anywhere by default:

```
# crm_attribute --name symmetric-cluster --update false
```

Then start enabling nodes. The following fragment says that the web server prefers **sles-1**, the database prefers **sles-2** and both can fail over to **sles-3** if their most preferred node fails.

Example 6.1. Opt-in location constraints for two resources

```
<constraints>
  <rsc_location id="loc-1" rsc="Webserver" node="sles-1" score="200"/>
  <rsc_location id="loc-2" rsc="Webserver" node="sles-3" score="0"/>
  <rsc_location id="loc-3" rsc="Database" node="sles-2" score="200"/>
  <rsc_location id="loc-4" rsc="Database" node="sles-3" score="0"/>
</constraints>
```

6.2.3. Symmetrical "Opt-Out" Clusters

To create an opt-out cluster, start by allowing resources to run anywhere by default:

```
# crm_attribute --name symmetric-cluster --update true
```

Then start disabling nodes. The following fragment is the equivalent of the above opt-in configuration.

Example 6.2. Opt-out location constraints for two resources

```
<constraints>
  <rsc_location id="loc-1" rsc="Webserver" node="sles-1" score="200"/>
  <rsc_location id="loc-2-dont-run" rsc="Webserver" node="sles-2" score="-INFINITY"/>
  <rsc_location id="loc-3-dont-run" rsc="Database" node="sles-1" score="-INFINITY"/>
  <rsc_location id="loc-4" rsc="Database" node="sles-2" score="200"/>
</constraints>
```

6.2.4. What if Two Nodes Have the Same Score

If two nodes have the same score, then the cluster will choose one. This choice may seem random and may not be what was intended, however the cluster was not given enough information to know any better.

Example 6.3. Constraints where a resource prefers two nodes equally

```
<constraints>
  <rsc_location id="loc-1" rsc="Webserver" node="sles-1" score="INFINITY"/>
  <rsc_location id="loc-2" rsc="Webserver" node="sles-2" score="INFINITY"/>
  <rsc_location id="loc-3" rsc="Database" node="sles-1" score="500"/>
  <rsc_location id="loc-4" rsc="Database" node="sles-2" score="300"/>
  <rsc_location id="loc-5" rsc="Database" node="sles-2" score="200"/>
</constraints>
```

In the example above, assuming no other constraints and an inactive cluster, **Webserver** would probably be placed on **sles-1** and **Database** on **sles-2**. It would likely have placed **Webserver** based on the node's uname and **Database** based on the desire to spread the resource load evenly across the cluster. However other factors can also be involved in more complex configurations.

6.3. Specifying the Order in which Resources Should Start/Stop

Ordering constraints tell the cluster the order in which resources should start.



Important

Ordering constraints affect *only* the ordering of resources; they do *not* require that the resources be placed on the same node. If you want resources to be started on the same node *and* in a specific order, you need both an ordering constraint *and* a colocation constraint (see [Section 6.4, “Placing Resources Relative to other Resources”](#)), or alternatively, a group (see [Section 10.1, “Groups - A Syntactic Shortcut”](#)).

6.3.1. Ordering Properties

Table 6.2. Properties of a `rsc_order` Constraint

Field	Default	Description
id		A unique name for the constraint
first		Name of the resource that the then resource depends on
then		Name of the dependent resource
first-action	start	The action that the first resource must complete before then-action can be initiated for the then resource. Allowed values: start , stop , promote , demote .
then-action	value of first-action	The action that the then resource can execute only after the first-action on the first resource has completed. Allowed values: start , stop , promote , demote .
kind		How to enforce the constraint. Allowed values: <ul style="list-style-type: none"> • Optional: Just a suggestion. Only applies if both resources are executing the specified actions. Any change in state by the first resource will have no effect on the then resource. • Mandatory: Always. If first does not perform first-action, then will not be allowed to performed then-action. If first is restarted, then (if running) will be stopped beforehand and started afterward. • Serialize: Ensure that no two stop/start actions occur concurrently for the resources. First and then can start in either order, but one must complete starting before the other can be started. A typical use case is when resource start-up puts a high load on the host.
symmetrical	TRUE	If true, the reverse of the constraint applies for the opposite action (for example, if B starts after A starts, then B stops before A stops).

Promote and **demote** apply to the master role of [multi-state](#) resources.

6.3.2. Optional and mandatory ordering

Here is an example of ordering constraints where **Database** *must* start before **Webserver**, and **IP** *should* start before **Webserver** if they both need to be started:

Example 6.4. Optional and mandatory ordering constraints

```
<constraints>
<rsc_order id="order-1" first="IP" then="Webserver" kind="Optional"/>
<rsc_order id="order-2" first="Database" then="Webserver" kind="Mandatory" />
</constraints>
```

Because the above example lets **symmetrical** default to TRUE, **Webserver** must be stopped before **Database** can be stopped, and **Webserver** should be stopped before **IP** if they both need to be stopped.

6.4. Placing Resources Relative to other Resources

Colocation constraints tell the cluster that the location of one resource depends on the location of another one.

Colocation has an important side-effect: it affects the order in which resources are assigned to a node. Think about it: You can't place A relative to B unless you know where B is.¹

So when you are creating colocation constraints, it is important to consider whether you should colocate A with B, or B with A.

Another thing to keep in mind is that, assuming A is colocated with B, the cluster will take into account A's preferences when deciding which node to choose for B.

For a detailed look at exactly how this occurs, see [Colocation Explained](#)².



Important

Colocation constraints affect *only* the placement of resources; they do *not* require that the resources be started in a particular order. If you want resources to be started on the same node *and* in a specific order, you need both an ordering constraint (see [Section 6.3, "Specifying the Order in which Resources Should Start/Stop"](#)) and a colocation constraint, or alternatively, a group (see [Section 10.1, "Groups - A Syntactic Shortcut"](#)).

6.4.1. Colocation Properties

Table 6.3. Properties of a `rsc_colocation` Constraint

Field	Description
id	A unique name for the constraint.
rsc	The name of a resource that should be located relative to with-rsc .

¹ While the human brain is sophisticated enough to read the constraint in any order and choose the correct one depending on the situation, the cluster is not quite so smart. Yet.

² http://clusterlabs.org/doc/Colocation_Explained.pdf

Field	Description
with-rsc	The name of the resource used as the colocation target. The cluster will decide where to put this resource first and then decide where to put rsc .
score	Positive values indicate the resources should run on the same node. Negative values indicate the resources should run on different nodes. Values of +/- INFINITY change "should" to "must".

6.4.2. Mandatory Placement

Mandatory placement occurs when the constraint's score is **+INFINITY** or **-INFINITY**. In such cases, if the constraint can't be satisfied, then the **rsc** resource is not permitted to run. For **score=INFINITY**, this includes cases where the **with-rsc** resource is not active.

If you need resource **A** to always run on the same machine as resource **B**, you would add the following constraint:

Example 6.5. Mandatory colocation constraint for two resources

```
<rsc_colocation id="colocate" rsc="A" with-rsc="B" score="INFINITY"/>
```

Remember, because **INFINITY** was used, if **B** can't run on any of the cluster nodes (for whatever reason) then **A** will not be allowed to run. Whether **A** is running or not has no effect on **B**.

Alternatively, you may want the opposite — that **A** *cannot* run on the same machine as **B**. In this case, use **score=" -INFINITY"**.

Example 6.6. Mandatory anti-colocation constraint for two resources

```
<rsc_colocation id="anti-colocate" rsc="A" with-rsc="B" score="- INFINITY"/>
```

Again, by specifying **-INFINITY**, the constraint is binding. So if the only place left to run is where **B** already is, then **A** may not run anywhere.

As with **INFINITY**, **B** can run even if **A** is stopped. However, in this case **A** also can run if **B** is stopped, because it still meets the constraint of **A** and **B** not running on the same node.

6.4.3. Advisory Placement

If mandatory placement is about "must" and "must not", then advisory placement is the "I'd prefer if" alternative. For constraints with scores greater than **-INFINITY** and less than **INFINITY**, the cluster will try to accommodate your wishes but may ignore them if the alternative is to stop some of the cluster resources.

As in life, where if enough people prefer something it effectively becomes mandatory, advisory colocation constraints can combine with other elements of the configuration to behave as if they were mandatory.

Example 6.7. Advisory colocation constraint for two resources

```
<rsc_colocation id="colocate-maybe" rsc="A" with-rsc="B" score="500"/>
```

6.5. Resource Sets

Resource sets allow multiple resources to be affected by a single constraint.

Example 6.8. A set of 3 resources

```
<resource_set id="resource-set-example">
  <resource_ref id="A"/>
  <resource_ref id="B"/>
  <resource_ref id="C"/>
</resource_set>
```

Resource sets are valid inside **rsc_location**, **rsc_order** (see [Section 6.6, “Ordering Sets of Resources”](#)), **rsc_colocation** (see [Section 6.7, “Colocating Sets of Resources”](#)), and **rsc_ticket** (see [Section 15.3, “Configuring Ticket Dependencies”](#)) constraints.

A resource set has a number of properties that can be set, though not all have an effect in all contexts.

Table 6.4. Properties of a resource_set

Field	Default	Description
id		A unique name for the set
sequential	true	Whether the members of the set must be acted on in order. Meaningful within rsc_order and rsc_colocation .
require-all	true	Whether all members of the set must be active before continuing. Meaningful within rsc_order . (<i>since 1.1.13</i>)
role		Limit the effect of the constraint to the specified role. Meaningful within rsc_location , rsc_colocation and rsc_ticket .
action		Limit the effect of the constraint to the specified action. Meaningful within rsc_order .
score		<i>Advanced use only.</i> Use a specific score for this set within the constraint.

6.6. Ordering Sets of Resources

A common situation is for an administrator to create a chain of ordered resources, such as:

Example 6.9. A chain of ordered resources

```
<constraints>
  <rsc_order id="order-1" first="A" then="B" />
  <rsc_order id="order-2" first="B" then="C" />
  <rsc_order id="order-3" first="C" then="D" />
</constraints>
```



Figure 6.1. Visual representation of the four resources' start order for the above constraints

6.6.1. Ordered Set

To simplify this situation, resource sets (see [Section 6.5, “Resource Sets”](#)) can be used within ordering constraints:

Example 6.10. A chain of ordered resources expressed as a set

```
<constraints>
  <rsc_order id="order-1">
    <resource_set id="ordered-set-example" sequential="true">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
  </rsc_order>
</constraints>
```

While the set-based format is not less verbose, it is significantly easier to get right and maintain.



Important

If you use a higher-level tool, pay attention to how it exposes this functionality. Depending on the tool, creating a set **A B** may be equivalent to **A then B**, or **B then A**.

6.6.2. Ordering Multiple Sets

The syntax can be expanded to allow ordered sets of (un)ordered resources. In the example below, **A** and **B** can both start in parallel, as can **C** and **D**, however **C** and **D** can only start once *both A and B* are active.

Example 6.11. Ordered sets of unordered resources

```
<constraints>
  <rsc_order id="order-1">
    <resource_set id="ordered-set-1" sequential="false">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="ordered-set-2" sequential="false">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
  </rsc_order>
</constraints>
```

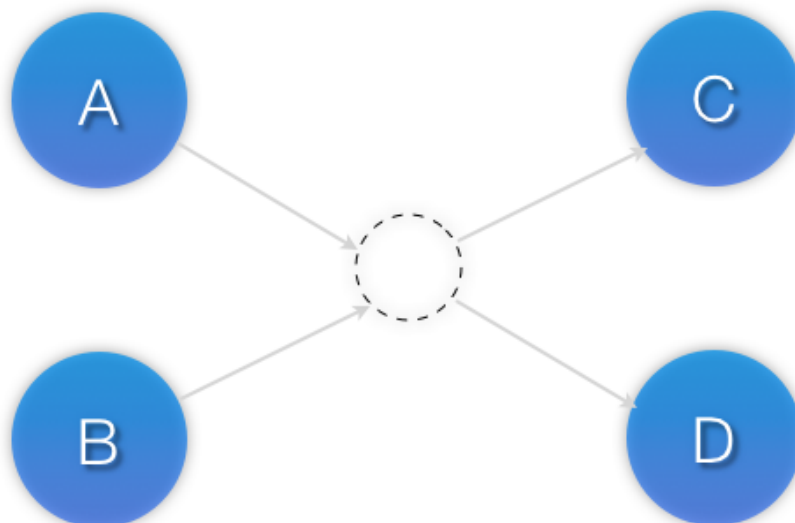


Figure 6.2. Visual representation of the start order for two ordered sets of unordered resources

Of course either set — or both sets — of resources can also be internally ordered (by setting **sequential="true"**) and there is no limit to the number of sets that can be specified.

Example 6.12. Advanced use of set ordering - Three ordered sets, two of which are internally unordered

```
<constraints>
  <rsc_order id="order-1">
    <resource_set id="ordered-set-1" sequential="false">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="ordered-set-2" sequential="true">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
    <resource_set id="ordered-set-3" sequential="false">
      <resource_ref id="E"/>
      <resource_ref id="F"/>
    </resource_set>
  </rsc_order>
</constraints>
```

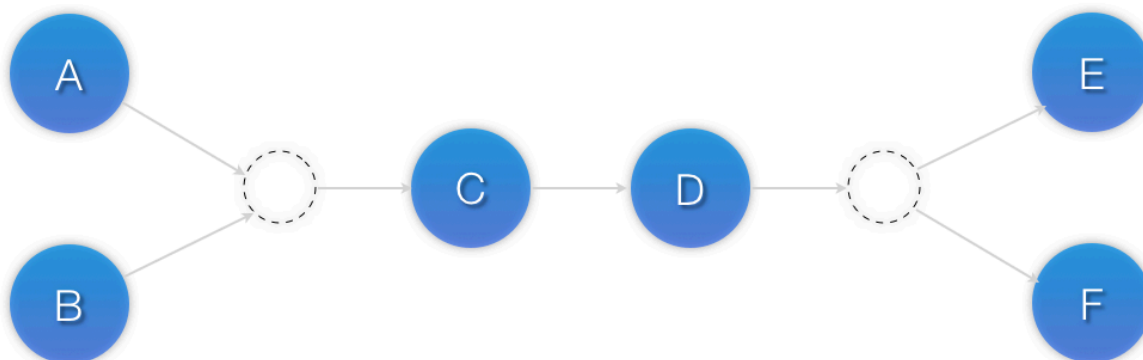


Figure 6.3. Visual representation of the start order for the three sets defined above

**Important**

An ordered set with **sequential=false** makes sense only if there is another set in the constraint. Otherwise, the constraint has no effect.

6.6.3. Resource Set OR Logic

The unordered set logic discussed so far has all been "AND" logic. To illustrate this take the 3 resource set figure in the previous section. Those sets can be expressed, **(A and B) then (C) then (D) then (E and F)**.

Say for example we want to change the first set, **(A and B)**, to use "OR" logic so the sets look like this: **(A or B) then (C) then (D) then (E and F)**. This functionality can be achieved through the use of the **require-all** option. This option defaults to TRUE which is why the "AND" logic is used by default. Setting **require-all=false** means only one resource in the set needs to be started before continuing on to the next set.

Example 6.13. Resource Set "OR" logic: Three ordered sets, where the first set is internally unordered with "OR" logic

```
<constraints>
  <rsc_order id="order-1">
    <resource_set id="ordered-set-1" sequential="false" require-all="false">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="ordered-set-2" sequential="true">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
    <resource_set id="ordered-set-3" sequential="false">
      <resource_ref id="E"/>
      <resource_ref id="F"/>
    </resource_set>
  </rsc_order>
</constraints>
```

**Important**

An ordered set with **require-all=false** makes sense only in conjunction with **sequential=false**. Think of it like this: **sequential=false** modifies the set to be an unordered set using "AND" logic by default, and adding **require-all=false** flips the unordered set's "AND" logic to "OR" logic.

6.7. Colocating Sets of Resources

Another common situation is for an administrator to create a set of colocated resources.

One way to do this would be to define a resource group (see [Section 10.1, "Groups - A Syntactic Shortcut"](#)), but that cannot always accurately express the desired state.

Another way would be to define each relationship as an individual constraint, but that causes a constraint explosion as the number of resources and combinations grow. An example of this approach:

Example 6.14. Chain of colocated resources

```
<constraints>
  <rsc_colocation id="coloc-1" rsc="D" with-rsc="C" score="INFINITY"/>
  <rsc_colocation id="coloc-2" rsc="C" with-rsc="B" score="INFINITY"/>
  <rsc_colocation id="coloc-3" rsc="B" with-rsc="A" score="INFINITY"/>
</constraints>
```

To make things easier, resource sets (see [Section 6.5, “Resource Sets”](#)) can be used within colocation constraints. As with the chained version, a resource that can't be active prevents any resource that must be colocated with it from being active. For example, if **B** is not able to run, then both **C** and by inference **D** must also remain stopped. Here is an example `resource_set`:

Example 6.15. Equivalent colocation chain expressed using `resource_set`

```
<constraints>
  <rsc_colocation id="coloc-1" score="INFINITY" >
    <resource_set id="colocated-set-example" sequential="true">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
  </rsc_colocation>
</constraints>
```



Important

If you use a higher-level tool, pay attention to how it exposes this functionality. Depending on the tool, creating a set **A B** may be equivalent to **A with B**, or **B with A**.

This notation can also be used to tell the cluster that a set of resources must all be located with a common peer, but have no dependencies on each other. In this scenario, unlike the previous, **B** *would* be allowed to remain active even if **A** or **C** (or both) were inactive.

Example 6.16. Using colocated sets to specify a common peer

```
<constraints>
  <rsc_colocation id="coloc-1" score="INFINITY" >
    <resource_set id="colocated-set-1" sequential="false">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
      <resource_ref id="C"/>
    </resource_set>
    <resource_set id="colocated-set-2" sequential="true">
      <resource_ref id="D"/>
    </resource_set>
  </rsc_colocation>
</constraints>
```




Important

A colocated set with **sequential=false** makes sense only if there is another set in the constraint. Otherwise, the constraint has no effect.

There is no inherent limit to the number and size of the sets used. The only thing that matters is that in order for any member of one set in the constraint to be active, all members of sets listed after it must also be active (and naturally on the same node); and if a set has **sequential=true**, then in order for one member of that set to be active, all members listed after it must also be active. You can even specify the role in which the members of a set must be in using the set's **role** attribute.

Example 6.17. A colocation chain where the members of the middle set have no interdependencies and the last has master status.

```
<constraints>
  <rsc_colocation id="coloc-1" score="INFINITY" >
    <resource_set id="colocated-set-1" sequential="true">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="colocated-set-2" sequential="false">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
      <resource_ref id="E"/>
    </resource_set>
    <resource_set id="colocated-set-3" sequential="true" role="Master">
      <resource_ref id="F"/>
      <resource_ref id="G"/>
    </resource_set>
  </rsc_colocation>
</constraints>
```

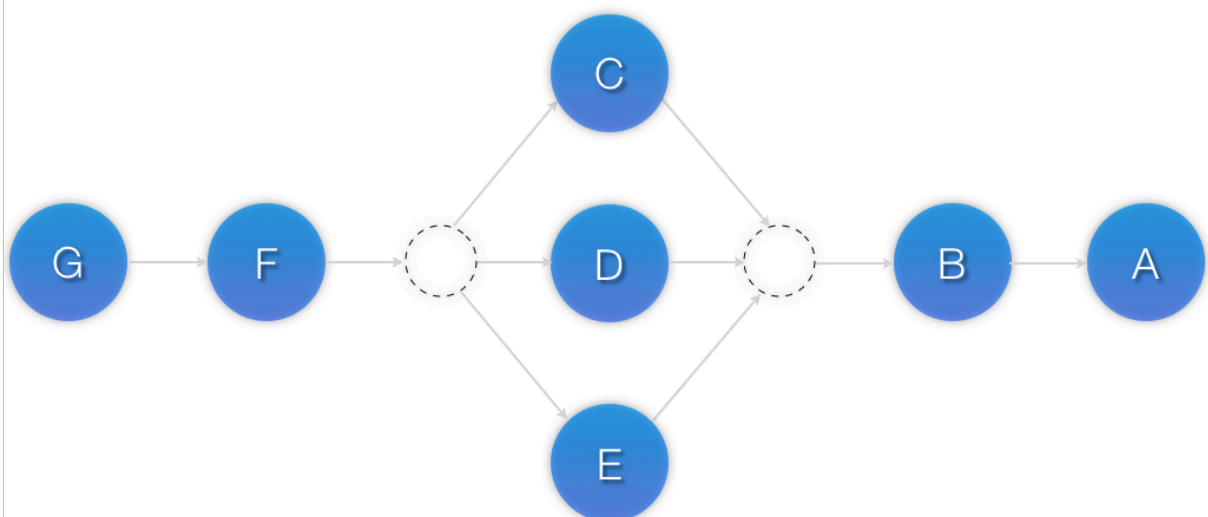


Figure 6.4. Visual representation of a colocation chain where the members of the middle set have no inter-dependencies



Note

Unlike ordered sets, colocated sets do not use the **require-all** option.

Alerts

Table of Contents

7.1. Alert Agents	55
7.2. Alert Recipients	55
7.3. Alert Meta-Attributes	56
7.4. Alert Instance Attributes	56
7.5. Using the Sample Alert Agents	57
7.6. Writing an Alert Agent	58

Alerts may be configured to take some external action when a cluster event occurs (node failure, resource starting or stopping, etc.).

7.1. Alert Agents

As with resource agents, the cluster calls an external program (an *alert agent*) to handle alerts. The cluster passes information about the event to the agent via environment variables. Agents can do anything desired with this information (send an e-mail, log to a file, update a monitoring system, etc.).

Example 7.1. Simple alert configuration

```
<configuration>
  <alerts>
    <alert id="my-alert" path="/path/to/my-script.sh" />
  </alerts>
</configuration>
```

In the example above, the cluster will call **my-script.sh** for each event.

Multiple alert agents may be configured; the cluster will call all of them for each event.

Alert agents will be called only on cluster nodes. They will be called for events involving Pacemaker Remote nodes, but they will never be called *on* those nodes.

7.2. Alert Recipients

Usually alerts are directed towards a recipient. Thus each alert may be additionally configured with one or more recipients. The cluster will call the agent separately for each recipient.

Example 7.2. Alert configuration with recipient

```
<configuration>
  <alerts>
    <alert id="my-alert" path="/path/to/my-script.sh">
      <recipient id="my-alert-recipient" value="some-address"/>
    </alert>
  </alerts>
</configuration>
```

In the above example, the cluster will call **my-script.sh** for each event, passing the recipient **some-address** as an environment variable.

The recipient may be anything the alert agent can recognize — an IP address, an e-mail address, a file name, whatever the particular agent supports.

7.3. Alert Meta-Attributes

As with resource agents, meta-attributes can be configured for alert agents to affect how Pacemaker calls them.

Table 7.1. Meta-Attributes of an Alert

Meta-Attribute	Default	Description
timestamp-format	%H:%M:%S.%06N	Format the cluster will use when sending the event's timestamp to the agent. This is a string as used with the date(1) command.
timeout	30s	If the alert agent does not complete within this amount of time, it will be terminated.

Meta-attributes can be configured per alert agent and/or per recipient.

Example 7.3. Alert configuration with meta-attributes

```
<configuration>
  <alerts>
    <alert id="my-alert" path="/path/to/my-script.sh">
      <meta_attributes id="my-alert-attributes">
        <nvpair id="my-alert-attributes-timeout" name="timeout"
          value="15s"/>
      </meta_attributes>
      <recipient id="my-alert-recipient1" value="someuser@example.com">
        <meta_attributes id="my-alert-recipient1-attributes">
          <nvpair id="my-alert-recipient1-timestamp-format"
            name="timestamp-format" value="%D %H:%M"/>
        </meta_attributes>
      </recipient>
      <recipient id="my-alert-recipient2" value="otheruser@example.com">
        <meta_attributes id="my-alert-recipient2-attributes">
          <nvpair id="my-alert-recipient2-timestamp-format"
            name="timestamp-format" value="%c"/>
        </meta_attributes>
      </recipient>
    </alert>
  </alerts>
</configuration>
```

In the above example, the **my-script.sh** will get called twice for each event, with each call using a 15-second timeout. One call will be passed the recipient **someuser@example.com** and a timestamp in the format **%D %H:%M**, while the other call will be passed the recipient **otheruser@example.com** and a timestamp in the format **%c**.

7.4. Alert Instance Attributes

As with resource agents, agent-specific configuration values may be configured as instance attributes. These will be passed to the agent as additional environment variables. The number, names and allowed values of these instance attributes are completely up to the particular agent.

Example 7.4. Alert configuration with instance attributes

```
<configuration>
  <alerts>
    <alert id="my-alert" path="/path/to/my-script.sh">
      <meta_attributes id="my-alert-attributes">
        <nvpair id="my-alert-attributes-timeout" name="timeout"
          value="15s"/>
      </meta_attributes>
      <instance_attributes id="my-alert-options">
        <nvpair id="my-alert-options-debug" name="debug" value="false"/>
      </instance_attributes>
      <recipient id="my-alert-recipient1" value="someuser@example.com"/>
    </alert>
  </alerts>
</configuration>
```

7.5. Using the Sample Alert Agents

Pacemaker provides several sample alert agents, installed in `/usr/share/pacemaker/alerts` by default.

While these sample scripts may be copied and used as-is, they are provided mainly as templates to be edited to suit your purposes. See their source code for the full set of instance attributes they support.

Example 7.5. Sending cluster events as SNMP traps

```
<configuration>
  <alerts>
    <alert id="snmp_alert" path="/path/to/alert_snmp.sh">
      <instance_attributes id="config_for_alert_snmp">
        <nvpair id="trap_node_states" name="trap_node_states" value="all"/>
      </instance_attributes>
      <meta_attributes id="config_for_timestamp">
        <nvpair id="ts_fmt" name="timestamp-format"
          value="%Y-%m-%d,%H:%M:%S.%01N"/>
      </meta_attributes>
      <recipient id="snmp_destination" value="192.168.1.2"/>
    </alert>
  </alerts>
</configuration>
```

Example 7.6. Sending cluster events as e-mails

```
<configuration>
  <alerts>
    <alert id="smtp_alert" path="/path/to/alert_smtp.sh">
      <instance_attributes id="config_for_alert_smtp">
        <nvpair id="email_sender" name="email_sender"
          value="donotreply@example.com"/>
      </instance_attributes>
      <recipient id="smtp_destination" value="admin@example.com"/>
    </alert>
  </alerts>
</configuration>
```

7.6. Writing an Alert Agent

Table 7.2. Environment variables passed to alert agents

Environment Variable	Description
CRM_alert_kind	The type of alert (node , fencing , or resource)
CRM_alert_version	The version of Pacemaker sending the alert
CRM_alert_recipient	The configured recipient
CRM_alert_node_sequence	A sequence number increased whenever an alert is being issued on the local node, which can be used to reference the order in which alerts have been issued by Pacemaker. An alert for an event that happened later in time reliably has a higher sequence number than alerts for earlier events. Be aware that this number has no cluster-wide meaning.
CRM_alert_timestamp	A timestamp created prior to executing the agent, in the format specified by the timestamp-format meta-attribute. This allows the agent to have a reliable, high-precision time of when the event occurred, regardless of when the agent itself was invoked (which could potentially be delayed due to system load, etc.).
CRM_alert_node	Name of affected node
CRM_alert_desc	Detail about event. For node alerts, this is the node's current state (member or lost). For fencing alerts, this is a summary of the requested fencing operation, including origin, target, and fencing operation error code, if any. For resource alerts, this is a readable string equivalent of CRM_alert_status .
CRM_alert_nodeid	ID of node whose status changed (provided with node alerts only)
CRM_alert_task	The requested fencing or resource operation (provided with fencing and resource alerts only)
CRM_alert_rc	The numerical return code of the fencing or resource operation (provided with fencing and resource alerts only)
CRM_alert_rsc	The name of the affected resource (resource alerts only)
CRM_alert_interval	The interval of the resource operation (resource alerts only)
CRM_alert_target_rc	The expected numerical return code of the operation (resource alerts only)
CRM_alert_status	A numerical code used by Pacemaker to represent the operation result (resource alerts only)

Special concerns when writing alert agents:

- Alert agents may be called with no recipient (if none is configured), so the agent must be able to handle this situation, even if it only exits in that case. (Users may modify the configuration in stages, and add a recipient later.)
- If more than one recipient is configured for an alert, the alert agent will be called once per recipient. If an agent is not able to run concurrently, it should be configured with only a single recipient. The agent is free, however, to interpret the recipient as a list.

- When a cluster event occurs, all alerts are fired off at the same time as separate processes. Depending on how many alerts and recipients are configured, and on what is done within the alert agents, a significant load burst may occur. The agent could be written to take this into consideration, for example by queueing resource-intensive actions into some other instance, instead of directly executing them.
- Alert agents are run as the **hacluster** user, which has a minimal set of permissions. If an agent requires additional privileges, it is recommended to configure **sudo** to allow the agent to run the necessary commands as another user with the appropriate privileges.
- As always, take care to validate and sanitize user-configured parameters, such as `CRM_alert_timestamp` (whose content is specified by the user-configured `timestamp-format`), `CRM_alert_recipient`, and all instance attributes. Mostly this is needed simply to protect against configuration errors, but if some user can modify the CIB without having `hacluster`-level access to the cluster nodes, it is a potential security concern as well, to avoid the possibility of code injection.



Note

The alerts interface is designed to be backward compatible with the external scripts interface used by the **ocf:pacemaker:ClusterMon** resource, which is now deprecated. To preserve this compatibility, the environment variables passed to alert agents are available prepended with **CRM_notify_** as well as **CRM_alert_**. One break in compatibility is that ClusterMon ran external scripts as the **root** user, while alert agents are run as the **hacluster** user.

Rules

Table of Contents

8.1. Rule Properties	61
8.2. Node Attribute Expressions	62
8.3. Time- and Date-Based Expressions	63
8.3.1. Date Specifications	63
8.3.2. Durations	64
8.3.3. Sample Time-Based Expressions	64
8.4. Using Rules to Determine Resource Location	66
8.4.1. Location Rules Based on Other Node Properties	66
8.4.2. Using score-attribute Instead of score	67
8.5. Using Rules to Control Resource Options	67
8.6. Using Rules to Control Cluster Options	68
8.7. Ensuring Time-Based Rules Take Effect	68

Rules can be used to make your configuration more dynamic. One common example is to set one value for **resource-stickiness** during working hours, to prevent resources from being moved back to their most preferred location, and another on weekends when no-one is around to notice an outage.

Another use of rules might be to assign machines to different processing groups (using a node attribute) based on time and to then use that attribute when creating location constraints.

Each rule can contain a number of expressions, date-expressions and even other rules. The results of the expressions are combined based on the rule's **boolean-op** field to determine if the rule ultimately evaluates to **true** or **false**. What happens next depends on the context in which the rule is being used.

8.1. Rule Properties

Table 8.1. Properties of a Rule

Field	Default	Description
role	Started	Limits the rule to apply only when the resource is in the specified role. Allowed values are Started , Slave , and Master . A rule with role="Master" cannot determine the initial location of a clone instance and will only affect which of the active instances will be promoted.
score		The score to apply if the rule evaluates to true . Limited to use in rules that are part of location constraints.
score-attribute		The node attribute to look up and use as a score if the rule evaluates to true . Limited to use in rules that are part of location constraints.
boolean-op	and	How to combine the result of multiple expression objects. Allowed values are and and or .

8.2. Node Attribute Expressions

Expression objects are used to control a resource based on the attributes defined by a node or nodes.

Table 8.2. Properties of an Expression

Field	Default	Description
value		User-supplied value for comparison
attribute		The node attribute to test
type	string	Determines how the value(s) should be tested. Allowed values are string , integer , and version .
operation		<p>The comparison to perform. Allowed values:</p> <ul style="list-style-type: none"> • lt: True if the value of the node's attribute is less than value • gt: True if the value of the node's attribute is greater than value • lte: True if the value of the node's attribute is less than or equal to value • gte: True if the value of the node's attribute is greater than or equal to value • eq: True if the value of the node's attribute is equal to value • ne: True if the value of the node's attribute is not equal to value • defined: True if the node has the named attribute • not_defined: True if the node does not have the named attribute

In addition to any attributes added by the administrator, the cluster defines special, built-in node attributes for each node that can also be used.

Table 8.3. Built-in node attributes

Name	Value
#uname	Node name
#kind	Node type. Possible values are cluster , remote , and container . Kind is remote for Pacemaker Remote nodes created with the ocf:pacemaker:remote resource, and container for Pacemaker Remote guest nodes (a legacy name unrelated to the now-common use of "container" for resource isolation). (<i>since 1.1.13</i>)
#ra-version	The installed version of the resource agent on the node, as defined by the version attribute of the resource-agent tag in the agent's metadata. Valid only within rules controlling resource options. This can be useful during rolling upgrades of a backward-incompatible resource agent. (<i>coming in 1.1.16</i>)

8.3. Time- and Date-Based Expressions

As the name suggests, **date_expressions** are used to control a resource or cluster option based on the current date/time. They may contain an optional **date_spec** and/or **duration** object depending on the context.

Table 8.4. Properties of a Date Expression

Field	Description
start	A date/time conforming to the ISO8601 ¹ specification.
end	A date/time conforming to the ISO8601 ² specification. Can be inferred by supplying a value for start and a duration .
operation	Compares the current date/time with the start and/or end date, depending on the context. Allowed values: <ul style="list-style-type: none"> • gt: True if the current date/time is after start • lt: True if the current date/time is before end • in-range: True if the current date/time is after start and before end • date-spec: True if the current date/time matches a date_spec object (described below)



Note

As these comparisons (except for **date_spec**) include the time, the **eq**, **neq**, **gte** and **lte** operators have not been implemented since they would only be valid for a single second.

8.3.1. Date Specifications

date_spec objects are used to create cron-like expressions relating to time. Each field can contain a single number or a single range. Instead of defaulting to zero, any field not supplied is ignored.

For example, **monthdays="1"** matches the first day of every month and **hours="09-17"** matches the hours between 9am and 5pm (inclusive). At this time, multiple ranges (e.g. **weekdays="1, 2"** or **weekdays="1-2, 5-6"**) are not supported; depending on demand, this might be implemented in a future release.

Table 8.5. Properties of a Date Specification

Field	Description
id	A unique name for the object
hours	Allowed values: 0-23

¹ http://en.wikipedia.org/wiki/ISO_8601

² http://en.wikipedia.org/wiki/ISO_8601

Field	Description
monthdays	Allowed values: 1-31 (depending on month and year)
weekdays	Allowed values: 1-7 (1=Monday, 7=Sunday)
yeardays	Allowed values: 1-366 (depending on the year)
months	Allowed values: 1-12
weeks	Allowed values: 1-53 (depending on weekyear)
years	Year according to the Gregorian calendar
weekyears	Year in which the week started; e.g. 1 January 2005 can be specified as <i>2005-001 Ordinal</i> , <i>2005-01-01 Gregorian</i> or <i>2004-W53-6 Weekly</i> and thus would match years="2005" or weekyears="2004"
moon	Allowed values are 0-7 (0 is new, 4 is full moon). Seriously, you can use this. This was implemented to demonstrate the ease with which new comparisons could be added.

8.3.2. Durations

Durations are used to calculate a value for **end** when one is not supplied to **in-range** operations. They contain the same fields as **date_spec** objects but without the limitations (e.g. you can have a duration of 19 months). As with **date_specs**, any field not supplied is ignored.

8.3.3. Sample Time-Based Expressions

A small sample of how time-based expressions can be used:

Example 8.1. True if now is any time in the year 2005

```
<rule id="rule1">
  <date_expression id="date_expr1" start="2005-001" operation="in_range">
    <duration years="1"/>
  </date_expression>
</rule>
```

Example 8.2. Equivalent expression

```
<rule id="rule2">
  <date_expression id="date_expr2" operation="date_spec">
    <date_spec years="2005"/>
  </date_expression>
</rule>
```

Example 8.3. 9am-5pm Monday-Friday

```
<rule id="rule3">
  <date_expression id="date_expr3" operation="date_spec">
    <date_spec hours="9-16" days="1-5"/>
  </date_expression>
</rule>
```

Please note that the **16** matches up to **16:59:59**, as the numeric value (hour) still matches!

Example 8.4. 9am-6pm Monday through Friday or anytime Saturday

```
<rule id="rule4" boolean_op="or">
  <date_expression id="date_expr4-1" operation="date_spec">
    <date_spec hours="9-16" days="1-5"/>
  </date_expression>
  <date_expression id="date_expr4-2" operation="date_spec">
    <date_spec days="6"/>
  </date_expression>
</rule>
```

Example 8.5. 9am-5pm or 9pm-12am Monday through Friday

```
<rule id="rule5" boolean_op="and">
  <rule id="rule5-nested1" boolean_op="or">
    <date_expression id="date_expr5-1" operation="date_spec">
      <date_spec hours="9-16"/>
    </date_expression>
    <date_expression id="date_expr5-2" operation="date_spec">
      <date_spec hours="21-23"/>
    </date_expression>
  </rule>
  <date_expression id="date_expr5-3" operation="date_spec">
    <date_spec days="1-5"/>
  </date_expression>
</rule>
```

Example 8.6. Mondays in March 2005

```
<rule id="rule6" boolean_op="and">
  <date_expression id="date_expr6-1" operation="date_spec">
    <date_spec weekdays="1"/>
  </date_expression>
  <date_expression id="date_expr6-2" operation="in_range">
    <start="2005-03-01" end="2005-04-01"/>
  </date_expression>
</rule>
```



Note

Because no time is specified with the above dates, 00:00:00 is implied. This means that the range includes all of 2005-03-01 but none of 2005-04-01. You may wish to write **end="2005-03-31T23:59:59"** to avoid confusion.

Example 8.7. A full moon on Friday the 13th

```
<rule id="rule7" boolean_op="and">
  <date_expression id="date_expr7" operation="date_spec">
    <date_spec weekdays="5" monthdays="13" moon="4"/>
  </date_expression>
</rule>
```

8.4. Using Rules to Determine Resource Location

A location constraint may contain rules. When the constraint's outermost rule evaluates to **false**, the cluster treats the constraint as if it were not there. When the rule evaluates to **true**, the node's preference for running the resource is updated with the score associated with the rule.

If this sounds familiar, it is because you have been using a simplified syntax for location constraint rules already. Consider the following location constraint:

Example 8.8. Prevent myApacheRsc from running on c001n03

```
<rsc_location id="dont-run-apache-on-c001n03" rsc="myApacheRsc"
             score="-INFINITY" node="c001n03"/>
```

This constraint can be more verbosely written as:

Example 8.9. Prevent myApacheRsc from running on c001n03 - expanded version

```
<rsc_location id="dont-run-apache-on-c001n03" rsc="myApacheRsc">
  <rule id="dont-run-apache-rule" score="-INFINITY">
    <expression id="dont-run-apache-expr" attribute="#uname"
               operation="eq" value="c00n03"/>
  </rule>
</rsc_location>
```

The advantage of using the expanded form is that one can then add extra clauses to the rule, such as limiting the rule such that it only applies during certain times of the day or days of the week.

8.4.1. Location Rules Based on Other Node Properties

The expanded form allows us to match on node properties other than its name. If we rated each machine's CPU power such that the cluster had the following nodes section:

Example 8.10. A sample nodes section for use with score-attribute

```
<nodes>
  <node id="uuid1" uname="c001n01" type="normal">
    <instance_attributes id="uuid1-custom_attrs">
      <nvpair id="uuid1-cpu_mips" name="cpu_mips" value="1234"/>
    </instance_attributes>
  </node>
  <node id="uuid2" uname="c001n02" type="normal">
    <instance_attributes id="uuid2-custom_attrs">
      <nvpair id="uuid2-cpu_mips" name="cpu_mips" value="5678"/>
    </instance_attributes>
  </node>
</nodes>
```

then we could prevent resources from running on underpowered machines with this rule:

```
<rule id="need-more-power-rule" score="-INFINITY">
  <expression id="need-more-power-expr" attribute="cpu_mips"
             operation="lt" value="3000"/>
```

```
</rule>
```

8.4.2. Using **score-attribute** Instead of **score**

When using **score-attribute** instead of **score**, each node matched by the rule has its score adjusted differently, according to its value for the named node attribute. Thus, in the previous example, if a rule used **score-attribute="cpu_mips"**, **c001n01** would have its preference to run the resource increased by **1234** whereas **c001n02** would have its preference increased by **5678**.

8.5. Using Rules to Control Resource Options

Often some cluster nodes will be different from their peers. Sometimes, these differences — e.g. the location of a binary or the names of network interfaces — require resources to be configured differently depending on the machine they're hosted on.

By defining multiple **instance_attributes** objects for the resource and adding a rule to each, we can easily handle these special cases.

In the example below, **mySpecialRsc** will use **eth1** and port **9999** when run on **node1**, **eth2** and port **8888** on **node2** and default to **eth0** and port **9999** for all other nodes.

Example 8.11. Defining different resource options based on the node name

```
<primitive id="mySpecialRsc" class="ocf" type="Special" provider="me">
  <instance_attributes id="special-node1" score="3">
    <rule id="node1-special-case" score="INFINITY" >
      <expression id="node1-special-case-expr" attribute="#uname"
        operation="eq" value="node1"/>
    </rule>
    <nvpair id="node1-interface" name="interface" value="eth1"/>
  </instance_attributes>
  <instance_attributes id="special-node2" score="2" >
    <rule id="node2-special-case" score="INFINITY">
      <expression id="node2-special-case-expr" attribute="#uname"
        operation="eq" value="node2"/>
    </rule>
    <nvpair id="node2-interface" name="interface" value="eth2"/>
    <nvpair id="node2-port" name="port" value="8888"/>
  </instance_attributes>
  <instance_attributes id="defaults" score="1" >
    <nvpair id="default-interface" name="interface" value="eth0"/>
    <nvpair id="default-port" name="port" value="9999"/>
  </instance_attributes>
</primitive>
```

The order in which **instance_attributes** objects are evaluated is determined by their score (highest to lowest). If not supplied, score defaults to zero, and objects with an equal score are processed in listed order. If the **instance_attributes** object has no rule or a **rule** that evaluates to **true**, then for any parameter the resource does not yet have a value for, the resource will use the parameter values defined by the **instance_attributes**.

For example, given the configuration above, if the resource is placed on node1:

1. **special-node1** has the highest score (3) and so is evaluated first; its rule evaluates to **true**, so **interface** is set to **eth1**.
2. **special-node2** is evaluated next with score 2, but its rule evaluates to **false**, so it is ignored.

3. **defaults** is evaluated last with score 1, and has no rule, so its values are examined; **interface** is already defined, so the value here is not used, but **port** is not yet defined, so **port** is set to **9999**.

8.6. Using Rules to Control Cluster Options

Controlling cluster options is achieved in much the same manner as specifying different resource options on different nodes.

The difference is that because they are cluster options, one cannot (or should not, because they won't work) use attribute-based expressions. The following example illustrates how to set a different **resource-stickiness** value during and outside work hours. This allows resources to automatically move back to their most preferred hosts, but at a time that (in theory) does not interfere with business activities.

Example 8.12. Change **resource-stickiness** during working hours

```
<rsc_defaults>
  <meta_attributes id="core-hours" score="2">
    <rule id="core-hour-rule" score="0">
      <date_expression id="nine-to-five-Mon-to-Fri" operation="date_spec">
        <date_spec id="nine-to-five-Mon-to-Fri-spec" hours="9-16" weekdays="1-5"/>
      </date_expression>
    </rule>
    <nvpair id="core-stickiness" name="resource-stickiness" value="INFINITY"/>
  </meta_attributes>
  <meta_attributes id="after-hours" score="1" >
    <nvpair id="after-stickiness" name="resource-stickiness" value="0"/>
  </meta_attributes>
</rsc_defaults>
```

8.7. Ensuring Time-Based Rules Take Effect

A Pacemaker cluster is an event-driven system. As such, it won't recalculate the best place for resources to run unless something (like a resource failure or configuration change) happens. This can mean that a location constraint that only allows resource X to run between 9am and 5pm is not enforced.

If you rely on time-based rules, the **cluster-recheck-interval** cluster option (which defaults to 15 minutes) is essential. This tells the cluster to periodically recalculate the ideal state of the cluster.

For example, if you set **cluster-recheck-interval="5m"**, then sometime between 09:00 and 09:05 the cluster would notice that it needs to start resource X, and between 17:00 and 17:05 it would realize that X needed to be stopped. The timing of the actual start and stop actions depends on what other actions the cluster may need to perform first.

Advanced Configuration

Table of Contents

9.1. Connecting from a Remote Machine	69
9.2. Specifying When Recurring Actions are Performed	70
9.3. Moving Resources	70
9.3.1. Moving Resources Manually	70
9.3.2. Moving Resources Due to Failure	72
9.3.3. Moving Resources Due to Connectivity Changes	73
9.3.4. Migrating Resources	75
9.4. Reusing Rules, Options and Sets of Operations	76
9.5. Reloading Services After a Definition Change	77

9.1. 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 9.1. 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 hacluster 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 **hacluster** 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 9.2. Extra top-level CIB properties for remote access

Field	Default	Description
remote-tls-port	<i>none</i>	Listen for encrypted remote connections on this port.
remote-clear-port	<i>none</i>	Listen for plaintext remote connections on this port.

9.2. Specifying When Recurring Actions are Performed

By default, recurring actions are scheduled relative to when the resource started. So if your resource was last started at 14:32 and you have a backup set to be performed every 24 hours, then the backup will always run in the middle of the business day — hardly desirable.

To specify a date and time that the operation should be relative to, set the operation's **interval-origin**. The cluster uses this point to calculate the correct **start-delay** such that the operation will occur at *origin + (interval * N)*.

So, if the operation's interval is 24h, its interval-origin is set to 02:00 and it is currently 14:32, then the cluster would initiate the operation with a start delay of 11 hours and 28 minutes. If the resource is moved to another node before 2am, then the operation is cancelled.

The value specified for **interval** and **interval-origin** can be any date/time conforming to the [ISO8601 standard](http://en.wikipedia.org/wiki/ISO_8601)¹. By way of example, to specify an operation that would run on the first Monday of 2009 and every Monday after that, you would add:

Example 9.1. Specifying a Base for Recurring Action Intervals

```
<op id="my-weekly-action" name="custom-action" interval="P7D" interval-origin="2009-
W01-1"/>
```

9.3. Moving Resources

9.3.1. Moving Resources Manually

There are primarily two occasions when you would want to move a resource from its current location: when the whole node is under maintenance, and when a single resource needs to be moved.

9.3.1.1. Standby Mode

Since everything eventually comes down to a score, you could create constraints for every resource to prevent them from running on one node. While pacemaker configuration can seem convoluted at times, not even we would require this of administrators.

Instead, one can set a special node attribute which tells the cluster "don't let anything run here". There is even a helpful tool to help query and set it, called **crm_standby**. To check the standby status of the current machine, run:

```
# crm_standby -G
```

A value of **on** indicates that the node is *not* able to host any resources, while a value of **off** says that it *can*.

You can also check the status of other nodes in the cluster by specifying the **--node** option:

```
# crm_standby -G --node sles-2
```

¹ http://en.wikipedia.org/wiki/ISO_8601

To change the current node's standby status, use **-v** instead of **-G**:

```
# crm_standby -v on
```

Again, you can change another host's value by supplying a hostname with **- -node**.

9.3.1.2. Moving One Resource

When only one resource is required to move, we could do this by creating location constraints. However, once again we provide a user-friendly shortcut as part of the **crm_resource** command, which creates and modifies the extra constraints for you. If **Email** were running on **sles-1** and you wanted it moved to a specific location, the command would look something like:

```
# crm_resource -M -r Email -H sles-2
```

Behind the scenes, the tool will create the following location constraint:

```
<rsc_location rsc="Email" node="sles-2" score="INFINITY"/>
```

It is important to note that subsequent invocations of **crm_resource -M** are not cumulative. So, if you ran these commands

```
# crm_resource -M -r Email -H sles-2
# crm_resource -M -r Email -H sles-3
```

then it is as if you had never performed the first command.

To allow the resource to move back again, use:

```
# crm_resource -U -r Email
```

Note the use of the word *allow*. The resource can move back to its original location but, depending on **resource-stickiness**, it might stay where it is. To be absolutely certain that it moves back to **sles-1**, move it there before issuing the call to **crm_resource -U**:

```
# crm_resource -M -r Email -H sles-1
# crm_resource -U -r Email
```

Alternatively, if you only care that the resource should be moved from its current location, try:

```
# crm_resource -B -r Email
```

Which will instead create a negative constraint, like

```
<rsc_location rsc="Email" node="sles-1" score="-INFINITY"/>
```

This will achieve the desired effect, but will also have long-term consequences. As the tool will warn you, the creation of a **-INFINITY** constraint will prevent the resource from running on that node until **crm_resource -U** is used. This includes the situation where every other cluster node is no longer available!

In some cases, such as when **resource-stickiness** is set to **INFINITY**, it is possible that you will end up with the problem described in [Section 6.2.4, "What if Two Nodes Have the Same Score"](#).

The tool can detect some of these cases and deals with them by creating both positive and negative constraints. E.g.

Email prefers **sles-1** with a score of **-INFINITY**

Email prefers **sles-2** with a score of **INFINITY**

which has the same long-term consequences as discussed earlier.

9.3.2. Moving Resources Due to Failure

Normally, if a running resource fails, pacemaker will try to start it again on the same node. However if a resource fails repeatedly, it is possible that there is an underlying problem on that node, and you might desire trying a different node in such a case.

Pacemaker allows you to set your preference via the **migration-threshold** resource option.²

Simply define **migration-threshold=N** for a resource and it will migrate to a new node after *N* failures. There is no threshold defined by default. To determine the resource's current failure status and limits, run **crm_mon --failcounts**.

By default, once the threshold has been reached, the troublesome node will no longer be allowed to run the failed resource until the administrator manually resets the resource's failcount using **crm_failcount** (after hopefully first fixing the failure's cause). Alternatively, it is possible to expire them by setting the **failure-timeout** option for the resource.

For example, a setting of **migration-threshold=2** and **failure-timeout=60s** would cause the resource to move to a new node after 2 failures, and allow it to move back (depending on stickiness and constraint scores) after one minute.

There are two exceptions to the migration threshold concept: when a resource either fails to start or fails to stop.

If the cluster property **start-failure-is-fatal** is set to **true** (which is the default), start failures cause the failcount to be set to **INFINITY** and thus always cause the resource to move immediately.

Stop failures are slightly different and crucial. If a resource fails to stop and STONITH is enabled, then the cluster will fence the node in order to be able to start the resource elsewhere. If STONITH is not enabled, then the cluster has no way to continue and will not try to start the resource elsewhere, but will try to stop it again after the failure timeout.



Important

Please read [Section 8.7, "Ensuring Time-Based Rules Take Effect"](#) to understand how timeouts work before configuring a **failure-timeout**.

² The naming of this option was perhaps unfortunate as it is easily confused with live migration, the process of moving a resource from one node to another without stopping it. Xen virtual guests are the most common example of resources that can be migrated in this manner.

9.3.3. Moving Resources Due to Connectivity Changes

You can configure the cluster to move resources when external connectivity is lost in two steps.

9.3.3.1. Tell Pacemaker to Monitor Connectivity

First, add an **ocf:pacemaker:ping** resource to the cluster. The **ping** resource uses the system utility of the same name to test whether a list of machines (specified by DNS hostname or IPv4/IPv6 address) are reachable and uses the results to maintain a node attribute called **pingd** by default.³



Note

Older versions of Heartbeat required users to add ping nodes to **ha.cf**, but this is no longer required.

Older versions of Pacemaker used a different agent **ocf:pacemaker:pingd** which is now deprecated in favor of **ping**. If your version of Pacemaker does not contain the **ping** resource agent, download the latest version from <https://github.com/ClusterLabs/pacemaker/tree/master/extra/resources/ping>

Normally, the ping resource should run on all cluster nodes, which means that you'll need to create a clone. A template for this can be found below along with a description of the most interesting parameters.

Table 9.3. Common Options for a *ping* Resource

Field	Description
dampen	The time to wait (dampening) for further changes to occur. Use this to prevent a resource from bouncing around the cluster when cluster nodes notice the loss of connectivity at slightly different times.
multiplier	The number of connected ping nodes gets multiplied by this value to get a score. Useful when there are multiple ping nodes configured.
host_list	The machines to contact in order to determine the current connectivity status. Allowed values include resolvable DNS host names, IPv4 and IPv6 addresses.

Example 9.2. An example ping cluster resource that checks node connectivity once every minute

```
<clone id="Connected">
  <primitive id="ping" provider="pacemaker" class="ocf" type="ping">
    <instance_attributes id="ping-attrs">
      <nvpair id="pingd-dampen" name="dampen" value="5s"/>
      <nvpair id="pingd-multiplier" name="multiplier" value="1000"/>
      <nvpair id="pingd-hosts" name="host_list" value="my.gateway.com www.bigcorp.com"/>
    </instance_attributes>
    <operations>
      <op id="ping-monitor-60s" interval="60s" name="monitor"/>
    </operations>
  </primitive>
</clone>
```

³ The attribute name is customizable, in order to allow multiple ping groups to be defined.



Important

You're only half done. The next section deals with telling Pacemaker how to deal with the connectivity status that `ocf:pacemaker:ping` is recording.

9.3.3.2. Tell Pacemaker How to Interpret the Connectivity Data



Important

Before attempting the following, make sure you understand [Chapter 8, Rules](#).

There are a number of ways to use the connectivity data.

The most common setup is for people to have a single ping target (e.g. the service network's default gateway), to prevent the cluster from running a resource on any unconnected node.

Example 9.3. Don't run a resource on unconnected nodes

```
<rsc_location id="WebServer-no-connectivity" rsc="Webserver">
  <rule id="ping-exclude-rule" score="-INFINITY" >
    <expression id="ping-exclude" attribute="pingd" operation="not_defined"/>
  </rule>
</rsc_location>
```

A more complex setup is to have a number of ping targets configured. You can require the cluster to only run resources on nodes that can connect to all (or a minimum subset) of them.

Example 9.4. Run only on nodes connected to three or more ping targets.

```
<primitive id="ping" provider="pacemaker" class="ocf" type="ping">
... <!-- omitting some configuration to highlight important parts -->
  <nvpair id="pingd-multiplier" name="multiplier" value="1000"/>
...
</primitive>
...
<rsc_location id="WebServer-connectivity" rsc="Webserver">
  <rule id="ping-prefer-rule" score="-INFINITY" >
    <expression id="ping-prefer" attribute="pingd" operation="lt" value="3000"/>
  </rule>
</rsc_location>
```

Alternatively, you can tell the cluster only to *prefer* nodes with the best connectivity. Just be sure to set **multiplier** to a value higher than that of **resource-stickiness** (and don't set either of them to **INFINITY**).

Example 9.5. Prefer the node with the most connected ping nodes

```
<rsc_location id="WebServer-connectivity" rsc="Webserver">
  <rule id="ping-prefer-rule" score-attribute="pingd" >
    <expression id="ping-prefer" attribute="pingd" operation="defined"/>
  </rule>
</rsc_location>
```

It is perhaps easier to think of this in terms of the simple constraints that the cluster translates it into. For example, if **sles-1** is connected to all five ping nodes but **sles-2** is only connected to two, then it would be as if you instead had the following constraints in your configuration:

Example 9.6. How the cluster translates the above location constraint

```
<rsc_location id="ping-1" rsc="Webserver" node="sles-1" score="5000"/>
<rsc_location id="ping-2" rsc="Webserver" node="sles-2" score="2000"/>
```

The advantage is that you don't have to manually update any constraints whenever your network connectivity changes.

You can also combine the concepts above into something even more complex. The example below shows how you can prefer the node with the most connected ping nodes provided they have connectivity to at least three (again assuming that **multiplier** is set to 1000).

Example 9.7. A more complex example of choosing a location based on connectivity

```
<rsc_location id="WebServer-connectivity" rsc="Webserver">
  <rule id="ping-exclude-rule" score="-INFINITY" >
    <expression id="ping-exclude" attribute="pingd" operation="lt" value="3000"/>
  </rule>
  <rule id="ping-prefer-rule" score-attribute="pingd" >
    <expression id="ping-prefer" attribute="pingd" operation="defined"/>
  </rule>
</rsc_location>
```

9.3.4. Migrating Resources

Normally, when the cluster needs to move a resource, it fully restarts the resource (i.e. stops the resource on the current node and starts it on the new node).

However, some types of resources, such as Xen virtual guests, are able to move to another location without loss of state (often referred to as live migration or hot migration). In pacemaker, this is called resource migration. Pacemaker can be configured to migrate a resource when moving it, rather than restarting it.

Not all resources are able to migrate; see the Migration Checklist below, and those that can, won't do so in all situations. Conceptually, there are two requirements from which the other prerequisites follow:

- The resource must be active and healthy at the old location; and
- everything required for the resource to run must be available on both the old and new locations.

The cluster is able to accommodate both *push* and *pull* migration models by requiring the resource agent to support two special actions: **migrate_to** (performed on the current location) and **migrate_from** (performed on the destination).

In push migration, the process on the current location transfers the resource to the new location where it is later activated. In this scenario, most of the work would be done in the **migrate_to** action and, if anything, the activation would occur during **migrate_from**.

Conversely for pull, the **migrate_to** action is practically empty and **migrate_from** does most of the work, extracting the relevant resource state from the old location and activating it.

There is no wrong or right way for a resource agent to implement migration, as long as it works.

Migration Checklist

- The resource may not be a clone.
- The resource must use an OCF style agent.
- The resource must not be in a failed or degraded state.
- The resource agent must support **migrate_to** and **migrate_from** actions, and advertise them in its metadata.
- The resource must have the **allow-migrate** meta-attribute set to **true** (which is not the default).

If an otherwise migratable resource depends on another resource via an ordering constraint, there are special situations in which it will be restarted rather than migrated.

For example, if the resource depends on a clone, and at the time the resource needs to be moved, the clone has instances that are stopping and instances that are starting, then the resource will be restarted. The Policy Engine is not yet able to model this situation correctly and so takes the safer (if less optimal) path.

In pacemaker 1.1.11 and earlier, a migratable resource will be restarted when moving if it directly or indirectly depends on *any* primitive or group resources.

Even in newer versions, if a migratable resource depends on a non-migratable resource, and both need to be moved, the migratable resource will be restarted.

9.4. Reusing Rules, Options and Sets of Operations

Sometimes a number of constraints need to use the same set of rules, and resources need to set the same options and parameters. To simplify this situation, you can refer to an existing object using an **id-ref** instead of an id.

So if for one resource you have

```
<rsc_location id="WebServer-connectivity" rsc="Webserver">
  <rule id="ping-prefer-rule" score-attribute="pingd" >
    <expression id="ping-prefer" attribute="pingd" operation="defined"/>
  </rule>
</rsc_location>
```

Then instead of duplicating the rule for all your other resources, you can instead specify:

Example 9.8. Referencing rules from other constraints

```
<rsc_location id="WebDB-connectivity" rsc="WebDB">
  <rule id-ref="ping-prefer-rule"/>
```



```
</rsc_location>
```



Important

The cluster will insist that the **rule** exists somewhere. Attempting to add a reference to a non-existing rule will cause a validation failure, as will attempting to remove a **rule** that is referenced elsewhere.

The same principle applies for **meta_attributes** and **instance_attributes** as illustrated in the example below:

Example 9.9. Referencing attributes, options, and operations from other resources

```
<primitive id="mySpecialRsc" class="ocf" type="Special" provider="me">
  <instance_attributes id="mySpecialRsc-attrs" score="1" >
    <nvpair id="default-interface" name="interface" value="eth0"/>
    <nvpair id="default-port" name="port" value="9999"/>
  </instance_attributes>
  <meta_attributes id="mySpecialRsc-options">
    <nvpair id="failure-timeout" name="failure-timeout" value="5m"/>
    <nvpair id="migration-threshold" name="migration-threshold" value="1"/>
    <nvpair id="stickiness" name="resource-stickiness" value="0"/>
  </meta_attributes>
  <operations id="health-checks">
    <op id="health-check" name="monitor" interval="60s"/>
    <op id="health-check" name="monitor" interval="30min"/>
  </operations>
</primitive>
<primitive id="myOtherRsc" class="ocf" type="Other" provider="me">
  <instance_attributes id-ref="mySpecialRsc-attrs"/>
  <meta_attributes id-ref="mySpecialRsc-options"/>
  <operations id-ref="health-checks"/>
</primitive>
```

9.5. Reloading Services After a Definition Change

The cluster automatically detects changes to the definition of services it manages. The normal response is to stop the service (using the old definition) and start it again (with the new definition). This works well, but some services are smarter and can be told to use a new set of options without restarting.

To take advantage of this capability, the resource agent must:

1. Accept the **reload** operation and perform any required actions. *The actions here depend completely on your application!*

Example 9.10. The DRBD agent's logic for supporting **reload**

```
case $1 in
  start)
    drbd_start
    ;;
  stop)
    drbd_stop
```

```
;;
reload)
    drbd_reload
;;
monitor)
    drbd_monitor
;;
*)
    drbd_usage
    exit $OCF_ERR_UNIMPLEMENTED
;;
esac
exit $?
```

2. Advertise the **reload** operation in the **actions** section of its metadata

Example 9.11. The DRBD Agent Advertising Support for the **reload** Operation

```
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="drbd">
  <version>1.1</version>

  <longdesc>
    Master/Slave OCF Resource Agent for DRBD
  </longdesc>

  ...

  <actions>
    <action name="start"    timeout="240" />
    <action name="reload"   timeout="240" />
    <action name="promote"  timeout="90" />
    <action name="demote"   timeout="90" />
    <action name="notify"   timeout="90" />
    <action name="stop"     timeout="100" />
    <action name="meta-data" timeout="5" />
    <action name="validate-all" timeout="30" />
  </actions>
</resource-agent>
```

3. Advertise one or more parameters that can take effect using **reload**.

Any parameter with the **unique** set to 0 is eligible to be used in this way.

Example 9.12. Parameter that can be changed using reload

```
<parameter name="drbdconf" unique="0">
  <longdesc>Full path to the drbd.conf file.</longdesc>
  <shortdesc>Path to drbd.conf</shortdesc>
  <content type="string" default="${OCF_RESKEY_drbdconf_default}"/>
</parameter>
```

Once these requirements are satisfied, the cluster will automatically know to reload the resource (instead of restarting) when a non-unique field changes.

**Note**

Metadata will not be re-read unless the resource needs to be started. This may mean that the resource will be restarted the first time, even though you changed a parameter with **unique=0**.

**Note**

If both a unique and non-unique field are changed simultaneously, the resource will still be restarted.

Advanced Resource Types

Table of Contents

10.1. Groups - A Syntactic Shortcut	81
10.1.1. Group Properties	82
10.1.2. Group Options	82
10.1.3. Group Instance Attributes	82
10.1.4. Group Contents	82
10.1.5. Group Constraints	83
10.1.6. Group Stickiness	83
10.2. Clones - Resources That Get Active on Multiple Hosts	83
10.2.1. Clone Properties	84
10.2.2. Clone Options	84
10.2.3. Clone Instance Attributes	84
10.2.4. Clone Contents	84
10.2.5. Clone Constraints	85
10.2.6. Clone Stickiness	85
10.2.7. Clone Resource Agent Requirements	85
10.3. Multi-state - Resources That Have Multiple Modes	87
10.3.1. Multi-state Properties	88
10.3.2. Multi-state Options	88
10.3.3. Multi-state Instance Attributes	88
10.3.4. Multi-state Contents	88
10.3.5. Monitoring Multi-State Resources	88
10.3.6. Multi-state Constraints	89
10.3.7. Multi-state Stickiness	91
10.3.8. Which Resource Instance is Promoted	91
10.3.9. Requirements for Multi-state Resource Agents	91

10.1. Groups - A Syntactic Shortcut

One of the most common elements of a cluster is a set of resources that need to be located together, start sequentially, and stop in the reverse order. To simplify this configuration, we support the concept of groups.

Example 10.1. A group of two primitive resources

```
<group id="shortcut">
  <primitive id="Public-IP" class="ocf" type="IPaddr" provider="heartbeat">
    <instance_attributes id="params-public-ip">
      <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
    </instance_attributes>
  </primitive>
  <primitive id="Email" class="lsb" type="exim"/>
</group>
```

Although the example above contains only two resources, there is no limit to the number of resources a group can contain. The example is also sufficient to explain the fundamental properties of a group:

- Resources are started in the order they appear in (**Public-IP** first, then **Email**)
- Resources are stopped in the reverse order to which they appear in (**Email** first, then **Public-IP**)

If a resource in the group can't run anywhere, then nothing after that is allowed to run, too.

- If **Public-IP** can't run anywhere, neither can **Email**;
- but if **Email** can't run anywhere, this does not affect **Public-IP** in any way

The group above is logically equivalent to writing:

Example 10.2. How the cluster sees a group resource

```
<configuration>
<resources>
  <primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
    <instance_attributes id="params-public-ip">
      <nvpair id="public-ip-addr" name="ip" value="192.0.2.2"/>
    </instance_attributes>
  </primitive>
  <primitive id="Email" class="lsb" type="exim"/>
</resources>
<constraints>
  <rsc_colocation id="xxx" rsc="Email" with-rsc="Public-IP" score="INFINITY"/>
  <rsc_order id="yyy" first="Public-IP" then="Email"/>
</constraints>
</configuration>
```

Obviously as the group grows bigger, the reduced configuration effort can become significant.

Another (typical) example of a group is a DRBD volume, the filesystem mount, an IP address, and an application that uses them.

10.1.1. Group Properties

Table 10.1. Properties of a Group Resource

Field	Description
id	A unique name for the group

10.1.2. Group Options

Groups inherit the **priority**, **target-role**, and **is-managed** properties from primitive resources. See [Section 5.4, “Resource Options”](#) for information about those properties.

10.1.3. Group Instance Attributes

Groups have no instance attributes. However, any that are set for the group object will be inherited by the group's children.

10.1.4. Group Contents

Groups may only contain a collection of cluster resources (see [Section 5.3, “Resource Properties”](#)). To refer to a child of a group resource, just use the child's **id** instead of the group's.

10.1.5. Group Constraints

Although it is possible to reference a group's children in constraints, it is usually preferable to reference the group itself.

Example 10.3. Some constraints involving groups

```
<constraints>
  <rsc_location id="group-prefers-node1" rsc="shortcut" node="node1" score="500"/>
  <rsc_colocation id="webserver-with-group" rsc="Webserver" with-rsc="shortcut"/>
  <rsc_order id="start-group-then-webserver" first="Webserver" then="shortcut"/>
</constraints>
```

10.1.6. Group Stickiness

Stickiness, the measure of how much a resource wants to stay where it is, is additive in groups. Every active resource of the group will contribute its stickiness value to the group's total. So if the default **resource-stickiness** is 100, and a group has seven members, five of which are active, then the group as a whole will prefer its current location with a score of 500.

10.2. Clones - Resources That Get Active on Multiple Hosts

Clones were initially conceived as a convenient way to start multiple instances of an IP address resource and have them distributed throughout the cluster for load balancing. They have turned out to quite useful for a number of purposes including integrating with the Distributed Lock Manager (used by many cluster filesystems), the fencing subsystem, and OCFS2.

You can clone any resource, provided the resource agent supports it.

Three types of cloned resources exist:

- Anonymous
- Globally unique
- Stateful

Anonymous clones are the simplest. These behave completely identically everywhere they are running. Because of this, there can be only one copy of an anonymous clone active per machine.

Globally unique clones are distinct entities. A copy of the clone running on one machine is not equivalent to another instance on another node, nor would any two copies on the same node be equivalent.

Stateful clones are covered later in [Section 10.3, "Multi-state - Resources That Have Multiple Modes"](#).

Example 10.4. A clone of an LSB resource

```
<clone id="apache-clone">
  <meta_attributes id="apache-clone-meta">
    <nvpair id="apache-unique" name="globally-unique" value="false"/>
  </meta_attributes>
  <primitive id="apache" class="lsb" type="apache"/>
</clone>
```

10.2.1. Clone Properties

Table 10.2. Properties of a Clone Resource

Field	Description
id	A unique name for the clone

10.2.2. Clone Options

Options inherited from *primitive* resources: **priority**, **target-role**, **is-managed**

Table 10.3. Clone-specific configuration options

Field	Default	Description
clone-max	number of nodes in cluster	How many copies of the resource to start
clone-node-max	1	How many copies of the resource can be started on a single node
clone-min	1	Require at least this number of clone instances to be runnable before allowing resources depending on the clone to be runnable (<i>since 1.1.14</i>)
notify	true	When stopping or starting a copy of the clone, tell all the other copies beforehand and again when the action was successful. Allowed values: false , true
globally-unique	false	Does each copy of the clone perform a different function? Allowed values: false , true
ordered	false	Should the copies be started in series (instead of in parallel)? Allowed values: false , true
interleave	false	If this clone depends on another clone via an ordering constraint, is it allowed to start after the local instance of the other clone starts, rather than wait for all instances of the other clone to start? Allowed values: false , true

10.2.3. Clone Instance Attributes

Clones have no instance attributes; however, any that are set here will be inherited by the clone's children.

10.2.4. Clone Contents

Clones must contain exactly one primitive or group resource.



Warning

You should never reference the name of a clone's child. If you think you need to do this, you probably need to re-evaluate your design.

10.2.5. Clone Constraints

In most cases, a clone will have a single copy on each active cluster node. If this is not the case, you can indicate which nodes the cluster should preferentially assign copies to with resource location constraints. These constraints are written no differently from those for primitive resources except that the clone's **id** is used.

Example 10.5. Some constraints involving clones

```
<constraints>
  <rsc_location id="clone-prefers-node1" rsc="apache-clone" node="node1" score="500"/>
  <rsc_colocation id="stats-with-clone" rsc="apache-stats" with="apache-clone"/>
  <rsc_order id="start-clone-then-stats" first="apache-clone" then="apache-stats"/>
</constraints>
```

Ordering constraints behave slightly differently for clones. In the example above, **apache-stats** will wait until all copies of **apache-clone** that need to be started have done so before being started itself. Only if *no* copies can be started will **apache-stats** be prevented from being active. Additionally, the clone will wait for **apache-stats** to be stopped before stopping itself.

Colocation of a primitive or group resource with a clone means that the resource can run on any machine with an active copy of the clone. The cluster will choose a copy based on where the clone is running and the resource's own location preferences.

Colocation between clones is also possible. If one clone **A** is colocated with another clone **B**, the set of allowed locations for **A** is limited to nodes on which **B** is (or will be) active. Placement is then performed normally.

10.2.6. Clone Stickiness

To achieve a stable allocation pattern, clones are slightly sticky by default. If no value for **resource-stickiness** is provided, the clone will use a value of 1. Being a small value, it causes minimal disturbance to the score calculations of other resources but is enough to prevent Pacemaker from needlessly moving copies around the cluster.



Note

For globally unique clones, this may result in multiple instances of the clone staying on a single node, even after another eligible node becomes active (for example, after being put into standby mode then made active again). If you do not want this behavior, specify a **resource-stickiness** of 0 for the clone temporarily and let the cluster adjust, then set it back to 1 if you want the default behavior to apply again.

10.2.7. Clone Resource Agent Requirements

Any resource can be used as an anonymous clone, as it requires no additional support from the resource agent. Whether it makes sense to do so depends on your resource and its resource agent.

Globally unique clones do require some additional support in the resource agent. In particular, it must only respond with **OCF_SUCCESS** if the node has that exact instance active. All other probes for

instances of the clone should result in **OCF_NOT_RUNNING** (or one of the other OCF error codes if they are failed).

Individual instances of a clone are identified by appending a colon and a numerical offset, e.g. **apache:2**.

Resource agents can find out how many copies there are by examining the **OCF_RESKEY_CRM_meta_clone_max** environment variable and which copy it is by examining **OCF_RESKEY_CRM_meta_clone**.

The resource agent must not make any assumptions (based on **OCF_RESKEY_CRM_meta_clone**) about which numerical instances are active. In particular, the list of active copies will not always be an unbroken sequence, nor always start at 0.

10.2.7.1. Clone Notifications

Supporting notifications requires the **notify** action to be implemented. If supported, the notify action will be passed a number of extra variables which, when combined with additional context, can be used to calculate the current state of the cluster and what is about to happen to it.

Table 10.4. Environment variables supplied with Clone notify actions

Variable	Description
OCF_RESKEY_CRM_meta_notify_type	Allowed values: pre , post
OCF_RESKEY_CRM_meta_notify_operation	Allowed values: start , stop
OCF_RESKEY_CRM_meta_notify_start_resource	Resources to be started
OCF_RESKEY_CRM_meta_notify_stop_resource	Resources to be stopped
OCF_RESKEY_CRM_meta_notify_active_resource	Resources that are running
OCF_RESKEY_CRM_meta_notify_inactive_resource	Resources that are not running
OCF_RESKEY_CRM_meta_notify_start_uname	Nodes on which resources will be started
OCF_RESKEY_CRM_meta_notify_stop_uname	Nodes on which resources will be stopped
OCF_RESKEY_CRM_meta_notify_active_uname	Nodes on which resources are running
OCF_RESKEY_CRM_meta_notify_inactive_uname	Nodes on which resources are not running

The variables come in pairs, such as **OCF_RESKEY_CRM_meta_notify_start_resource** and **OCF_RESKEY_CRM_meta_notify_start_uname** and should be treated as an array of whitespace-separated elements.

Thus in order to indicate that **clone:0** will be started on **sles-1**, **clone:2** will be started on **sles-3**, and **clone:3** will be started on **sles-2**, the cluster would set

Example 10.6. Notification variables

```
OCF_RESKEY_CRM_meta_notify_start_resource="clone:0 clone:2 clone:3"
OCF_RESKEY_CRM_meta_notify_start_uname="sles-1 sles-3 sles-2"
```

10.2.7.2. Proper Interpretation of Notification Environment Variables

Pre-notification (stop):

- Active resources: **\$OCF_RESKEY_CRM_meta_notify_active_resource**
- Inactive resources: **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

Post-notification (stop) / Pre-notification (start):

- Active resources
 - **\$OCF_RESKEY_CRM_meta_notify_active_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Inactive resources
 - **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Resources that were started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources that were stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

Post-notification (start):

- Active resources:
 - **\$OCF_RESKEY_CRM_meta_notify_active_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Inactive resources:
 - **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources that were started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources that were stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

10.3. Multi-state - Resources That Have Multiple Modes

Multi-state resources are a specialization of clone resources; please ensure you understand [Section 10.2, “Clones - Resources That Get Active on Multiple Hosts”](#) before continuing!

Multi-state resources allow the instances to be in one of two operating modes (called *roles*). The roles are called *master* and *slave*, but can mean whatever you wish them to mean. The only limitation is that when an instance is started, it must come up in the slave role.

10.3.1. Multi-state Properties

Table 10.5. Properties of a Multi-State Resource

Field	Description
id	Your name for the multi-state resource

10.3.2. Multi-state Options

Options inherited from *primitive* resources: **priority**, **target-role**, **is-managed**

Options inherited from *clone* resources: **clone-max**, **clone-node-max**, **notify**, **globally-unique**, **ordered**, **interleave**

Table 10.6. Multi-state-specific resource configuration options

Field	Default	Description
master-max	1	How many copies of the resource can be promoted to the master role
master-node-max	1	How many copies of the resource can be promoted to the master role on a single node

10.3.3. Multi-state Instance Attributes

Multi-state resources have no instance attributes; however, any that are set here will be inherited by a master's children.

10.3.4. Multi-state Contents

Masters must contain exactly one primitive or group resource.



Warning

You should never reference the name of a master's child. If you think you need to do this, you probably need to re-evaluate your design.

10.3.5. Monitoring Multi-State Resources

The usual monitor actions are insufficient to monitor a multi-state resource, because pacemaker needs to verify not only that the resource is active, but also that its actual role matches its intended one.

Define two monitoring actions: the usual one will cover the slave role, and an additional one with **role="master"** will cover the master role.

Example 10.7. Monitoring both states of a multi-state resource

```
<master id="myMasterRsc">
  <primitive id="myRsc" class="ocf" type="myApp" provider="myCorp">
    <operations>
      <op id="public-ip-slave-check" name="monitor" interval="60"/>
      <op id="public-ip-master-check" name="monitor" interval="61" role="Master"/>
    </operations>
  </primitive>
```

```
</master>
```



Important

It is crucial that every monitor operation has a different interval! Pacemaker currently differentiates between operations only by resource and interval; so if (for example) a master/slave resource had the same monitor interval for both roles, Pacemaker would ignore the role when checking the status — which would cause unexpected return codes, and therefore unnecessary complications.

10.3.6. Multi-state Constraints

In most cases, multi-state resources will have a single copy on each active cluster node. If this is not the case, you can indicate which nodes the cluster should preferentially assign copies to with resource location constraints. These constraints are written no differently from those for primitive resources except that the master's **id** is used.

When considering multi-state resources in constraints, for most purposes it is sufficient to treat them as clones. The exception is that the **first-action** and/or **then-action** fields for ordering constraints may be set to **promote** or **demote** to constrain the master role, and colocation constraints may contain **rsc-role** and/or **with-rsc-role** fields.

Table 10.7. Additional colocation constraint options for multi-state resources

Field	Default	Description
rsc-role	Started	An additional attribute of colocation constraints that specifies the role that rsc must be in. Allowed values: Started, Master, Slave .
with-rsc-role	Started	An additional attribute of colocation constraints that specifies the role that with-rsc must be in. Allowed values: Started, Master, Slave .

Example 10.8. Constraints involving multi-state resources

```
<constraints>
  <rsc_location id="db-prefers-node1" rsc="database" node="node1" score="500"/>
  <rsc_colocation id="backup-with-db-slave" rsc="backup"
    with-rsc="database" with-rsc-role="Slave"/>
  <rsc_colocation id="myapp-with-db-master" rsc="myApp"
    with-rsc="database" with-rsc-role="Master"/>
  <rsc_order id="start-db-before-backup" first="database" then="backup"/>
  <rsc_order id="promote-db-then-app" first="database" first-action="promote"
    then="myApp" then-action="start"/>
</constraints>
```

In the example above, **myApp** will wait until one of the database copies has been started and promoted to master before being started itself on the same node. Only if no copies can be promoted will **myApp** be prevented from being active. Additionally, the cluster will wait for **myApp** to be stopped before demoting the database.

Colocation of a primitive or group resource with a multi-state resource means that it can run on any machine with an active copy of the multi-state resource that has the specified role (**master** or

slave). In the example above, the cluster will choose a location based on where database is running as a **master**, and if there are multiple **master** instances it will also factor in **myApp**'s own location preferences when deciding which location to choose.

Colocation with regular clones and other multi-state resources is also possible. In such cases, the set of allowed locations for the **rsc** clone is (after role filtering) limited to nodes on which the **with-rsc** multi-state resource is (or will be) in the specified role. Placement is then performed as normal.

10.3.6.1. Using Multi-state Resources in Colocation Sets

Table 10.8. Additional colocation set options relevant to multi-state resources

Field	Default	Description
role	Started	The role that <i>all members</i> of the set must be in. Allowed values: Started, Master, Slave .

In the following example **B**'s master must be located on the same node as **A**'s master. Additionally resources **C** and **D** must be located on the same node as **A**'s and **B**'s masters.

Example 10.9. Colocate C and D with A's and B's master instances

```
<constraints>
  <rsc_colocation id="coloc-1" score="INFINITY" >
    <resource_set id="colocated-set-example-1" sequential="true" role="Master">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="colocated-set-example-2" sequential="true">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
  </rsc_colocation>
</constraints>
```

10.3.6.2. Using Multi-state Resources in Ordering Sets

Table 10.9. Additional ordered set options relevant to multi-state resources

Field	Default	Description
action	value of first-action	An additional attribute of ordering constraint sets that specifies the action that applies to <i>all members</i> of the set. Allowed values: start, stop, promote, demote .

Example 10.10. Start C and D after first promoting A and B

```
<constraints>
  <rsc_order id="order-1" score="INFINITY" >
    <resource_set id="ordered-set-1" sequential="true" action="promote">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
    <resource_set id="ordered-set-2" sequential="true" action="start">
      <resource_ref id="C"/>
      <resource_ref id="D"/>
    </resource_set>
  </rsc_order>
```

```
</constraints>
```

In the above example, **B** cannot be promoted to a master role until **A** has been promoted. Additionally, resources **C** and **D** must wait until **A** and **B** have been promoted before they can start.

10.3.7. Multi-state Stickiness

As with regular clones, multi-state resources are slightly sticky by default. See [Section 10.2.6, “Clone Stickiness”](#) for details.

10.3.8. Which Resource Instance is Promoted

During the start operation, most resource agents should call the **crm_master** utility. This tool automatically detects both the resource and host and should be used to set a preference for being promoted. Based on this, **master-max**, and **master-node-max**, the instance(s) with the highest preference will be promoted.

An alternative is to create a location constraint that indicates which nodes are most preferred as masters.

Example 10.11. Explicitly preferring node1 to be promoted to master

```
<rsc_location id="master-location" rsc="myMasterRsc">
  <rule id="master-rule" score="100" role="Master">
    <expression id="master-exp" attribute="#uname" operation="eq" value="node1"/>
  </rule>
</rsc_location>
```

10.3.9. Requirements for Multi-state Resource Agents

Since multi-state resources are an extension of cloned resources, all the requirements for resource agents that support clones are also requirements for resource agents that support multi-state resources.

Additionally, multi-state resources require two extra actions, **demote** and **promote**, which are responsible for changing the state of the resource. Like **start** and **stop**, they should return **#{OCF_SUCCESS}** if they completed successfully or a relevant error code if they did not.

The states can mean whatever you wish, but when the resource is started, it must come up in the mode called **slave**. From there the cluster will decide which instances to promote to **master**.

In addition to the clone requirements for monitor actions, agents must also *accurately* report which state they are in. The cluster relies on the agent to report its status (including role) accurately and does not indicate to the agent what role it currently believes it to be in.

Table 10.10. Role implications of OCF return codes

Monitor Return Code	Description
OCF_NOT_RUNNING	Stopped
OCF_SUCCESS	Running (Slave)
OCF_RUNNING_MASTER	Running (Master)
OCF_FAILED_MASTER	Failed (Master)
Other	Failed (Slave)

10.3.9.1. Multi-state Notifications

Like clones, supporting notifications requires the **notify** action to be implemented. If supported, the notify action will be passed a number of extra variables which, when combined with additional context, can be used to calculate the current state of the cluster and what is about to happen to it.

Table 10.11. Environment variables supplied with multi-state notify actions ¹

Variable	Description
OCF_RESKEY_CRM_meta_notify_type	Allowed values: pre , post
OCF_RESKEY_CRM_meta_notify_operation	Allowed values: start , stop
OCF_RESKEY_CRM_meta_notify_active_resource	Resources the that are running
OCF_RESKEY_CRM_meta_notify_inactive_resource	Resources the that are not running
<i>OCF_RESKEY_CRM_meta_notify_master_resource</i>	Resources that are running in Master mode
<i>OCF_RESKEY_CRM_meta_notify_slave_resource</i>	Resources that are running in Slave mode
OCF_RESKEY_CRM_meta_notify_start_resource	Resources to be started
OCF_RESKEY_CRM_meta_notify_stop_resource	Resources to be stopped
<i>OCF_RESKEY_CRM_meta_notify_promote_resource</i>	Resources to be promoted
<i>OCF_RESKEY_CRM_meta_notify_demote_resource</i>	Resources to be demoted
OCF_RESKEY_CRM_meta_notify_start_uname	Nodes on which resources will be started
OCF_RESKEY_CRM_meta_notify_stop_uname	Nodes on which resources will be stopped
<i>OCF_RESKEY_CRM_meta_notify_promote_uname</i>	Nodes on which resources will be promote
<i>OCF_RESKEY_CRM_meta_notify_demote_uname</i>	Nodes on which resources will be demoted
OCF_RESKEY_CRM_meta_notify_active_uname	Nodes on which resources are running
OCF_RESKEY_CRM_meta_notify_inactive_uname	Nodes on which resources are not running
<i>OCF_RESKEY_CRM_meta_notify_master_uname</i>	Nodes on which resources are running in Master mode
<i>OCF_RESKEY_CRM_meta_notify_slave_uname</i>	Nodes on which resources are running in Slave mode

¹ Emphasized variables are specific to **Master** resources, and all behave in the same manner as described for Clone resources.

10.3.9.2. Proper Interpretation of Multi-state Notification Environment Variables

Pre-notification (demote):

- **Active** resources: **\$OCF_RESKEY_CRM_meta_notify_active_resource**
- **Master** resources: **\$OCF_RESKEY_CRM_meta_notify_master_resource**

- **Slave** resources: **\$OCF_RESKEY_CRM_meta_notify_slave_resource**
- Inactive resources: **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources to be demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

[Post-notification \(demote\) / Pre-notification \(stop\):](#)

- **Active** resources: **\$OCF_RESKEY_CRM_meta_notify_active_resource**
- **Master** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_master_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- **Slave** resources: **\$OCF_RESKEY_CRM_meta_notify_slave_resource**
- Inactive resources: **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources to be demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Resources that were demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**

[Post-notification \(stop\) / Pre-notification \(start\)](#)

- **Active** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_active_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- **Master** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_master_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- **Slave** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_slave_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Inactive resources:
 - **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources to be demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Resources that were demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources that were stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

Post-notification (start) / Pre-notification (promote)

- **Active** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_active_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- **Master** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_master_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- **Slave** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_slave_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Inactive resources:
 - **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources to be demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Resources that were started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources that were demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources that were stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

Post-notification (promote)

- **Active** resources:

- **\$OCF_RESKEY_CRM_meta_notify_active_resource**
- minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- plus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- **Master** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_master_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- **Slave** resources:
 - **\$OCF_RESKEY_CRM_meta_notify_slave_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Inactive resources:
 - **\$OCF_RESKEY_CRM_meta_notify_inactive_resource**
 - plus **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
 - minus **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources to be promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources to be demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources to be stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**
- Resources that were started: **\$OCF_RESKEY_CRM_meta_notify_start_resource**
- Resources that were promoted: **\$OCF_RESKEY_CRM_meta_notify_promote_resource**
- Resources that were demoted: **\$OCF_RESKEY_CRM_meta_notify_demote_resource**
- Resources that were stopped: **\$OCF_RESKEY_CRM_meta_notify_stop_resource**

Utilization and Placement Strategy

Table of Contents

11.1. Utilization attributes	97
11.2. Placement Strategy	98
11.3. Allocation Details	99
11.3.1. Which node is preferred to get consumed first when allocating resources?	99
11.3.2. Which node has more free capacity?	99
11.3.3. Which resource is preferred to be assigned first?	99
11.4. Limitations and Workarounds	100

Pacemaker decides where to place a resource according to the resource allocation scores on every node. The resource will be allocated to the node where the resource has the highest score.

If the resource allocation scores on all the nodes are equal, by the default placement strategy, Pacemaker will choose a node with the least number of allocated resources for balancing the load. If the number of resources on each node is equal, the first eligible node listed in the CIB will be chosen to run the resource.

Often, in real-world situations, different resources use significantly different proportions of a node's capacities (memory, I/O, etc.). We cannot balance the load ideally just according to the number of resources allocated to a node. Besides, if resources are placed such that their combined requirements exceed the provided capacity, they may fail to start completely or run with degraded performance.

To take these factors into account, Pacemaker allows you to configure:

1. The capacity a certain node provides.
2. The capacity a certain resource requires.
3. An overall strategy for placement of resources.

11.1. Utilization attributes

To configure the capacity that a node provides or a resource requires, you can use *utilization attributes* in **node** and **resource** objects. You can name utilization attributes according to your preferences and define as many name/value pairs as your configuration needs. However, the attributes' values must be integers.

Example 11.1. Specifying CPU and RAM capacities of two nodes

```
<node id="node1" type="normal" uname="node1">
  <utilization id="node1-utilization">
    <nvpair id="node1-utilization-cpu" name="cpu" value="2"/>
    <nvpair id="node1-utilization-memory" name="memory" value="2048"/>
  </utilization>
</node>
<node id="node2" type="normal" uname="node2">
  <utilization id="node2-utilization">
    <nvpair id="node2-utilization-cpu" name="cpu" value="4"/>
    <nvpair id="node2-utilization-memory" name="memory" value="4096"/>
  </utilization>
</node>
```

Example 11.2. Specifying CPU and RAM consumed by several resources

```
<primitive id="rsc-small" class="ocf" provider="pacemaker" type="Dummy">
  <utilization id="rsc-small-utilization">
    <nvpair id="rsc-small-utilization-cpu" name="cpu" value="1"/>
    <nvpair id="rsc-small-utilization-memory" name="memory" value="1024"/>
  </utilization>
</primitive>
<primitive id="rsc-medium" class="ocf" provider="pacemaker" type="Dummy">
  <utilization id="rsc-medium-utilization">
    <nvpair id="rsc-medium-utilization-cpu" name="cpu" value="2"/>
    <nvpair id="rsc-medium-utilization-memory" name="memory" value="2048"/>
  </utilization>
</primitive>
<primitive id="rsc-large" class="ocf" provider="pacemaker" type="Dummy">
  <utilization id="rsc-large-utilization">
    <nvpair id="rsc-large-utilization-cpu" name="cpu" value="3"/>
    <nvpair id="rsc-large-utilization-memory" name="memory" value="3072"/>
  </utilization>
</primitive>
```

A node is considered eligible for a resource if it has sufficient free capacity to satisfy the resource's requirements. The nature of the required or provided capacities is completely irrelevant to Pacemaker — it just makes sure that all capacity requirements of a resource are satisfied before placing a resource to a node.

11.2. Placement Strategy

After you have configured the capacities your nodes provide and the capacities your resources require, you need to set the **placement-strategy** in the global cluster options, otherwise the capacity configurations have *no effect*.

Four values are available for the **placement-strategy**:

default

Utilization values are not taken into account at all. Resources are allocated according to allocation scores. If scores are equal, resources are evenly distributed across nodes.

utilization

Utilization values are taken into account *only* when deciding whether a node is considered eligible (i.e. whether it has sufficient free capacity to satisfy the resource's requirements). Load-balancing is still done based on the number of resources allocated to a node.

balanced

Utilization values are taken into account when deciding whether a node is eligible to serve a resource *and* when load-balancing, so an attempt is made to spread the resources in a way that optimizes resource performance.

minimal

Utilization values are taken into account *only* when deciding whether a node is eligible to serve a resource. For load-balancing, an attempt is made to concentrate the resources on as few nodes as possible, thereby enabling possible power savings on the remaining nodes.

Set **placement-strategy** with **crm_attribute**:

```
# crm_attribute --name placement-strategy --update balanced
```

Now Pacemaker will ensure the load from your resources will be distributed evenly throughout the cluster, without the need for convoluted sets of colocation constraints.

11.3. Allocation Details

11.3.1. Which node is preferred to get consumed first when allocating resources?

- The node with the highest node weight gets consumed first. Node weight is a score maintained by the cluster to represent node health.
- If multiple nodes have the same node weight:
 - If **placement-strategy** is **default** or **utilization**, the node that has the least number of allocated resources gets consumed first.
 - If their numbers of allocated resources are equal, the first eligible node listed in the CIB gets consumed first.
 - If **placement-strategy** is **balanced**, the node that has the most free capacity gets consumed first.
 - If the free capacities of the nodes are equal, the node that has the least number of allocated resources gets consumed first.
 - If their numbers of allocated resources are equal, the first eligible node listed in the CIB gets consumed first.
 - If **placement-strategy** is **minimal**, the first eligible node listed in the CIB gets consumed first.

11.3.2. Which node has more free capacity?

If only one type of utilization attribute has been defined, free capacity is a simple numeric comparison.

If multiple types of utilization attributes have been defined, then the node that is numerically highest in the the most attribute types has the most free capacity. For example:

- If **nodeA** has more free **cpus**, and **nodeB** has more free **memory**, then their free capacities are equal.
- If **nodeA** has more free **cpus**, while **nodeB** has more free **memory** and **storage**, then **nodeB** has more free capacity.

11.3.3. Which resource is preferred to be assigned first?

- The resource that has the highest **priority** (see [Section 5.4, “Resource Options”](#)) gets allocated first.
- If their priorities are equal, check whether they are already running. The resource that has the highest score on the node where it's running gets allocated first, to prevent resource shuffling.
- If the scores above are equal or the resources are not running, the resource has the highest score on the preferred node gets allocated first.
- If the scores above are equal, the first runnable resource listed in the CIB gets allocated first.

11.4. Limitations and Workarounds

The type of problem Pacemaker is dealing with here is known as the *knapsack problem*¹ and falls into the *NP-complete*² category of computer science problems — a fancy way of saying "it takes a really long time to solve".

Clearly in a HA cluster, it's not acceptable to spend minutes, let alone hours or days, finding an optional solution while services remain unavailable.

So instead of trying to solve the problem completely, Pacemaker uses a *best effort* algorithm for determining which node should host a particular service. This means it arrives at a solution much faster than traditional linear programming algorithms, but by doing so at the price of leaving some services stopped.

In the contrived example at the start of this section:

- **rsc-small** would be allocated to **node1**
- **rsc-medium** would be allocated to **node2**
- **rsc-large** would remain inactive

Which is not ideal.

There are various approaches to dealing with the limitations of pacemaker's placement strategy:

Ensure you have sufficient physical capacity.

It might sound obvious, but if the physical capacity of your nodes is (close to) maxed out by the cluster under normal conditions, then failover isn't going to go well. Even without the utilization feature, you'll start hitting timeouts and getting secondary failures.

Build some buffer into the capabilities advertised by the nodes.

Advertise slightly more resources than we physically have, on the (usually valid) assumption that a resource will not use 100% of the configured amount of CPU, memory and so forth *all* the time. This practice is sometimes called *overcommit*.

Specify resource priorities.

If the cluster is going to sacrifice services, it should be the ones you care about (comparatively) the least. Ensure that resource priorities are properly set so that your most important resources are scheduled first.

¹ http://en.wikipedia.org/wiki/Knapsack_problem

² <http://en.wikipedia.org/wiki/NP-complete>

Resource Templates

Table of Contents

12.1. Configuring Resources with Templates	101
12.2. Referencing Templates in Constraints	103
12.2.1. Referencing Resource Templates in Sequential Resource Sets	103
12.2.2. Referencing Resource Templates in Parallel Resource Sets	104

If you want to create lots of resources with similar configurations, defining a resource template simplifies the task. Once defined, it can be referenced in primitives or in certain types of constraints.

12.1. Configuring Resources with Templates

The primitives referencing the template will inherit all meta-attributes, instance attributes, utilization attributes and operations defined in the template. And you can define specific attributes and operations for any of the primitives. If any of these are defined in both the template and the primitive, the values defined in the primitive will take precedence over the ones defined in the template.

Hence, resource templates help to reduce the amount of configuration work. If any changes are needed, they can be done to the template definition and will take effect globally in all resource definitions referencing that template.

Resource templates have a syntax similar to that of primitives.

Example 12.1. Resource template for a migratable Xen virtual machine

```
<template id="vm-template" class="ocf" provider="heartbeat" type="Xen">
  <meta_attributes id="vm-template-meta_attributes">
    <nvpair id="vm-template-meta_attributes-allow-migrate" name="allow-migrate"
value="true"/>
  </meta_attributes>
  <utilization id="vm-template-utilization">
    <nvpair id="vm-template-utilization-memory" name="memory" value="512"/>
  </utilization>
  <operations>
    <op id="vm-template-monitor-15s" interval="15s" name="monitor" timeout="60s"/>
    <op id="vm-template-start-0" interval="0" name="start" timeout="60s"/>
  </operations>
</template>
```

Once you define a resource template, you can use it in primitives by specifying the **template** property.

Example 12.2. Xen primitive resource using a resource template

```
<primitive id="vm1" template="vm-template">
  <instance_attributes id="vm1-instance_attributes">
    <nvpair id="vm1-instance_attributes-name" name="name" value="vm1"/>
    <nvpair id="vm1-instance_attributes-xmfile" name="xmfile" value="/etc/xen/shared-vm/
vm1"/>
  </instance_attributes>
</primitive>
```

In the example above, the new primitive **vm1** will inherit everything from **vm-template**. For example, the equivalent of the above two examples would be:

Example 12.3. Equivalent Xen primitive resource not using a resource template

```
<primitive id="vm1" class="ocf" provider="heartbeat" type="Xen">
  <meta_attributes id="vm-template-meta_attributes">
    <nvpair id="vm-template-meta_attributes-allow-migrate" name="allow-migrate"
value="true"/>
  </meta_attributes>
  <utilization id="vm-template-utilization">
    <nvpair id="vm-template-utilization-memory" name="memory" value="512"/>
  </utilization>
  <operations>
    <op id="vm-template-monitor-15s" interval="15s" name="monitor" timeout="60s"/>
    <op id="vm-template-start-0" interval="0" name="start" timeout="60s"/>
  </operations>
  <instance_attributes id="vm1-instance_attributes">
    <nvpair id="vm1-instance_attributes-name" name="name" value="vm1"/>
    <nvpair id="vm1-instance_attributes-xmfile" name="xmfile" value="/etc/xen/shared-vm/
vm1"/>
  </instance_attributes>
</primitive>
```

If you want to overwrite some attributes or operations, add them to the particular primitive's definition.

Example 12.4. Xen resource overriding template values

```
<primitive id="vm2" template="vm-template">
  <meta_attributes id="vm2-meta_attributes">
    <nvpair id="vm2-meta_attributes-allow-migrate" name="allow-migrate" value="false"/>
  </meta_attributes>
  <utilization id="vm2-utilization">
    <nvpair id="vm2-utilization-memory" name="memory" value="1024"/>
  </utilization>
  <instance_attributes id="vm2-instance_attributes">
    <nvpair id="vm2-instance_attributes-name" name="name" value="vm2"/>
    <nvpair id="vm2-instance_attributes-xmfile" name="xmfile" value="/etc/xen/shared-vm/
vm2"/>
  </instance_attributes>
  <operations>
    <op id="vm2-monitor-30s" interval="30s" name="monitor" timeout="120s"/>
    <op id="vm2-stop-0" interval="0" name="stop" timeout="60s"/>
  </operations>
</primitive>
```

In the example above, the new primitive **vm2** has special attribute values. Its **monitor** operation has a longer **timeout** and **interval**, and the primitive has an additional **stop** operation.

To see the resulting definition of a resource, run:

```
# crm_resource --query-xml --resource vm2
```

To see the raw definition of a resource in the CIB, run:

```
# crm_resource --query-xml-raw --resource vm2
```

12.2. Referencing Templates in Constraints

A resource template can be referenced in the following types of constraints:

- **order** constraints (see [Section 6.3, “Specifying the Order in which Resources Should Start/Stop”](#))
- **colocation** constraints (see [Section 6.4, “Placing Resources Relative to other Resources”](#))
- **rsc_ticket** constraints (for multi-site clusters as described in [Section 15.3, “Configuring Ticket Dependencies”](#))

Resource templates referenced in constraints stand for all primitives which are derived from that template. This means, the constraint applies to all primitive resources referencing the resource template. Referencing resource templates in constraints is an alternative to resource sets and can simplify the cluster configuration considerably.

For example, given the example templates earlier in this section:

```
<rsc_colocation id="vm-template-colo-base-rsc" rsc="vm-template" rsc-role="Started" with-
rsc="base-rsc" score="INFINITY"/>
```

would colocate all VMs with **base-rsc** and is the equivalent of the following constraint configuration:

```
<rsc_colocation id="vm-colo-base-rsc" score="INFINITY">
  <resource_set id="vm-colo-base-rsc-0" sequential="false" role="Started">
    <resource_ref id="vm1"/>
    <resource_ref id="vm2"/>
  </resource_set>
  <resource_set id="vm-colo-base-rsc-1">
    <resource_ref id="base-rsc"/>
  </resource_set>
</rsc_colocation>
```



Note

In a colocation constraint, only one template may be referenced from either **rsc** or **with-rsc**; the other reference must be a regular resource.

12.2.1. Referencing Resource Templates in Sequential Resource Sets

Resource templates can also be referenced in resource sets.

For example:

```
<rsc_order id="order1" score="INFINITY">
  <resource_set id="order1-0">
    <resource_ref id="base-rsc"/>
    <resource_ref id="vm-template"/>
    <resource_ref id="top-rsc"/>
  </resource_set>
</rsc_order>
```

is the equivalent of the following constraint configuration:

```
<rsc_order id="order1" score="INFINITY">
  <resource_set id="order1-0">
    <resource_ref id="base-rsc"/>
    <resource_ref id="vm1"/>
    <resource_ref id="vm2"/>
    <resource_ref id="top-rsc"/>
  </resource_set>
</rsc_order>
```

12.2.2. Referencing Resource Templates in Parallel Resource Sets

If the resources referencing the template can run in parallel:

```
<rsc_order id="order2" score="INFINITY">
  <resource_set id="order2-0">
    <resource_ref id="base-rsc"/>
  </resource_set>
  <resource_set id="order2-1" sequential="false">
    <resource_ref id="vm-template"/>
  </resource_set>
  <resource_set id="order2-2">
    <resource_ref id="top-rsc"/>
  </resource_set>
</rsc_order>
```

is the equivalent of the following constraint configuration:

```
<rsc_order id="order2" score="INFINITY">
  <resource_set id="order2-0">
    <resource_ref id="base-rsc"/>
  </resource_set>
  <resource_set id="order2-1" sequential="false">
    <resource_ref id="vm1"/>
    <resource_ref id="vm2"/>
  </resource_set>
  <resource_set id="order2-2">
    <resource_ref id="top-rsc"/>
  </resource_set>
</rsc_order>
```

STONITH

Table of Contents

13.1. What Is STONITH?	105
13.2. What STONITH Device Should You Use?	105
13.3. Special Treatment of STONITH Resources	105
13.4. Configuring STONITH	110
13.4.1. Example STONITH Configuration	111
13.5. Advanced STONITH Configurations	113
13.5.1. Example Dual-Layer, Dual-Device Fencing Topologies	114
13.6. Remapping Reboots	119

13.1. What Is STONITH?

STONITH (an acronym for "Shoot The Other Node In The Head"), also called *fencing*, protects your data from being corrupted by rogue nodes or concurrent access.

Just because a node is unresponsive, this doesn't mean it isn't accessing your data. The only way to be 100% sure that your data is safe, is to use STONITH so we can be certain that the node is truly offline, before allowing the data to be accessed from another node.

STONITH also has a role to play in the event that a clustered service cannot be stopped. In this case, the cluster uses STONITH to force the whole node offline, thereby making it safe to start the service elsewhere.

13.2. What STONITH Device Should You Use?

It is crucial that the STONITH device can allow the cluster to differentiate between a node failure and a network one.

The biggest mistake people make in choosing a STONITH device is to use a remote power switch (such as many on-board IPMI controllers) that shares power with the node it controls. In such cases, the cluster cannot be sure if the node is really offline, or active and suffering from a network fault.

Likewise, any device that relies on the machine being active (such as SSH-based "devices" used during testing) are inappropriate.

13.3. Special Treatment of STONITH Resources

STONITH resources are somewhat special in Pacemaker.

STONITH may be initiated by pacemaker or by other parts of the cluster (such as resources like DRBD or DLM). To accommodate this, pacemaker does not require the STONITH resource to be in the *started* state in order to be used, thus allowing reliable use of STONITH devices in such a case.

**Note**

In pacemaker versions 1.1.9 and earlier, this feature either did not exist or did not work well. Only "running" STONITH resources could be used by Pacemaker for fencing, and if another component tried to fence a node while Pacemaker was moving STONITH resources, the fencing could fail.

All nodes have access to STONITH devices' definitions and instantiate them on-the-fly when needed, but preference is given to *verified* instances, which are the ones that are *started* according to the cluster's knowledge.

In the case of a cluster split, the partition with a verified instance will have a slight advantage, because the STONITH daemon in the other partition will have to hear from all its current peers before choosing a node to perform the fencing.

Fencing resources do work the same as regular resources in some respects:

- **target-role** can be used to enable or disable the resource
- Location constraints can be used to prevent a specific node from using the resource

**Important**

Currently there is a limitation that fencing resources may only have one set of meta-attributes and one set of instance attributes. This can be revisited if it becomes a significant limitation for people.

See the table below or run **man stonithd** to see special instance attributes that may be set for any fencing resource, regardless of fence agent.

Table 13.1. Properties of Fencing Resources

Field	Type	Default	Description
stonith-timeout	NA	NA	Older versions used this to override the default period to wait for a STONITH (reboot, on, off) action to complete for this device. It has been replaced by the pcmk_reboot_timeout and pcmk_off_timeout properties.
priority	integer	0	The priority of the STONITH resource. Devices are tried in order of highest priority to lowest.
pcmk_host_map	string		A mapping of host names to ports numbers for devices that do not support host names. Example: node1:1;node2:2,3 tells the cluster to use port 1 for node1 and ports 2 and 3 for node2 .

Field	Type	Default	Description
pcmk_host_list	string		A list of machines controlled by this device (optional unless pcmk_host_check is static-list).
pcmk_host_check	string	dynamic-list	How to determine which machines are controlled by the device. Allowed values: <ul style="list-style-type: none"> • dynamic-list: query the device • static-list: check the pcmk_host_list attribute • none: assume every device can fence every machine
pcmk_delay_max	time	0s	Enable a random delay of up to the time specified before executing stonith actions. This is sometimes used in two-node clusters to ensure that the nodes don't fence each other at the same time.
pcmk_action_limit	integer	1	The maximum number of actions that can be performed in parallel on this device, if the cluster option concurrent-fencing is true . -1 is unlimited.
pcmk_host_argument	string	port	<i>Advanced use only.</i> Which parameter should be supplied to the resource agent to identify the node to be fenced. Some devices do not support the standard port parameter or may provide additional ones. Use this to specify an alternate, device-specific parameter. A value of none tells the cluster not to supply any additional parameters.
pcmk_reboot_action	string	reboot	<i>Advanced use only.</i> The command to send to the resource agent in order to reboot a node. Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific command.
pcmk_reboot_timeout	time	60s	<i>Advanced use only.</i> Specify an alternate timeout to use for reboot actions instead of the value of stonith-timeout . Some devices need much more or less time to complete than normal. Use this to specify an alternate, device-specific timeout.
pcmk_reboot_retries	integer	2	<i>Advanced use only.</i> The maximum number of times to retry the reboot command within the timeout period. Some devices do not support

Field	Type	Default	Description
			multiple connections, and operations may fail if the device is busy with another task, so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries before giving up.
pcmk_off_action	string	off	<i>Advanced use only.</i> The command to send to the resource agent in order to shut down a node. Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific command.
pcmk_off_timeout	time	60s	<i>Advanced use only.</i> Specify an alternate timeout to use for off actions instead of the value of stonith-timeout . Some devices need much more or less time to complete than normal. Use this to specify an alternate, device-specific timeout.
pcmk_off_retries	integer	2	<i>Advanced use only.</i> The maximum number of times to retry the off command within the timeout period. Some devices do not support multiple connections, and operations may fail if the device is busy with another task, so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries before giving up.
pcmk_list_action	string	list	<i>Advanced use only.</i> The command to send to the resource agent in order to list nodes. Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific command.
pcmk_list_timeout	time	60s	<i>Advanced use only.</i> Specify an alternate timeout to use for list actions instead of the value of stonith-timeout . Some devices need much more or less time to complete than normal. Use this to specify an alternate, device-specific timeout.
pcmk_list_retries	integer	2	<i>Advanced use only.</i> The maximum number of times to retry the list command within the timeout period. Some devices do not support multiple connections, and operations may fail if the device is busy with another task, so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries before giving up.
pcmk_monitor_action	string	monitor	<i>Advanced use only.</i> The command to send to the resource agent in order to report extended

Field	Type	Default	Description
			status. Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific command.
pcmk_monitor_timeout	time	60s	<i>Advanced use only.</i> Specify an alternate timeout to use for monitor actions instead of the value of stonith-timeout . Some devices need much more or less time to complete than normal. Use this to specify an alternate, device-specific timeout.
pcmk_monitor_retries	integer	2	<i>Advanced use only.</i> The maximum number of times to retry the monitor command within the timeout period. Some devices do not support multiple connections, and operations may fail if the device is busy with another task, so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries before giving up.
pcmk_status_action	string	status	<i>Advanced use only.</i> The command to send to the resource agent in order to report status. Some devices do not support the standard commands or may provide additional ones. Use this to specify an alternate, device-specific command.
pcmk_status_timeout	time	60s	<i>Advanced use only.</i> Specify an alternate timeout to use for status actions instead of the value of stonith-timeout . Some devices need much more or less time to complete than normal. Use this to specify an alternate, device-specific timeout.
pcmk_status_retries	integer	2	<i>Advanced use only.</i> The maximum number of times to retry the status command within the timeout period. Some devices do not support multiple connections, and operations may fail if the device is busy with another task, so Pacemaker will automatically retry the operation, if there is time remaining. Use this option to alter the number of times Pacemaker retries before giving up.

13.4. Configuring STONITH



Note

Higher-level configuration shells include functionality to simplify the process below, particularly the step for deciding which parameters are required. However since this document deals only with core components, you should refer to the STONITH section of the [Clusters from Scratch](http://www.clusterlabs.org/doc/Clusters from Scratch)¹ guide for those details.

1. Find the correct driver:

```
# stonith_admin --list-installed
```

2. Find the required parameters associated with the device (replacing \$AGENT_NAME with the name obtained from the previous step):

```
# stonith_admin --metadata --agent $AGENT_NAME
```

3. Create a file called **stonith.xml** containing a primitive resource with a class of **stonith**, a type equal to the agent name obtained earlier, and a parameter for each of the values returned in the previous step.
4. If the device does not know how to fence nodes based on their uname, you may also need to set the special **pcmk_host_map** parameter. See **man stonithd** for details.
5. If the device does not support the **list** command, you may also need to set the special **pcmk_host_list** and/or **pcmk_host_check** parameters. See **man stonithd** for details.
6. If the device does not expect the victim to be specified with the **port** parameter, you may also need to set the special **pcmk_host_argument** parameter. See **man stonithd** for details.
7. Upload it into the CIB using cibadmin:

```
# cibadmin -C -o resources --xml-file stonith.xml
```

8. Set **stonith-enabled** to true:

```
# crm_attribute -t crm_config -n stonith-enabled -v true
```

9. Once the stonith resource is running, you can test it by executing the following (although you might want to stop the cluster on that machine first):

```
# stonith_admin --reboot nodename
```

¹ <http://www.clusterlabs.org/doc/>

13.4.1. Example STONITH Configuration

Assume we have a chassis containing four nodes and an IPMI device active on 192.0.2.1. We would choose the **fence_ipmilan** driver, and obtain the following list of parameters:

Example 13.1. Obtaining a list of STONITH Parameters

```
# stonith_admin --metadata -a fence_ipmilan
```

```
<resource-agent name="fence_ipmilan" shortdesc="Fence agent for IPMI over LAN">
  <symlink name="fence_ilo3" shortdesc="Fence agent for HP iLO3"/>
  <symlink name="fence_ilo4" shortdesc="Fence agent for HP iLO4"/>
  <symlink name="fence_idrac" shortdesc="Fence agent for Dell iDRAC"/>
  <symlink name="fence_imm" shortdesc="Fence agent for IBM Integrated Management Module"/>
  <longdesc>
  </longdesc>
  <vendor-url>
  </vendor-url>
  <parameters>
    <parameter name="auth" unique="0" required="0">
      <getopt mixed="-A"/>
      <content type="string"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="ipaddr" unique="0" required="1">
      <getopt mixed="-a"/>
      <content type="string"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="passwd" unique="0" required="0">
      <getopt mixed="-p"/>
      <content type="string"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="passwd_script" unique="0" required="0">
      <getopt mixed="-S"/>
      <content type="string"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="lanplus" unique="0" required="0">
      <getopt mixed="-P"/>
      <content type="boolean"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="login" unique="0" required="0">
      <getopt mixed="-l"/>
      <content type="string"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="action" unique="0" required="0">
      <getopt mixed="-o"/>
      <content type="string" default="reboot"/>
      <shortdesc>
      </shortdesc>
    </parameter>
    <parameter name="timeout" unique="0" required="0">
      <getopt mixed="-t"/>
      <content type="string"/>
      <shortdesc>
```

```

    </shortdesc>
</parameter>
<parameter name="cipher" unique="0" required="0">
  <getopt mixed="-C"/>
  <content type="string"/>
  <shortdesc>
  </shortdesc>
</parameter>
<parameter name="method" unique="0" required="0">
  <getopt mixed="-M"/>
  <content type="string" default="onoff"/>
  <shortdesc>
  </shortdesc>
</parameter>
<parameter name="power_wait" unique="0" required="0">
  <getopt mixed="-T"/>
  <content type="string" default="2"/>
  <shortdesc>
  </shortdesc>
</parameter>
<parameter name="delay" unique="0" required="0">
  <getopt mixed="-f"/>
  <content type="string"/>
  <shortdesc>
  </shortdesc>
</parameter>
<parameter name="privlvl" unique="0" required="0">
  <getopt mixed="-L"/>
  <content type="string"/>
  <shortdesc>
  </shortdesc>
</parameter>
<parameter name="verbose" unique="0" required="0">
  <getopt mixed="-v"/>
  <content type="boolean"/>
  <shortdesc>
  </shortdesc>
</parameter>
</parameters>
<actions>
  <action name="on"/>
  <action name="off"/>
  <action name="reboot"/>
  <action name="status"/>
  <action name="diag"/>
  <action name="list"/>
  <action name="monitor"/>
  <action name="metadata"/>
  <action name="stop" timeout="20s"/>
  <action name="start" timeout="20s"/>
</actions>
</resource-agent>

```

Based on that, we would create a STONITH resource fragment that might look like this:

Example 13.2. An IPMI-based STONITH Resource

```

<primitive id="Fencing" class="stonith" type="fence_ipmilan" >
  <instance_attributes id="Fencing-params" >
    <nvpair id="Fencing-passwd" name="passwd" value="testuser" />
    <nvpair id="Fencing-login" name="login" value="abc123" />
    <nvpair id="Fencing-ipaddr" name="ipaddr" value="192.0.2.1" />
    <nvpair id="Fencing-pcmk_host_list" name="pcmk_host_list" value="pcmk-1 pcmk-2" />
  </instance_attributes>

```

```
<operations >
  <op id="Fencing-monitor-10m" interval="10m" name="monitor" timeout="300s" />
</operations>
</primitive>
```

Finally, we need to enable STONITH:

```
# crm_attribute -t crm_config -n stonith-enabled -v true
```

13.5. Advanced STONITH Configurations

Some people consider that having one fencing device is a single point of failure²; others prefer removing the node from the storage and network instead of turning it off.

Whatever the reason, Pacemaker supports fencing nodes with multiple devices through a feature called *fencing topologies*.

Simply create the individual devices as you normally would, then define one or more **fencing-level** entries in the **fencing-topology** section of the configuration.

- Each fencing level is attempted in order of ascending **index**. Allowed indexes are 0 to 9.
- If a device fails, processing terminates for the current level. No further devices in that level are exercised, and the next level is attempted instead.
- If the operation succeeds for all the listed devices in a level, the level is deemed to have passed.
- The operation is finished when a level has passed (success), or all levels have been attempted (failed).
- If the operation failed, the next step is determined by the Policy Engine and/or **crmd**.

Some possible uses of topologies include:

- Try poison-pill and fail back to power
- Try disk and network, and fail back to power if either fails
- Initiate a kdump and then poweroff the node

Table 13.2. Properties of Fencing Levels

Field	Description
id	A unique name for the level
target	The name of a single node to which this level applies
target-pattern	A regular expression matching the names of nodes to which this level applies (<i>since 1.1.14</i>)
target-attribute	The name of a node attribute that is set (to target-value) for nodes to which this level applies (<i>since 1.1.14</i>)
target-value	The node attribute value (of target-attribute) that is set for nodes to which this level applies (<i>since 1.1.14</i>)

² Not true, since a node or resource must fail before fencing even has a chance to

Field	Description
index	The order in which to attempt the levels. Levels are attempted in ascending order <i>until one succeeds</i> .
devices	A comma-separated list of devices that must all be tried for this level

Example 13.3. Fencing topology with different devices for different nodes

```
<cib crm_feature_set="3.0.6" validate-with="pacemaker-1.2" admin_epoch="1" epoch="0"
num_updates="0">
  <configuration>
    ...
    <fencing-topology>
      <!-- For pcmk-1, try poison-pill and fail back to power -->
      <fencing-level id="f-p1.1" target="pcmk-1" index="1" devices="poison-pill"/>
      <fencing-level id="f-p1.2" target="pcmk-1" index="2" devices="power"/>

      <!-- For pcmk-2, try disk and network, and fail back to power -->
      <fencing-level id="f-p2.1" target="pcmk-2" index="1" devices="disk,network"/>
      <fencing-level id="f-p2.2" target="pcmk-2" index="2" devices="power"/>
    </fencing-topology>
    ...
  </configuration>
  <status/>
</cib>
```

13.5.1. Example Dual-Layer, Dual-Device Fencing Topologies

The following example illustrates an advanced use of **fencing-topology** in a cluster with the following properties:

- 3 nodes (2 active prod-mysql nodes, 1 prod_mysql-rep in standby for quorum purposes)
- the active nodes have an IPMI-controlled power board reached at 192.0.2.1 and 192.0.2.2
- the active nodes also have two independent PSUs (Power Supply Units) connected to two independent PDUs (Power Distribution Units) reached at 198.51.100.1 (port 10 and port 11) and 203.0.113.1 (port 10 and port 11)
- the first fencing method uses the **fence_ipmi** agent
- the second fencing method uses the **fence_apc_snmp** agent targetting 2 fencing devices (one per PSU, either port 10 or 11)
- fencing is only implemented for the active nodes and has location constraints
- fencing topology is set to try IPMI fencing first then default to a "sure-kill" dual PDU fencing

In a normal failure scenario, STONITH will first select **fence_ipmi** to try to kill the faulty node. Using a fencing topology, if that first method fails, STONITH will then move on to selecting **fence_apc_snmp** twice:

- once for the first PDU
- again for the second PDU

The fence action is considered successful only if both PDUs report the required status. If any of them fails, STONITH loops back to the first fencing method, **fence_ipmi**, and so on until the node is fenced or fencing action is cancelled.

First fencing method: single IPMI device

Each cluster node has its own dedicated IPMI channel that can be called for fencing using the following primitives:

```
<primitive class="stonith" id="fence_prod-mysql1_ipmi" type="fence_ipmilan">
  <instance_attributes id="fence_prod-mysql1_ipmi-instance_attributes">
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-ipaddr" name="ipaddr"
value="192.0.2.1"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-action" name="action" value="off"/>
  >
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-passwd" name="passwd"
value="finishme"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-verbose" name="verbose"
value="true"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-lanplus" name="lanplus"
value="true"/>
  </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_ipmi" type="fence_ipmilan">
  <instance_attributes id="fence_prod-mysql2_ipmi-instance_attributes">
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-ipaddr" name="ipaddr"
value="192.0.2.2"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-action" name="action" value="off"/>
  >
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-passwd" name="passwd"
value="finishme"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-verbose" name="verbose"
value="true"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-lanplus" name="lanplus"
value="true"/>
  </instance_attributes>
</primitive>
```

Second fencing method: dual PDU devices

Each cluster node also has two distinct power channels controlled by two distinct PDUs. That means a total of 4 fencing devices configured as follows:

- Node 1, PDU 1, PSU 1 @ port 10
- Node 1, PDU 2, PSU 2 @ port 10
- Node 2, PDU 1, PSU 1 @ port 11
- Node 2, PDU 2, PSU 2 @ port 11

The matching fencing agents are configured as follows:

```
<primitive class="stonith" id="fence_prod-mysql1_apc1" type="fence_apc_snmp">
  <instance_attributes id="fence_prod-mysql1_apc1-instance_attributes">
    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-ipaddr" name="ipaddr"
value="198.51.100.1"/>
    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-action" name="action" value="off"/>
  >
```

```

    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-port" name="port" value="10"/>
    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-passwd" name="passwd"
value="fencing"/>
    <nvpair id="fence_prod-mysql1_apc1-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
  </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql1_apc2" type="fence_apc_snmp">
  <instance_attributes id="fence_prod-mysql1_apc2-instance_attributes">
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-ipaddr" name="ipaddr"
value="203.0.113.1"/>
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-action" name="action" value="off"/>
  >
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-port" name="port" value="10"/>
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-passwd" name="passwd"
value="fencing"/>
    <nvpair id="fence_prod-mysql1_apc2-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
  </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_apc1" type="fence_apc_snmp">
  <instance_attributes id="fence_prod-mysql2_apc1-instance_attributes">
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-ipaddr" name="ipaddr"
value="198.51.100.1"/>
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-action" name="action" value="off"/>
  >
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-port" name="port" value="11"/>
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-passwd" name="passwd"
value="fencing"/>
    <nvpair id="fence_prod-mysql2_apc1-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
  </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_apc2" type="fence_apc_snmp">
  <instance_attributes id="fence_prod-mysql2_apc2-instance_attributes">
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-ipaddr" name="ipaddr"
value="203.0.113.1"/>
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-action" name="action" value="off"/>
  >
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-port" name="port" value="11"/>
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-login" name="login"
value="fencing"/>
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-passwd" name="passwd"
value="fencing"/>
    <nvpair id="fence_prod-mysql2_apc2-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
  </instance_attributes>
</primitive>

```

Location Constraints

To prevent STONITH from trying to run a fencing agent on the same node it is supposed to fence, constraints are placed on all the fencing primitives:

```

<constraints>
  <rsc_location id="l_fence_prod-mysql1_ipmi" node="prod-mysql1" rsc="fence_prod-mysql1_ipmi"
score="-INFINITY"/>
  <rsc_location id="l_fence_prod-mysql2_ipmi" node="prod-mysql2" rsc="fence_prod-mysql2_ipmi"
score="-INFINITY"/>

```



```

<rscl_location id="l_fence_prod-mysql1_apc2" node="prod-mysql1" rsc="fence_prod-mysql1_apc2"
score="- INFINITY"/>
<rscl_location id="l_fence_prod-mysql1_apc1" node="prod-mysql1" rsc="fence_prod-mysql1_apc1"
score="- INFINITY"/>
<rscl_location id="l_fence_prod-mysql2_apc1" node="prod-mysql2" rsc="fence_prod-mysql2_apc1"
score="- INFINITY"/>
<rscl_location id="l_fence_prod-mysql2_apc2" node="prod-mysql2" rsc="fence_prod-mysql2_apc2"
score="- INFINITY"/>
</constraints>

```

Fencing topology

Now that all the fencing resources are defined, it's time to create the right topology. We want to first fence using IPMI and if that does not work, fence both PDUs to effectively and surely kill the node.

```

<fencing-topology>
  <fencing-level devices="fence_prod-mysql1_ipmi" id="fencing-2" index="1" target="prod-
mysql1"/>
  <fencing-level devices="fence_prod-mysql1_apc1,fence_prod-mysql1_apc2" id="fencing-3"
index="2" target="prod-mysql1"/>
  <fencing-level devices="fence_prod-mysql2_ipmi" id="fencing-0" index="1" target="prod-
mysql2"/>
  <fencing-level devices="fence_prod-mysql2_apc1,fence_prod-mysql2_apc2" id="fencing-1"
index="2" target="prod-mysql2"/>
</fencing-topology>

```

Please note, in **fencing-topology**, the lowest **index** value determines the priority of the first fencing method.

Final configuration

Put together, the configuration looks like this:

```

<cib admin_epoch="0" crm_feature_set="3.0.7" epoch="292" have-quorum="1" num_updates="29"
validate-with="pacemaker-1.2">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="cib-bootstrap-options-stonith-enabled" name="stonith-enabled"
value="true"/>
        <nvpair id="cib-bootstrap-options-stonith-action" name="stonith-action" value="off"/>
        <nvpair id="cib-bootstrap-options-expected-quorum-votes" name="expected-quorum-votes"
value="3"/>
        ...
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="prod-mysql1" uname="prod-mysql1">
      <node id="prod-mysql2" uname="prod-mysql2"/>
      <node id="prod-mysql-rep1" uname="prod-mysql-rep1"/>
        <instance_attributes id="prod-mysql-rep1">
          <nvpair id="prod-mysql-rep1-standby" name="standby" value="on"/>
        </instance_attributes>
      </node>
    </nodes>
    <resources>
      <primitive class="stonith" id="fence_prod-mysql1_ipmi" type="fence_ipmilan">
        <instance_attributes id="fence_prod-mysql1_ipmi-instance_attributes">
          <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-ipaddr" name="ipaddr"
value="192.0.2.1"/>
          <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-action" name="action"
value="off"/>

```

```

        <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-passwd" name="passwd"
value="finishme"/>
        <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-verbose" name="verbose"
value="true"/>
        <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
        <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-lanplus" name="lanplus"
value="true"/>
    </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_ipmi" type="fence_ipmilan">
    <instance_attributes id="fence_prod-mysql2_ipmi-instance_attributes">
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-ipaddr" name="ipaddr"
value="192.0.2.2"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-action" name="action"
value="off"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-passwd" name="passwd"
value="finishme"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-verbose" name="verbose"
value="true"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
        <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-lanplus" name="lanplus"
value="true"/>
    </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql1_apc1" type="fence_apc_snmp">
    <instance_attributes id="fence_prod-mysql1_apc1-instance_attributes">
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-ipaddr" name="ipaddr"
value="198.51.100.1"/>
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-action" name="action"
value="off"/>
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-port" name="port"
value="10"/>
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-passwd" name="passwd"
value="fencing"/>
        <nvpair id="fence_prod-mysql1_apc1-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
    </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql1_apc2" type="fence_apc_snmp">
    <instance_attributes id="fence_prod-mysql1_apc2-instance_attributes">
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-ipaddr" name="ipaddr"
value="203.0.113.1"/>
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-action" name="action"
value="off"/>
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-port" name="port"
value="10"/>
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-passwd" name="passwd"
value="fencing"/>
        <nvpair id="fence_prod-mysql1_apc2-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql1"/>
    </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_apc1" type="fence_apc_snmp">
    <instance_attributes id="fence_prod-mysql2_apc1-instance_attributes">
        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-ipaddr" name="ipaddr"
value="198.51.100.1"/>

```

```

        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-action" name="action"
value="off"/>
        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-port" name="port"
value="11"/>
        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-passwd" name="passwd"
value="fencing"/>
        <nvpair id="fence_prod-mysql2_apc1-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
    </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_prod-mysql2_apc2" type="fence_apc_snmp">
    <instance_attributes id="fence_prod-mysql2_apc2-instance_attributes">
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-ipaddr" name="ipaddr"
value="203.0.113.1"/>
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-action" name="action"
value="off"/>
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-port" name="port"
value="11"/>
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-login" name="login"
value="fencing"/>
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-passwd" name="passwd"
value="fencing"/>
        <nvpair id="fence_prod-mysql2_apc2-instance_attributes-pcmk_host_list"
name="pcm_k_host_list" value="prod-mysql2"/>
    </instance_attributes>
</primitive>
</resources>
<constraints>
    <rsc_location id="l_fence_prod-mysql1_ipmi" node="prod-mysql1" rsc="fence_prod-
mysql1_ipmi" score="-INFINITY"/>
    <rsc_location id="l_fence_prod-mysql2_ipmi" node="prod-mysql2" rsc="fence_prod-
mysql2_ipmi" score="-INFINITY"/>
    <rsc_location id="l_fence_prod-mysql1_apc2" node="prod-mysql1" rsc="fence_prod-
mysql1_apc2" score="-INFINITY"/>
    <rsc_location id="l_fence_prod-mysql1_apc1" node="prod-mysql1" rsc="fence_prod-
mysql1_apc1" score="-INFINITY"/>
    <rsc_location id="l_fence_prod-mysql2_apc1" node="prod-mysql2" rsc="fence_prod-
mysql2_apc1" score="-INFINITY"/>
    <rsc_location id="l_fence_prod-mysql2_apc2" node="prod-mysql2" rsc="fence_prod-
mysql2_apc2" score="-INFINITY"/>
</constraints>
<fencing-topology>
    <fencing-level devices="fence_prod-mysql1_ipmi" id="fencing-2" index="1" target="prod-
mysql1"/>
    <fencing-level devices="fence_prod-mysql1_apc1,fence_prod-mysql1_apc2" id="fencing-3"
index="2" target="prod-mysql1"/>
    <fencing-level devices="fence_prod-mysql2_ipmi" id="fencing-0" index="1" target="prod-
mysql2"/>
    <fencing-level devices="fence_prod-mysql2_apc1,fence_prod-mysql2_apc2" id="fencing-1"
index="2" target="prod-mysql2"/>
</fencing-topology>
    ...
</configuration>
</cib>

```

13.6. Remapping Reboots

When the cluster needs to reboot a node, whether because **stonith-action** is **reboot** or because a reboot was manually requested (such as by **stonith_admin --reboot**), it will remap that to other commands in two cases:

1. If the chosen fencing device does not support the **reboot** command, the cluster will ask it to perform **off** instead.

2. If a fencing topology level with multiple devices must be executed, the cluster will ask all the devices to perform **off**, then ask the devices to perform **on**.

To understand the second case, consider the example of a node with redundant power supplies connected to intelligent power switches. Rebooting one switch and then the other would have no effect on the node. Turning both switches off, and then on, actually reboots the node.

In such a case, the fencing operation will be treated as successful as long as the **off** commands succeed, because then it is safe for the cluster to recover any resources that were on the node. Timeouts and errors in the **on** phase will be logged but ignored.

When a reboot operation is remapped, any action-specific timeout for the remapped action will be used (for example, **pcmk_off_timeout** will be used when executing the **off** command, not **pcmk_reboot_timeout**).



Note

In Pacemaker versions 1.1.13 and earlier, reboots will not be remapped in the second case. To achieve the same effect, separate fencing devices for off and on actions must be configured.

Status — Here be dragons

Table of Contents

14.1. Node Status	121
14.2. Transient Node Attributes	122
14.3. Operation History	122
14.3.1. Simple Operation History Example	124
14.3.2. Complex Operation History Example	125

Most users never need to understand the contents of the status section and can be happy with the output from **crm_mon**.

However for those with a curious inclination, this section attempts to provide an overview of its contents.

14.1. Node Status

In addition to the cluster's configuration, the CIB holds an up-to-date representation of each cluster node in the **status** section.

Example 14.1. A bare-bones status entry for a healthy node **cl-virt-1**

```
<node_state id="cl-virt-1" uname="cl-virt-2" ha="active" in_ccm="true" crmd="online"
join="member" expected="member" crm-debug-origin="do_update_resource">
  <transient_attributes id="cl-virt-1"/>
  <lrm id="cl-virt-1"/>
</node_state>
```

Users are highly recommended *not* to modify any part of a node's state *directly*. The cluster will periodically regenerate the entire section from authoritative sources, so any changes should be done with the tools appropriate to those sources.

Table 14.1. Authoritative Sources for State Information

CIB Object	Authoritative Source
node_state	crmd
transient_attributes	attrd
lrm	lrmd

The fields used in the **node_state** objects are named as they are largely for historical reasons and are rooted in Pacemaker's origins as the Heartbeat resource manager. They have remained unchanged to preserve compatibility with older versions.

Table 14.2. Node Status Fields

Field	Description
id	Unique identifier for the node. Corosync-based clusters use a numeric counter, while Heartbeat clusters use a (barely) human-readable UUID.
uname	The node's machine name (output from uname -n).

Field	Description
ha	Is the cluster software active on this node? Allowed values: active , dead .
in_ccm	Is the node a member of the cluster? Allowed values: true , false .
crmd	Is the crmd process active on the node? Allowed values: online , offline .
join	Does the node participate in hosting resources? Allowed values: down , pending , member , banned .
expected	Expected value for join .
crm-debug-origin	The origin of the most recent change(s). For diagnostic purposes.

The cluster uses these fields to determine whether, at the node level, the node is healthy or is in a failed state and needs to be fenced.

14.2. Transient Node Attributes

Like regular [node attributes](#), the name/value pairs listed in the **transient_attributes** section help to describe the node. However they are forgotten by the cluster when the node goes offline. This can be useful, for instance, when you want a node to be in standby mode (not able to run resources) just until the next reboot.

In addition to any values the administrator sets, the cluster will also store information about failed resources here.

Example 14.2. A set of transient node attributes for node **cl-virt-1**

```
<transient_attributes id="cl-virt-1">
  <instance_attributes id="status-cl-virt-1">
    <nvpair id="status-cl-virt-1-pingd" name="pingd" value="3"/>
    <nvpair id="status-cl-virt-1-probe_complete" name="probe_complete" value="true"/>
    <nvpair id="status-cl-virt-1-fail-count-pingd:0" name="fail-count-pingd:0" value="1"/>
  >
  <nvpair id="status-cl-virt-1-last-failure-pingd:0" name="last-failure-pingd:0"
value="1239009742"/>
</instance_attributes>
</transient_attributes>
```

In the above example, we can see that the **pingd:0** resource has failed once, at 09:22:22 UTC 6 April 2009.¹ We also see that the node is connected to three **pingd** peers and that all known resources have been checked for on this machine (**probe_complete**).

14.3. Operation History

A node's resource history is held in the **lrm_resources** tag (a child of the **lrm** tag). The information stored here includes enough information for the cluster to stop the resource safely if it is removed from the **configuration** section. Specifically, the resource's **id**, **class**, **type** and **provider** are stored.

¹ You can use the standard **date** command to print a human-readable version of any seconds-since-epoch value, for example **date -d @1239009742**.

Example 14.3. A record of the **apcstonith** resource

```
<lrn_resource id="apcstonith" type="apcmastersnmp" class="stonith"/>
```

Additionally, we store the last job for every combination of **resource**, **action** and **interval**. The concatenation of the values in this tuple are used to create the id of the **lrn_rsc_op** object.

Table 14.3. Contents of an **lrn_rsc_op** job

Field	Description
id	Identifier for the job constructed from the resource's id , operation and interval .
call-id	The job's ticket number. Used as a sort key to determine the order in which the jobs were executed.
operation	The action the resource agent was invoked with.
interval	The frequency, in milliseconds, at which the operation will be repeated. A one-off job is indicated by 0.
op-status	The job's status. Generally this will be either 0 (done) or -1 (pending). Rarely used in favor of rc-code .
rc-code	The job's result. Refer to Section B.4, "OCF Return Codes" for details on what the values here mean and how they are interpreted.
last-run	Machine-local date/time, in seconds since epoch, at which the job was executed. For diagnostic purposes.
last-rc-change	Machine-local date/time, in seconds since epoch, at which the job first returned the current value of rc-code . For diagnostic purposes.
exec-time	Time, in milliseconds, that the job was running for. For diagnostic purposes.
queue-time	Time, in seconds, that the job was queued for in the LRMD. For diagnostic purposes.

Field	Description
crm_feature_set	The version which this job description conforms to. Used when processing op-digest .
transition-key	A concatenation of the job's graph action number, the graph number, the expected result and the UUID of the crmd instance that scheduled it. This is used to construct transition-magic (below).
transition-magic	A concatenation of the job's op-status , rc-code and transition-key . Guaranteed to be unique for the life of the cluster (which ensures it is part of CIB update notifications) and contains all the information needed for the crmd to correctly analyze and process the completed job. Most importantly, the decomposed elements tell the crmd if the job entry was expected and whether it failed.
op-digest	An MD5 sum representing the parameters passed to the job. Used to detect changes to the configuration, to restart resources if necessary.
crm-debug-origin	The origin of the current values. For diagnostic purposes.

14.3.1. Simple Operation History Example

Example 14.4. A monitor operation (determines current state of the **apcstonith** resource)

```
<lr_resource id="apcstonith" type="apcmastersnmp" class="stonith">
  <lr_rsc_op id="apcstonith_monitor_0" operation="monitor" call-id="2"
    rc-code="7" op-status="0" interval="0"
    crm-debug-origin="do_update_resource" crm_feature_set="3.0.1"
    op-digest="2e3da9274d3550dc6526fb24bfcba0"
    transition-key="22:2:7:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    transition-magic="0:7;22:2:7:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    last-run="1239008085" last-rc-change="1239008085" exec-time="10" queue-time="0"/>
</lr_resource>
```

In the above example, the job is a non-recurring monitor operation often referred to as a "probe" for the **apcstonith** resource.

The cluster schedules probes for every configured resource on a node when the node first starts, in order to determine the resource's current state before it takes any further action.

From the **transition-key**, we can see that this was the 22nd action of the 2nd graph produced by this instance of the crmd (2668bbeb-06d5-40f9-936d-24cb7f87006a).

The third field of the **transition-key** contains a 7, which indicates that the job expects to find the resource inactive. By looking at the **rc-code** property, we see that this was the case.

As that is the only job recorded for this node, we can conclude that the cluster started the resource elsewhere.

14.3.2. Complex Operation History Example

Example 14.5. Resource history of a **pingd** clone with multiple jobs

```
<lr_resource id="pingd:0" type="pingd" class="ocf" provider="pacemaker">
  <lr_rsc_op id="pingd:0_monitor_30000" operation="monitor" call-id="34"
    rc-code="0" op-status="0" interval="30000"
    crm-debug-origin="do_update_resource" crm_feature_set="3.0.1"
    transition-key="10:11:0:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    ...
    last-run="1239009741" last-rc-change="1239009741" exec-time="10" queue-time="0"/>
  <lr_rsc_op id="pingd:0_stop_0" operation="stop"
    crm-debug-origin="do_update_resource" crm_feature_set="3.0.1" call-id="32"
    rc-code="0" op-status="0" interval="0"
    transition-key="11:11:0:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    ...
    last-run="1239009741" last-rc-change="1239009741" exec-time="10" queue-time="0"/>
  <lr_rsc_op id="pingd:0_start_0" operation="start" call-id="33"
    rc-code="0" op-status="0" interval="0"
    crm-debug-origin="do_update_resource" crm_feature_set="3.0.1"
    transition-key="31:11:0:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    ...
    last-run="1239009741" last-rc-change="1239009741" exec-time="10" queue-time="0" />
  <lr_rsc_op id="pingd:0_monitor_0" operation="monitor" call-id="3"
    rc-code="0" op-status="0" interval="0"
    crm-debug-origin="do_update_resource" crm_feature_set="3.0.1"
    transition-key="23:2:7:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    ...
    last-run="1239008085" last-rc-change="1239008085" exec-time="20" queue-time="0"/>
</lr_resource>
```

When more than one job record exists, it is important to first sort them by **call-id** before interpreting them.

Once sorted, the above example can be summarized as:

1. A non-recurring monitor operation returning 7 (not running), with a **call-id** of 3
2. A stop operation returning 0 (success), with a **call-id** of 32
3. A start operation returning 0 (success), with a **call-id** of 33
4. A recurring monitor returning 0 (success), with a **call-id** of 34

The cluster processes each job record to build up a picture of the resource's state. After the first and second entries, it is considered stopped, and after the third it considered active.

Based on the last operation, we can tell that the resource is currently active.

Additionally, from the presence of a **stop** operation with a lower **call-id** than that of the **start** operation, we can conclude that the resource has been restarted. Specifically this occurred as part of actions 11 and 31 of transition 11 from the crmd instance with the key **2668bbeb....**. This information can be helpful for locating the relevant section of the logs when looking for the source of a failure.

Multi-Site Clusters and Tickets

Table of Contents

15.1. Challenges for Multi-Site Clusters	127
15.2. Conceptual Overview	127
15.2.1. Ticket	127
15.2.2. Dead Man Dependency	128
15.2.3. Cluster Ticket Registry	128
15.2.4. Configuration Replication	128
15.3. Configuring Ticket Dependencies	129
15.4. Managing Multi-Site Clusters	130
15.4.1. Granting and Revoking Tickets Manually	130
15.4.2. Granting and Revoking Tickets via a Cluster Ticket Registry	130
15.4.3. General Management of Tickets	131
15.5. For more information	132

Apart from local clusters, Pacemaker also supports multi-site clusters. That means you can have multiple, geographically dispersed sites, each with a local cluster. Failover between these clusters can be coordinated manually by the administrator, or automatically by a higher-level entity called a *Cluster Ticket Registry (CTR)*.

15.1. Challenges for Multi-Site Clusters

Typically, multi-site environments are too far apart to support synchronous communication and data replication between the sites. That leads to significant challenges:

- How do we make sure that a cluster site is up and running?
- How do we make sure that resources are only started once?
- How do we make sure that quorum can be reached between the different sites and a split-brain scenario avoided?
- How do we manage failover between sites?
- How do we deal with high latency in case of resources that need to be stopped?

In the following sections, learn how to meet these challenges.

15.2. Conceptual Overview

Multi-site clusters can be considered as “overlay” clusters where each cluster site corresponds to a cluster node in a traditional cluster. The overlay cluster can be managed by a CTR in order to guarantee that any cluster resource will be active on no more than one cluster site. This is achieved by using *tickets* that are treated as failover domain between cluster sites, in case a site should be down.

The following sections explain the individual components and mechanisms that were introduced for multi-site clusters in more detail.

15.2.1. Ticket

Tickets are, essentially, cluster-wide attributes. A ticket grants the right to run certain resources on a specific cluster site. Resources can be bound to a certain ticket by **rsc_ticket** constraints. Only if

the ticket is available at a site can the respective resources be started there. Vice versa, if the ticket is revoked, the resources depending on that ticket must be stopped.

The ticket thus is similar to a *site quorum*, i.e. the permission to manage/own resources associated with that site. (One can also think of the current **have-quorum** flag as a special, cluster-wide ticket that is granted in case of node majority.)

Tickets can be granted and revoked either manually by administrators (which could be the default for classic enterprise clusters), or via the automated CTR mechanism described below.

A ticket can only be owned by one site at a time. Initially, none of the sites has a ticket. Each ticket must be granted once by the cluster administrator.

The presence or absence of tickets for a site is stored in the CIB as a cluster status. With regards to a certain ticket, there are only two states for a site: **true** (the site has the ticket) or **false** (the site does not have the ticket). The absence of a certain ticket (during the initial state of the multi-site cluster) is the same as the value **false**.

15.2.2. Dead Man Dependency

A site can only activate resources safely if it can be sure that the other site has deactivated them. However after a ticket is revoked, it can take a long time until all resources depending on that ticket are stopped "cleanly", especially in case of cascaded resources. To cut that process short, the concept of a *Dead Man Dependency* was introduced.

If a dead man dependency is in force, if a ticket is revoked from a site, the nodes that are hosting dependent resources are fenced. This considerably speeds up the recovery process of the cluster and makes sure that resources can be migrated more quickly.

This can be configured by specifying a **loss-policy="fence"** in **rsc_ticket** constraints.

15.2.3. Cluster Ticket Registry

A CTR is a coordinated group of network daemons that automatically handles granting, revoking, and timing out tickets (instead of the administrator revoking the ticket somewhere, waiting for everything to stop, and then granting it on the desired site).

Pacemaker does not implement its own CTR, but interoperates with external software designed for that purpose (similar to how resource and fencing agents are not directly part of pacemaker).

Participating clusters run the CTR daemons, which connect to each other, exchange information about their connectivity, and vote on which sites gets which tickets.

A ticket is granted to a site only once the CTR is sure that the ticket has been relinquished by the previous owner, implemented via a timer in most scenarios. If a site loses connection to its peers, its tickets time out and recovery occurs. After the connection timeout plus the recovery timeout has passed, the other sites are allowed to re-acquire the ticket and start the resources again.

This can also be thought of as a "quorum server", except that it is not a single quorum ticket, but several.

15.2.4. Configuration Replication

As usual, the CIB is synchronized within each cluster, but it is *not* synchronized across cluster sites of a multi-site cluster. You have to configure the resources that will be highly available across the multi-site cluster for every site accordingly.

15.3. Configuring Ticket Dependencies

The **rsc_ticket** constraint lets you specify the resources depending on a certain ticket. Together with the constraint, you can set a **loss-policy** that defines what should happen to the respective resources if the ticket is revoked.

The attribute **loss-policy** can have the following values:

- **fence**: Fence the nodes that are running the relevant resources.
- **stop**: Stop the relevant resources.
- **freeze**: Do nothing to the relevant resources.
- **demote**: Demote relevant resources that are running in master mode to slave mode.

Example 15.1. Constraint that fences node if **ticketA** is revoked

```
<rsc_ticket id="rsc1-req-ticketA" rsc="rsc1" ticket="ticketA" loss-policy="fence"/>
```

The example above creates a constraint with the ID **rsc1-req-ticketA**. It defines that the resource **rsc1** depends on **ticketA** and that the node running the resource should be fenced if **ticketA** is revoked.

If resource **rsc1** were a multi-state resource (i.e. it could run in master or slave mode), you might want to configure that only master mode depends on **ticketA**. With the following configuration, **rsc1** will be demoted to slave mode if **ticketA** is revoked:

Example 15.2. Constraint that demotes **rsc1** if **ticketA** is revoked

```
<rsc_ticket id="rsc1-req-ticketA" rsc="rsc1" rsc-role="Master" ticket="ticketA" loss-policy="demote"/>
```

You can create multiple **rsc_ticket** constraints to let multiple resources depend on the same ticket. However, **rsc_ticket** also supports resource sets (see [Section 6.5, “Resource Sets”](#)), so one can easily list all the resources in one **rsc_ticket** constraint instead.

Example 15.3. Ticket constraint for multiple resources

```
<rsc_ticket id="resources-dep-ticketA" ticket="ticketA" loss-policy="fence">
  <resource_set id="resources-dep-ticketA-0" role="Started">
    <resource_ref id="rsc1"/>
    <resource_ref id="group1"/>
    <resource_ref id="clone1"/>
  </resource_set>
  <resource_set id="resources-dep-ticketA-1" role="Master">
    <resource_ref id="ms1"/>
  </resource_set>
</rsc_ticket>
```

In the example above, there are two resource sets, so we can list resources with different roles in a single **rsc_ticket** constraint. There's no dependency between the two resource sets, and there's no dependency among the resources within a resource set. Each of the resources just depends on **ticketA**.

Referencing resource templates in **rsc_ticket** constraints, and even referencing them within resource sets, is also supported.

If you want other resources to depend on further tickets, create as many constraints as necessary with **rsc_ticket**.

15.4. Managing Multi-Site Clusters

15.4.1. Granting and Revoking Tickets Manually

You can grant tickets to sites or revoke them from sites manually. If you want to re-distribute a ticket, you should wait for the dependent resources to stop cleanly at the previous site before you grant the ticket to the new site.

Use the **crm_ticket** command line tool to grant and revoke tickets.

To grant a ticket to this site:

```
# crm_ticket --ticket ticketA --grant
```

To revoke a ticket from this site:

```
# crm_ticket --ticket ticketA --revoke
```



Important

If you are managing tickets manually, use the **crm_ticket** command with great care, because it cannot check whether the same ticket is already granted elsewhere.

15.4.2. Granting and Revoking Tickets via a Cluster Ticket Registry

We will use [Booth](https://github.com/ClusterLabs/booth)¹ here as an example of software that can be used with pacemaker as a Cluster Ticket Registry. Booth implements the [Raft](http://en.wikipedia.org/wiki/Raft_%28computer_science%29)² algorithm to guarantee the distributed consensus among different cluster sites, and manages the ticket distribution (and thus the failover process between sites).

Each of the participating clusters and *arbitrators* runs the Booth daemon **boothd**.

An *arbitrator* is the multi-site equivalent of a quorum-only node in a local cluster. If you have a setup with an even number of sites, you need an additional instance to reach consensus about decisions such as failover of resources across sites. In this case, add one or more arbitrators running at additional sites. Arbitrators are single machines that run a booth instance in a special mode. An arbitrator is especially important for a two-site scenario, otherwise there is no way for one site to distinguish between a network failure between it and the other site, and a failure of the other site.

The most common multi-site scenario is probably a multi-site cluster with two sites and a single arbitrator on a third site. However, technically, there are no limitations with regards to the number of sites and the number of arbitrators involved.

¹ <https://github.com/ClusterLabs/booth>

² http://en.wikipedia.org/wiki/Raft_%28computer_science%29

Boothd at each site connects to its peers running at the other sites and exchanges connectivity details. Once a ticket is granted to a site, the booth mechanism will manage the ticket automatically: If the site which holds the ticket is out of service, the booth daemons will vote which of the other sites will get the ticket. To protect against brief connection failures, sites that lose the vote (either explicitly or implicitly by being disconnected from the voting body) need to relinquish the ticket after a time-out. Thus, it is made sure that a ticket will only be re-distributed after it has been relinquished by the previous site. The resources that depend on that ticket will fail over to the new site holding the ticket. The nodes that have run the resources before will be treated according to the **loss-policy** you set within the **rsc_ticket** constraint.

Before the booth can manage a certain ticket within the multi-site cluster, you initially need to grant it to a site manually via the **booth** command-line tool. After you have initially granted a ticket to a site, **boothd** will take over and manage the ticket automatically.



Important

The **booth** command-line tool can be used to grant, list, or revoke tickets and can be run on any machine where **boothd** is running. If you are managing tickets via Booth, use only **booth** for manual intervention, not **crm_ticket**. That ensures the same ticket will only be owned by one cluster site at a time.

15.4.2.1. Booth Requirements

- All clusters that will be part of the multi-site cluster must be based on Pacemaker.
- Booth must be installed on all cluster nodes and on all arbitrators that will be part of the multi-site cluster.
- Nodes belonging to the same cluster site should be synchronized via NTP. However, time synchronization is not required between the individual cluster sites.

15.4.3. General Management of Tickets

Display the information of tickets:

```
# crm_ticket --info
```

Or you can monitor them with:

```
# crm_mon --tickets
```

Display the **rsc_ticket** constraints that apply to a ticket:

```
# crm_ticket --ticket ticketA --constraints
```

When you want to do maintenance or manual switch-over of a ticket, revoking the ticket would trigger the loss policies. If **loss-policy="fence"**, the dependent resources could not be gracefully stopped/demoted, and other unrelated resources could even be affected.

The proper way is making the ticket *standby* first with:

```
# crm_ticket --ticket ticketA --standby
```

Then the dependent resources will be stopped or demoted gracefully without triggering the loss policies.

If you have finished the maintenance and want to activate the ticket again, you can run:

```
# crm_ticket --ticket ticketA --activate
```

15.5. For more information

- [SUSE's Geo Clustering quick start](#)³
- [Booth](#)⁴

³ https://www.suse.com/documentation/sle-ha-geo-12/art_ha_geo_quick/data/art_ha_geo_quick.html

⁴ <https://github.com/ClusterLabs/booth>

Appendix A. FAQ

Q: Why is the Project Called Pacemaker?

A: First of all, the reason it's not called the CRM is because of the abundance of terms¹ that are commonly abbreviated to those three letters. The Pacemaker name came from Kham,² a good friend of Pacemaker developer Andrew Beekhof's, and was originally used by a Java GUI that Beekhof was prototyping in early 2007. Alas, other commitments prevented the GUI from progressing much and, when it came time to choose a name for this project, Lars Marowsky-Bree suggested it was an even better fit for an independent CRM. The idea stems from the analogy between the role of this software and that of the little device that keeps the human heart pumping. Pacemaker monitors the cluster and intervenes when necessary to ensure the smooth operation of the services it provides. There were a number of other names (and acronyms) tossed around, but suffice to say "Pacemaker" was the best.

Q: Why was the Pacemaker Project Created?

A: The decision was made to spin-off the CRM into its own project after the 2.1.3 Heartbeat release in order to:

- support both the Corosync and Heartbeat cluster stacks equally
 - decouple the release cycles of two projects at very different stages of their life-cycles
 - foster clearer package boundaries, thus leading to better and more stable interfaces
-

Q: What Messaging Layers are Supported?

A:

- [Corosync](#)³
 - [Heartbeat](#)⁴
-

Q: Can I Choose Which Messaging Layer to Use at Run Time?

A: Yes. The CRM will automatically detect which started it and behave accordingly.

Q: Can I Have a Mixed Heartbeat-Corosync Cluster?

A: No.

Q: Which Messaging Layer Should I Choose?

A: You can choose from multiple messaging layers, including heartbeat, corosync 1 (with or without CMAN), and corosync 2. Corosync 2 is the current state of the art due to its more advanced features and better support for pacemaker, but often the best choice is to use whatever comes with your Linux distribution, and follow the distribution's setup instructions.

¹ <http://en.wikipedia.org/wiki/CRM>

² <http://khamsook.souvanlasy.com/>

³ <http://www.corosync.org/>

⁴ <http://linux-ha.org/>

Q: Where Can I Get Pre-built Packages?

A: Most major Linux distributions have pacemaker packages in their standard package repositories. See the [Install wiki page](http://clusterlabs.org/wiki/Install)⁵ for details.

Q: What Versions of Pacemaker Are Supported?

A: Some Linux distributions (such as Red Hat Enterprise Linux and SUSE Linux Enterprise) offer technical support for their customers; contact them for details of such support. For help within the community (mailing lists, IRC, etc.) from Pacemaker developers and users, refer to the [Releases wiki page](http://clusterlabs.org/wiki/Releases)⁶ for an up-to-date list of versions considered to be supported by the project. When seeking assistance, please try to ensure you have one of these versions.

⁵ <http://clusterlabs.org/wiki/Install>

⁶ <http://clusterlabs.org/wiki/Releases>

Appendix B. More About OCF Resource Agents

Table of Contents

B.1. Location of Custom Scripts	135
B.2. Actions	135
B.3. How are OCF Return Codes Interpreted?	136
B.4. OCF Return Codes	136

B.1. 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 `bigCorp` and want a new resource named `bigApp`, you would create a resource agent called `/usr/lib/ocf/resource.d/bigCorp/bigApp` and define a resource:

```
<primitive id="custom-app" class="ocf" provider="bigCorp" type="bigApp"/>
```

B.2. Actions

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

Table B.1. 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 anything else if it is failed. NOTE: The monitor script should test the state of the resource on the local machine only.
meta-data	Describe the resource	Provide information about this resource as an XML snippet. Exit with 0. NOTE: This is <i>not</i> performed as root.
validate-all	Verify the supplied parameters	Return 0 if parameters are valid, 2 if not valid, and 6 if resource is not configured.

Additional requirements (not part of the OCF specification) are placed on agents that will be used for advanced concepts such as [clones](#) and [multi-state](#) resources.

Table B.2. Optional Actions for OCF Resource Agents

Action	Description	Instructions
promote	Promote the local instance of a multi-state resource to the master (primary) state.	Return 0 on success
demote	Demote the local instance of a multi-state resource to the slave (secondary) state.	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

One action specified in the OCF specs, **recover**, is not currently used by the cluster. It is intended to be a variant of the **start** action that tries to recover a resource locally.



Important

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

B.3. 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 B.3. 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 occurred	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

B.4. 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 B.4. OCF Return Codes and their Recovery Types

RC	OCF Alias	Description	RT
0	OCF_SUCCESS	Success. The command completed successfully. This is the expected result for all start, stop, promote and demote commands.	soft

RC	OCF Alias	Description	RT
1	OCF_ERR_GENERIC	Generic "there was a problem" error code.	soft
2	OCF_ERR_ARGS	The resource's configuration is not valid on this machine. E.g. it refers to a location not found on the node.	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 configuration is invalid. E.g. required parameters are missing.	fatal
7	OCF_NOT_RUNNING	The resource is safely stopped. The cluster will not attempt to stop a resource that returns this for any action.	N/A
8	OCF_RUNNING_MASTER	The resource is running in master mode.	soft
9	OCF_FAILED_MASTER	The resource is in master mode but has failed. The resource will be demoted, stopped and then started (and possibly promoted) again.	soft
other	N/A	Custom error code.	soft

Exceptions to the recovery handling described above:

- Probes (non-recurring monitor actions) that find a resource active (or in master mode) 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 (see [Section 5.4, "Resource Options"](#)).
- Recurring actions that return **OCF_ERR_UNIMPLEMENTED** do not cause any type of recovery.

Appendix C. What Changed in 1.0

Table of Contents

C.1. New	139
C.2. Changed	139
C.3. Removed	140

C.1. New

- Failure timeouts. See [Section 9.3.2, “Moving Resources Due to Failure”](#)
- New section for resource and operation defaults. See [Section 5.4.2, “Setting Global Defaults for Resource Meta-Attributes”](#) and [Section 5.5.3, “Setting Global Defaults for Operations”](#)
- Tool for making offline configuration changes. See [Section 2.4, “Making Configuration Changes in a Sandbox”](#)
- **Rules**, **instance_attributes**, **meta_attributes** and sets of operations can be defined once and referenced in multiple places. See [Section 9.4, “Reusing Rules, Options and Sets of Operations”](#)
- The CIB now accepts XPath-based create/modify/delete operations. See the **cibadmin** help text.
- Multi-dimensional colocation and ordering constraints. See [Section 6.6, “Ordering Sets of Resources”](#) and [Section 6.7, “Colocating Sets of Resources”](#)
- The ability to connect to the CIB from non-cluster machines. See [Section 9.1, “Connecting from a Remote Machine”](#)
- Allow recurring actions to be triggered at known times. See [Section 9.2, “Specifying When Recurring Actions are Performed”](#)

C.2. 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. See [Section 6.3, “Specifying the Order in which Resources Should Start/Stop”](#) and [Section 6.4, “Placing Resources Relative to other Resources”](#)

- **resource-failure-stickiness** has been replaced by **migration-threshold**. See [Section 9.3.2, “Moving Resources Due to Failure”](#)

- 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).

C.3. 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**. See [Section 5.4.2, “Setting Global Defaults for Resource Meta-Attributes”](#) and [Section 5.5.3, “Setting Global Defaults for Operations”](#) instead.

Appendix D. Installing

Table of Contents

D.1. Installing the Software	141
D.2. Enabling Pacemaker	141
D.2.1. Enabling Pacemaker For Corosync 2.x	141
D.2.2. Enabling Pacemaker For Corosync 1.x	142
D.2.3. Enabling Pacemaker For Heartbeat	143

D.1. Installing the 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](#)¹ for details.

See [Which Messaging Layer Should I Choose?](#) for information about choosing a messaging layer.

D.2. Enabling Pacemaker

D.2.1. Enabling Pacemaker For Corosync 2.x

High-level cluster management tools are available that can configure corosync for you. This document focuses on the lower-level details if you want to configure corosync yourself.

Corosync configuration is normally located in `/etc/corosync/corosync.conf`.

Example D.1. Corosync 2.x configuration file for two nodes **myhost1** and **myhost2**

```
totem {
  version: 2
  secauth: off
  cluster_name: mycluster
  transport: udpu
}

nodelist {
  node {
    ring0_addr: myhost1
    nodeid: 1
  }
  node {
    ring0_addr: myhost2
    nodeid: 2
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}

logging {
  to_syslog: yes
}
```

¹ <http://clusterlabs.org/wiki/Install>

```
}
```

Example D.2. Corosync 2.x configuration file for three nodes **myhost1**, **myhost2** and **myhost3**

```
totem {
  version: 2
  secauth: off
  cluster_name: mycluster
  transport: udpu
}

nodelist {
  node {
    ring0_addr: myhost1
    nodeid: 1
  }
  node {
    ring0_addr: myhost2
    nodeid: 2
  }
  node {
    ring0_addr: myhost3
    nodeid: 3
  }
}

quorum {
  provider: corosync_votequorum
}

logging {
  to_syslog: yes
}
```

In the above examples, the **totem** section defines what protocol version and options (including encryption) to use,² and gives the cluster a unique name (**mycluster** in these examples).

The **node** section lists the nodes in this cluster. (See [Section 4.2, “Where Pacemaker Gets the Node Name”](#) for how this affects pacemaker.)

The **quorum** section defines how the cluster uses quorum. The important thing is that two-node clusters must be handled specially, so **two_node: 1** must be defined for two-node clusters (and only for two-node clusters).

The **logging** section should be self-explanatory.

D.2.2. Enabling Pacemaker For Corosync 1.x

Example D.3. Corosync 1.x configuration file for a cluster with all nodes on the **192.0.2.0/24** network

```
totem {
  version: 2
```

² Please consult the Corosync website (<http://www.corosync.org>) and documentation for details on enabling encryption and peer authentication for the cluster.

```

    secauth: off
    threads: 0
    interface {
        ringnumber: 0
        bindnetaddr: 192.0.2.0
        mcastaddr: 239.255.1.1
        mcastport: 1234
    }
}
logging {
    fileline: off
    to_syslog: yes
    syslog_facility: daemon
}
amf {
    mode: disabled
}

```

With corosync 1.x, the **totem** section contains the protocol version and options as with 2.x. However, nodes are also listed here, in the **interface** section. The **bindnetaddr** option is usually the *network* address, thus allowing the same configuration file to be used on all nodes. IPv4 or IPv6 addresses can be used with corosync.

The **amf** section refers to the Availability Management Framework and is not covered in this document.

The above corosync configuration is enough for corosync to operate by itself, but corosync 1.x additionally needs to be told when it is being used in conjunction with Pacemaker. This can be accomplished in one of two ways:

- Via the CMAN software provided with Red Hat Enterprise Linux 6 and its derivatives
- Via the pacemaker corosync plugin

To use CMAN, consult its documentation.

To use the pacemaker corosync plugin, add the following fragment to the corosync configuration and restart the cluster.

Example D.4. Corosync 1.x configuration fragment to enable Pacemaker plugin

```

aisexec {
    user: root
    group: root
}
service {
    name: pacemaker
    ver: 0
}

```

The cluster needs to be run as root so that its child processes (the **lrmd** in particular) have sufficient privileges to perform the actions requested of it. After all, a cluster manager that can't add an IP address or start apache is of little use.

The second directive is the one that actually instructs the cluster to run Pacemaker.

D.2.3. Enabling Pacemaker For Heartbeat

See the heartbeat documentation for how to set up a **ha.cf** configuration file.

Appendix D. Installing

To enable the use of pacemaker with heartbeat, add the following to a functional **ha.cf** configuration file and restart Heartbeat:

Example D.5. Heartbeat configuration fragment to enable Pacemaker

```
crm respawn
```

Appendix E. Upgrading Cluster Software

Table of Contents

E.1. Complete Cluster Shutdown	145
E.2. Rolling (node by node)	146
E.3. Disconnect and Reattach	146

There will always be an upgrade path from any pacemaker 1.x release to any other 1.y release.

Consult the documentation for your messaging layer (Heartbeat or Corosync) to see whether upgrading them to a newer version is also supported.

There are three approaches to upgrading your cluster software:

- Complete Cluster Shutdown
- Rolling (node by node)
- Disconnect and Reattach

Each method has advantages and disadvantages, some of which are listed in the table below, and you should choose the one most appropriate to your needs.

Table E.1. Upgrade Methods

Type	Available between all software versions	Service Outage During Upgrade	Service Recovery During Upgrade	Exercises Failover Logic/ Configuration	Allows change of cluster stack type ¹
Shutdown	yes	always	N/A	no	yes
Rolling	no	always	yes	yes	no
Reattach	yes	only due to failure	no	no	yes

¹ For example, switching from Heartbeat to Corosync.

E.1. 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 manually or with the **crm_verify** tool if available.

2. On each node:
 - a. Start the cluster software. The messaging layer can be either Corosync or Heartbeat and does not need to be the same one before the upgrade.

E.2. 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.

Rolling upgrades should always be possible for pacemaker versions 1.0.0 and later.

On each node:

1. Put the node into standby mode, and wait for any active resources to be moved cleanly to another node.
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 manually or with the **crm_verify** tool if available.
5. Start the messaging layer. This must be the same messaging layer (Corosync or Heartbeat) that the rest of the cluster is using. Upgrading the messaging layer may also be possible; consult the documentation for those projects to see whether the two versions will be compatible.



Note

Rolling upgrades were not always possible with older heartbeat and pacemaker versions. The table below shows which versions were compatible during rolling upgrades. Rolling upgrades that cross compatibility boundaries must be performed in multiple steps (for example, upgrading heartbeat 2.0.6 to heartbeat 2.1.3, and then upgrading again to pacemaker 0.6.6). Rolling upgrades from pacemaker 0.x to 1.y are not possible.

Table E.2. Version Compatibility Table

Version being Installed	Oldest Compatible Version
Pacemaker 1.0.x	Pacemaker 1.0.0
Pacemaker 0.7.x	Pacemaker 0.6 or Heartbeat 2.1.3
Pacemaker 0.6.x	Heartbeat 2.0.8
Heartbeat 2.1.3 (or less)	Heartbeat 2.0.4
Heartbeat 2.0.4 (or less)	Heartbeat 2.0.0
Heartbeat 2.0.0	None. Use an alternate upgrade strategy.

E.3. Disconnect 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.

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 -t crm_config -n is-managed-default -v false
```

2. For any resource that has a value for **is-managed**, make sure it is set to **false** so that the cluster will not stop it (replacing \$rsc_id appropriately):

```
# crm_resource -t primitive -r $rsc_id -p is-managed -v false
```

3. 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.
4. Check the configuration manually or with the **crm_verify** tool if available.
5. On each node:
 - a. Start the cluster software. The messaging layer can be either Corosync or Heartbeat and does not need to be the same one as before the upgrade.
6. Verify that the cluster re-detected all resources correctly.
7. Allow the cluster to resume managing resources again:

```
# crm_attribute -t crm_config -n is-managed-default -v true
```

8. For any resource that has a value for **is-managed**, reset it to **true** (so the cluster can recover the service if it fails) if desired:

```
# crm_resource -t primitive -r $rsc_id -p is-managed -v true
```



Note

The oldest version of the CRM to support this upgrade type was in Heartbeat 2.0.4.



Important

Always check your existing configuration is still compatible with the version you are installing before starting the cluster.

Appendix F. Upgrading the Configuration

Table of Contents

F.1. Perform the upgrade	149
F.1.1. Upgrade the software	149
F.1.2. Upgrade the Configuration	149
F.1.3. Manually Upgrading the Configuration	150

This process was originally written for the upgrade from 0.6.x to 1.y, but the concepts should apply for any upgrade involving a change in the XML schema version.

F.1. Perform the upgrade

F.1.1. Upgrade the software

Refer to the appendix: [Appendix E, Upgrading Cluster Software](#)

F.1.2. Upgrade the Configuration

As XML is not the friendliest of languages, it is common for cluster administrators to have scripted some of their activities. In such cases, it is likely that those scripts will not work with the new XML syntax.

In order to support such environments, it is actually possible to continue using the old XML syntax.

The downside is, however, that not all the new features will be available 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 advisable to follow the upgrade procedure (except for the last step) to ensure that the cluster is able to use your existing configuration (since it will perform much the same task internally).

1. Create a shadow copy to work with

```
# crm_shadow --create upgrade06
```

2. Verify the configuration is valid

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

3. Fix any errors or warnings

4. Perform the upgrade:

```
# cibadmin --upgrade
```

5. If this step fails, there are three main possibilities:
 - a. The configuration was not valid to start with - go back to step 2
 - b. The transformation failed - report a bug or [email the project](#)¹
 - c. The transformation was successful but produced an invalid result²

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](#)³ and/or try the procedure described below under [Section F.1.3, “Manually Upgrading the Configuration”](#)

6. 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. Since the shadow configuration is not in use by the cluster, it is safe to edit the file manually:

```
# crm_shadow --edit
```

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

7. Preview how the cluster will react:

```
# crm_simulate --live-check --save-dotfile upgrade06.dot -S
# graphviz upgrade06.dot
```

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 [Section 2.5, “Testing Your Configuration Changes”](#) for further details on how to interpret the output of **crm_simulate** and **graphviz**.

8. Upload the changes

```
# crm_shadow --commit upgrade06 --force
```

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

F.1.3. Manually Upgrading the Configuration

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

1. Locate the **upgrade06.xsl** conversion script provided with the source code (the [latest version](#)⁴ is available via git).

¹ <mailto:pacemaker@oss.clusterlabs.org?subject=Transformation%20failed%20during%20upgrade>

² The most common reason is ID values being repeated or invalid. Pacemaker 1.0 is much stricter regarding this type of validation.

³ http://clusterlabs.org/wiki/Validation_FAQ

⁴ <https://github.com/ClusterLabs/pacemaker/tree/master/xml/upgrade06.xsl>

2. Convert the XML blob:

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

3. Locate the **pacemaker.rng** script.
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 should be more informative (despite being generated by the same library!) since they include line numbers.

Appendix G. Init Script LSB Compliance

The relevant part of the [LSB specifications](http://refspecs.linuxfoundation.org/lsg.shtml)¹ 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: $?"
```

- a. Did the service start?
- b. Did the command print **result: 0** (in addition to its usual output)?

2. Status (running):

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

- a. Did the script accept the command?
- b. Did the script indicate the service was running?
- c. Did the command print **result: 0** (in addition to its usual output)?

3. Start (running):

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

- a. Is the service still running?
- b. Did the command print **result: 0** (in addition to its usual output)?

4. Stop (running):

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

- a. Was the service stopped?
- b. Did the command print **result: 0** (in addition to its usual output)?

5. Status (stopped):

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

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

¹ <http://refspecs.linuxfoundation.org/lsg.shtml>

- c. Did the command print **result: 3** (in addition to its usual output)?

6. Stop (stopped):

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

- a. Is the service still stopped?
- b. Did the command print **result: 0** (in addition to its usual output)?

7. Status (failed):

- a. 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.

Appendix H. Sample Configurations

Table of Contents

H.1. Empty	155
H.2. Simple	155
H.3. Advanced Configuration	156

H.1. Empty

Example H.1. An Empty Configuration

```
<cib crm_feature_set="3.0.7" validate-with="pacemaker-1.2" admin_epoch="1" epoch="0"
num_updates="0">
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
```

H.2. Simple

Example H.2. A simple configuration with two nodes, some cluster options and a resource

```
<cib crm_feature_set="3.0.7" validate-with="pacemaker-1.2" admin_epoch="1" epoch="0"
num_updates="0">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="option-1" name="symmetric-cluster" value="true"/>
        <nvpair id="option-2" name="no-quorum-policy" value="stop"/>
        <nvpair id="option-3" name="stonith-enabled" value="0"/>
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="xxx" uname="c001n01" type="normal"/>
      <node id="yyy" uname="c001n02" type="normal"/>
    </nodes>
    <resources>
      <primitive id="myAddr" class="ocf" provider="heartbeat" type="IPAddr">
        <operations>
          <op id="myAddr-monitor" name="monitor" interval="300s"/>
        </operations>
        <instance_attributes id="myAddr-params">
          <nvpair id="myAddr-ip" name="ip" value="192.0.2.10"/>
        </instance_attributes>
      </primitive>
    </resources>
    <constraints>
      <rsc_location id="myAddr-prefer" rsc="myAddr" node="c001n01" score="INFINITY"/>
    </constraints>
    <rsc_defaults>
      <meta_attributes id="rsc_defaults-options">
```

```
<nvpair id="rsc-default-1" name="resource-stickiness" value="100"/>
<nvpair id="rsc-default-2" name="migration-threshold" value="10"/>
</meta_attributes>
</rsc_defaults>
<op_defaults>
  <meta_attributes id="op_defaults-options">
    <nvpair id="op-default-1" name="timeout" value="30s"/>
  </meta_attributes>
</op_defaults>
</configuration>
<status/>
</cib>
```

In the above example, we have one resource (an IP address) that we check every five minutes and will run on host **c001n01** until either the resource fails 10 times or the host shuts down.

H.3. Advanced Configuration

Example H.3. An advanced configuration with groups, clones and STONITH

```
<cib crm_feature_set="3.0.7" validate-with="pacemaker-1.2" admin_epoch="1" epoch="0"
num_updates="0">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="option-1" name="symmetric-cluster" value="true"/>
        <nvpair id="option-2" name="no-quorum-policy" value="stop"/>
        <nvpair id="option-3" name="stonith-enabled" value="true"/>
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="xxx" uname="c001n01" type="normal"/>
      <node id="yyy" uname="c001n02" type="normal"/>
      <node id="zzz" uname="c001n03" type="normal"/>
    </nodes>
    <resources>
      <primitive id="myAddr" class="ocf" provider="heartbeat" type="IPaddr">
        <operations>
          <op id="myAddr-monitor" name="monitor" interval="300s"/>
        </operations>
        <instance_attributes id="myAddr-attrs">
          <nvpair id="myAddr-attr-1" name="ip" value="192.0.2.10"/>
        </instance_attributes>
      </primitive>
      <group id="myGroup">
        <primitive id="database" class="lsb" type="oracle">
          <operations>
            <op id="database-monitor" name="monitor" interval="300s"/>
          </operations>
        </primitive>
        <primitive id="webserver" class="lsb" type="apache">
          <operations>
            <op id="webserver-monitor" name="monitor" interval="300s"/>
          </operations>
        </primitive>
      </group>
      <clone id="STONITH">
        <meta_attributes id="stonith-options">
          <nvpair id="stonith-option-1" name="globally-unique" value="false"/>
        </meta_attributes>
        <primitive id="stonithclone" class="stonith" type="external/ssh">
          <operations>
```



```
        <op id="stonith-op-mon" name="monitor" interval="5s"/>
    </operations>
    <instance_attributes id="stonith-attrs">
        <nvpair id="stonith-attr-1" name="hostlist" value="c001n01,c001n02"/>
    </instance_attributes>
</primitive>
</clone>
</resources>
<constraints>
    <rsc_location id="myAddr-prefer" rsc="myAddr" node="c001n01"
        score="INFINITY"/>
    <rsc_colocation id="group-with-ip" rsc="myGroup" with-rsc="myAddr"
        score="INFINITY"/>
</constraints>
<op_defaults>
    <meta_attributes id="op_defaults-options">
        <nvpair id="op-default-1" name="timeout" value="30s"/>
    </meta_attributes>
</op_defaults>
<rsc_defaults>
    <meta_attributes id="rsc_defaults-options">
        <nvpair id="rsc-default-1" name="resource-stickiness" value="100"/>
        <nvpair id="rsc-default-2" name="migration-threshold" value="10"/>
    </meta_attributes>
</rsc_defaults>
</configuration>
<status/>
</cib>
```

Appendix I. Further Reading

- Project Website: <http://www.clusterlabs.org/>
- Project Documentation: <http://www.clusterlabs.org/wiki/Documentation>
- SUSE High Availability Guide: http://www.suse.com/documentation/sle_ha/book_sleha/data/book_sleha.html
- Heartbeat configuration: <http://www.linux-ha.org/>
- Corosync Configuration: <http://www.corosync.org/>

Appendix J. Revision History

Revision 1-0 **19 Oct 2009**

Andrew Beekhof andrew@beekhof.net

Import from Pages.app

Revision 2-0 **26 Oct 2009**

Andrew Beekhof andrew@beekhof.net

Cleanup and reformatting of docbook xml complete

Revision 3-0 **Tue Nov 12 2009**

Andrew Beekhof andrew@beekhof.net

Split book into chapters and pass validation
Re-organize book for use with [Publican](https://fedorahosted.org/publican/)¹

Revision 4-0 **Mon Oct 8 2012**

Andrew Beekhof andrew@beekhof.net

Converted to [asciidoc](http://www.methods.co.nz/asciidoc)² (which is converted to docbook for use with [Publican](https://fedorahosted.org/publican/)³)

Revision 5-0 **Mon Feb 23 2015**

Ken Gaillot kgaillet@redhat.com

Update for clarity, stylistic consistency and current command-line syntax

Revision 6-0 **Tue Dec 8 2015**

Ken Gaillot kgaillet@redhat.com

Update for Pacemaker 1.1.14

Revision 7-0 **Tue May 3 2016**

Ken Gaillot kgaillet@redhat.com

Update for Pacemaker 1.1.15

¹ <https://fedorahosted.org/publican/>

² <http://www.methods.co.nz/asciidoc>

³ <https://fedorahosted.org/publican/>

Index

Symbols

0

OCF_SUCCESS, 136

1

OCF_ERR_GENERIC, 137

2

OCF_ERR_ARGS, 137

3

OCF_ERR_UNIMPLEMENTED, 137

4

OCF_ERR_PERM, 137

5

OCF_ERR_INSTALLED, 137

6

OCF_ERR_CONFIGURED, 137

7

OCF_NOT_RUNNING, 137

8

OCF_RUNNING_MASTER, 137

9

OCF_FAILED_MASTER, 137

A

Action, 36

demote, 136

meta-data, 135

monitor, 135

notify, 136

promote, 136

Property

enabled, 37, 37

id, 36

interval, 36

name, 36

on-fail, 37

role, 37

timeout, 36

start, 135

Status

call-id, 123

crm-debug-origin, 124

crm_feature_set, 124

exec-time, 123

id, 123

interval, 123

last-rc-change, 123

last-run, 123

op-digest, 124

op-status, 123

operation, 123

queue-time, 123

rc-code, 123

transition-key, 124

transition-magic, 124

stop, 135

validate-all, 135

action, 48, 90

Ordering Constraints, 90

Resource Sets, 48

Action Property, 36, 36, 36, 36, 37, 37, 37, 37

Action Status, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 124, 124, 124, 124, 124

active_resource, 86, 92

Notification Environment Variable, 86, 92

active_uname, 86, 92

Notification Environment Variable, 86, 92

Add Cluster Node, 24, 25

Corosync, 24

Heartbeat, 25

admin_epoch, 17

Cluster Option, 17

Alert

Option

timeout, 56

timestamp-format, 56

Alerts, 55

Asymmetrical Opt-In, 43

Asymmetrical Opt-In Clusters, 43

attribute, 24, 62

Constraint Expression, 62

Attribute Expression, 62

attribute, 62

operation, 62

type, 62

value, 62

B

batch-limit, 19

Cluster Option, 19

boolean-op, 61

Constraint Rule, 61

C

call-id, 123

Action Status, 123

Changing Cluster Stack, 145

Choosing Between Heartbeat and Corosync, 133

cib-last-written, 17

Cluster Property, 17

CIB_encrypted, 69

CIB_passwd, 69

CIB_port, 69

CIB_server, 69

CIB_user, 69

- class, 27, 31
 - Resource, 31
- Clone
 - Option
 - clone-max, 84
 - clone-min, 84
 - clone-node-max, 84
 - globally-unique, 84
 - interleave, 84
 - notify, 84
 - ordered, 84
 - Property
 - id, 84
- Clone Option, 84, 84, 84, 84, 84, 84, 84
- Clone Property, 84
- Clone Resources, 83
- clone-max, 84
 - Clone Option, 84
- clone-min, 84
 - Clone Option, 84
- clone-node-max, 84
 - Clone Option, 84
- Clones, 83, 85
- Cluster, 17
 - Choosing Between Heartbeat and Corosync, 133
 - Option
 - admin_epoch, 17
 - batch-limit, 19
 - cluster-delay, 20
 - cluster-recheck-interval, 20
 - concurrent-fencing, 19
 - Configuration Version, 17
 - crmd-finalization-timeout, 20
 - crmd-integration-timeout, 20
 - crmd-transition-delay, 20
 - dc-deadtime, 20
 - default-action-timeout, 21
 - default-resource-stickiness, 21
 - election-timeout, 20
 - enable-startup-probes, 19
 - epoch, 17
 - is-managed-default, 21
 - maintenance-mode, 19
 - migration-limit, 19
 - no-quorum-policy, 18
 - num_updates, 17
 - pe-error-series-max, 20
 - pe-input-series-max, 20
 - pe-warn-series-max, 20
 - remove-after-stop, 20
 - shutdown-escalation, 20
 - start-failure-is-fatal, 19
 - startup-fencing, 20
 - stonith-action, 19
 - stonith-enabled, 19
 - stonith-timeout, 19
 - stop-all-resources, 19
 - stop-orphan-actions, 19
 - stop-orphan-resources, 19
 - symmetric-cluster, 19
 - validate-with, 17
 - Property
 - cib-last-written, 17
 - cluster-infrastructure, 18
 - dc-uuid, 17
 - dc-version, 18
 - expected-quorum-votes, 18
 - have-quorum, 17
 - Querying Options, 21
 - Remote administration, 69
 - Remote connection, 69
 - Setting Options, 21
 - Setting Options with Rules, 68
 - Switching between Stacks, 145
- Cluster Option, 17, 17, 17, 17, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21
- Cluster Property, 17, 17, 17, 18, 18, 18
- Cluster Stack
 - Corosync, 133
 - Heartbeat, 133
- Cluster Type
 - Asymmetrical Opt-In, 43
 - Symmetrical Opt-Out, 44
- cluster-delay, 20
 - Cluster Option, 20
- cluster-infrastructure, 18
 - Cluster Property, 18
- cluster-recheck-interval, 20
 - Cluster Option, 20
- Colocation, 46
 - id, 46
 - rsc, 46
 - score, 47
 - with-rsc, 47
- Colocation Constraints, 46, 46, 47, 47
- concurrent-fencing, 19
 - Cluster Option, 19
- Configuration, 105, 149
 - Upgrade manually, 150
 - Upgrading, 149
 - Validate XML, 151
 - Verify, 149
- Configuration Version, 17
 - Cluster, 17
- Constraint

- Attribute Expression, 62
 - attribute, 62
 - operation, 62
 - type, 62
 - value, 62
- Date Specification, 63
 - hours, 63
 - id, 63
 - monthdays, 64
 - months, 64
 - moon, 64
 - weekdays, 64
 - weeks, 64
 - weekyears, 64
 - yeardays, 64
 - years, 64
- Date/Time Expression, 63
 - end, 63
 - operation, 63
 - start, 63
- Duration, 64
- Rule, 61
 - boolean-op, 61
 - role, 61
 - score, 61
 - score-attribute, 61
- Constraint Expression, 62, 62, 62, 62, 63, 63, 63
- Constraint Rule, 61, 61, 61, 61
- Constraints, 41
 - Colocation, 46
 - id, 46
 - rsc, 46
 - score, 47
 - with-rsc, 47
 - Location, 42
 - id, 42
 - node, 42
 - Resource Discovery, 43
 - rsc, 42
 - score, 42
 - Ordering, 44
 - action, 90
 - first, 45
 - first-action, 45
 - id, 45
 - kind, 45
 - role, 90
 - rsc-role, 89
 - then, 45
 - then-action, 45
 - with-rsc-role, 89
 - Resource Sets
 - action, 48
 - id, 48
 - require-all, 48
 - role, 48
 - score, 48
 - sequential, 48
- Controlling Cluster Options, 68
- Convert, 151
- Corosync, 24, 25, 25, 133, 133
 - Add Cluster Node, 24
 - Remove Cluster Node, 25
 - Replace Cluster Node, 25
- crm-debug-origin, 122, 124
 - Action Status, 124
 - Node Status, 122
- crmd, 122
 - Node Status, 122
- crmd-finalization-timeout, 20
 - Cluster Option, 20
- crmd-integration-timeout, 20
 - Cluster Option, 20
- crmd-transition-delay, 20
 - Cluster Option, 20
- CRM_alert_
 - desc, 58
 - interval, 58
 - kind, 58
 - node, 58
 - nodeid, 58
 - rc, 58
 - recipient, 58
 - rsc, 58
 - status, 58
 - target_rc, 58
 - task, 58
 - timestamp, 58
 - version, 58
- CRM_alert_node_
 - sequence, 58
- crm_feature_set, 124
 - Action Status, 124

D

- dampen, 73
 - Ping Resource Option, 73
- Date Specification, 63, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64
 - hours, 63
 - id, 63
 - monthdays, 64
 - months, 64
 - moon, 64
 - weekdays, 64
 - weeks, 64
 - weekyears, 64
 - yeardays, 64

- years, 64
- Date/Time Expression, 63
 - end, 63
 - operation, 63
 - start, 63
- dc-deadtime, 20
 - Cluster Option, 20
- dc-uuid, 17
 - Cluster Property, 17
- dc-version, 18
 - Cluster Property, 18
- default-action-timeout, 21
 - Cluster Option, 21
- default-resource-stickiness, 21
 - Cluster Option, 21
- demote, 136
 - OCF Action, 136
- demote_resource, 92
 - Notification Environment Variable, 92
- demote_uname, 92
 - Notification Environment Variable, 92
- desc, 58
- Determine by Rules, 66
- Determine Resource Location, 66
- devices, 114
 - fencing-level, 114
- Duration, 64, 64

E

- election-timeout, 20
 - Cluster Option, 20
- enable-startup-probes, 19
 - Cluster Option, 19
- enabled, 37, 37
 - Action Property, 37, 37
- end, 63
 - Constraint Expression, 63
- Environment Variable
 - CIB_encrypted, 69
 - CIB_passwd, 69
 - CIB_port, 69
 - CIB_server, 69
 - CIB_user, 69
 - CRM_alert_
 - desc, 58
 - interval, 58
 - kind, 58
 - node, 58
 - nodeid, 58
 - rc, 58
 - recipient, 58
 - rsc, 58
 - status, 58
 - target_rc, 58

- task, 58
- timestamp, 58
- version, 58
- CRM_alert_node_
 - sequence, 58
- OCF_RESKEY_CRM_meta_notify_
 - active_resource, 86, 92
 - active_uname, 86, 92
 - demote_resource, 92
 - demote_uname, 92
 - inactive_resource, 86, 92
 - inactive_uname, 86, 92
 - master_resource, 92
 - master_uname, 92
 - operation, 86, 92
 - promote_resource, 92
 - promote_uname, 92
 - slave_resource, 92
 - slave_uname, 92
 - start_resource, 86, 92
 - start_uname, 86, 92
 - stop_resource, 86, 92
 - stop_uname, 86, 92
 - type, 86, 92
- epoch, 17
 - Cluster Option, 17
- error
 - fatal, 136
 - hard, 136
 - soft, 136
- exec-time, 123
 - Action Status, 123
- expected, 122
 - Node Status, 122
- expected-quorum-votes, 18
 - Cluster Property, 18

F

- failure-timeout, 33, 72
 - Resource Option, 33
- fatal, 136
 - OCF error, 136
- feedback
 - contact information for this manual, xix
- Fencing, 106, 106, 106, 107, 107, 107, 107, 107, 107, 107, 108, 108, 108, 108, 108, 108, 108, 108, 108, 108, 109, 109, 109, 109, 109, 109, 109
- fencing-level
 - devices, 114
 - id, 113
 - index, 114
 - target, 113
 - target-attribute, 113, 113
 - target-pattern, 113

-
- Property
 - pcmk_action_limit, 107
 - pcmk_delay_max, 107
 - pcmk_host_argument, 107
 - pcmk_host_check, 107
 - pcmk_host_list, 107
 - pcmk_host_map, 106
 - pcmk_list_action, 108
 - pcmk_list_retries, 108
 - pcmk_list_timeout, 108
 - pcmk_monitor_action, 109
 - pcmk_monitor_retries, 109
 - pcmk_monitor_timeout, 109
 - pcmk_off_action, 108
 - pcmk_off_retries, 108
 - pcmk_off_timeout, 108
 - pcmk_reboot_action, 107
 - pcmk_reboot_retries, 108
 - pcmk_reboot_timeout, 107
 - pcmk_status_action, 109
 - pcmk_status_retries, 109
 - pcmk_status_timeout, 109
 - priority, 106
 - stonith-timeout, 106, 107, 108
 - fencing-level, 113, 113, 113, 113, 113, 114, 114
 - devices, 114
 - id, 113
 - index, 114
 - target, 113
 - target-attribute, 113, 113
 - target-pattern, 113
 - first, 45
 - Ordering Constraints, 45
 - first-action, 45
 - Ordering Constraints, 45
- ## G
- globally-unique, 84
 - Clone Option, 84
 - Group Property
 - id, 82
 - Group Resource Property, 82
 - Group Resources, 81
 - Groups, 81, 83
- ## H
- ha, 122
 - Node Status, 122
 - hard, 136
 - OCF error, 136
 - have-quorum, 17
 - Cluster Property, 17
 - Heartbeat, 25, 26, 26, 133, 133
 - Add Cluster Node, 25
 - Remove Cluster Node, 26
 - Replace Cluster Node, 26
 - host_list, 73
 - Ping Resource Option, 73
 - hours, 63
 - Date Specification, 63
- ## I
- id, 31, 36, 42, 45, 46, 48, 63, 82, 84, 88, 113, 121, 123
 - Action Property, 36
 - Action Status, 123
 - Clone Property, 84
 - Colocation Constraints, 46
 - Date Specification, 63
 - fencing-level, 113
 - Group Resource Property, 82
 - Location Constraints, 42
 - Multi-State Property, 88
 - Node Status, 121
 - Ordering Constraints, 45
 - Resource, 31
 - Resource Sets, 48
 - inactive_resource, 86, 92
 - Notification Environment Variable, 86, 92
 - inactive_uname, 86, 92
 - Notification Environment Variable, 86, 92
 - index, 114
 - fencing-level, 114
 - interleave, 84
 - Clone Option, 84
 - interval, 36, 58, 123
 - Action Property, 36
 - Action Status, 123
 - in_ccm, 122
 - Node Status, 122
 - is-managed, 32
 - Resource Option, 32
 - is-managed-default, 21
 - Cluster Option, 21
- ## J
- join, 122
 - Node Status, 122
- ## K
- kind, 45, 58
 - Ordering Constraints, 45
- ## L
- last-rc-change, 123
 - Action Status, 123

OCF Action, 135, 135, 135, 135, 135, 136, 136, 136
 OCF error, 136, 136, 136
 OCF Resource Agents, 135
 ocf-tester, 136
 OCF_ERR_ARGS, 137, 137
 OCF_ERR_CONFIGURED, 137, 137
 OCF_ERR_GENERIC, 137, 137
 OCF_ERR_INSTALLED, 137, 137
 OCF_ERR_PERM, 137, 137
 OCF_ERR_UNIMPLEMENTED, 137, 137
 OCF_FAILED_MASTER, 91, 137, 137
 OCF_NOT_RUNNING, 91, 137, 137
 OCF_RESKEY_CRM_meta_notify_
 active_resource, 86, 92
 active_uname, 86, 92
 demote_resource, 92
 demote_uname, 92
 inactive_resource, 86, 92
 inactive_uname, 86, 92
 master_resource, 92
 master_uname, 92
 operation, 86, 92
 promote_resource, 92
 promote_uname, 92
 slave_resource, 92
 slave_uname, 92
 start_resource, 86, 92
 start_uname, 86, 92
 stop_resource, 86, 92
 stop_uname, 86, 92
 type, 86, 92
 OCF_RUNNING_MASTER, 91, 137, 137
 OCF_SUCCESS, 91, 136, 136
 on-fail, 37
 Action Property, 37
 op-digest, 124
 Action Status, 124
 op-status, 123
 Action Status, 123
 Open Cluster Framework
 Resources, 28
 operation, 62, 63, 86, 92, 123
 Action Status, 123
 Constraint Expression, 62, 63
 Notification Environment Variable, 86, 92
 Operation History, 122
 Option
 admin_epoch, 17
 batch-limit, 19
 clone-max, 84
 clone-min, 84
 clone-node-max, 84
 cluster-delay, 20
 cluster-recheck-interval, 20
 concurrent-fencing, 19
 Configuration Version, 17
 crmd-finalization-timeout, 20
 crmd-integration-timeout, 20
 crmd-transition-delay, 20
 dampen, 73
 dc-deadtime, 20
 default-action-timeout, 21
 default-resource-stickiness, 21
 election-timeout, 20
 enable-startup-probes, 19
 epoch, 17
 failure-timeout, 33
 globally-unique, 84
 host_list, 73
 interleave, 84
 is-managed, 32
 is-managed-default, 21
 maintenance-mode, 19
 master-max, 88
 master-node-max, 88
 migration-limit, 19
 migration-threshold, 33
 multiple-active, 33
 multiplier, 73
 no-quorum-policy, 18
 notify, 84
 num_updates, 17
 ordered, 84
 pe-error-series-max, 20
 pe-input-series-max, 20
 pe-warn-series-max, 20
 priority, 32
 remote-clear-port, 69
 remote-tls-port, 69
 remove-after-stop, 20
 requires, 32
 resource-stickiness, 32
 shutdown-escalation, 20
 start-failure-is-fatal, 19
 startup-fencing, 20
 stonith-action, 19
 stonith-enabled, 19
 stonith-timeout, 19
 stop-all-resources, 19
 stop-orphan-actions, 19
 stop-orphan-resources, 19
 symmetric-cluster, 19
 target-role, 32
 timeout, 56
 timestamp-format, 56
 validate-with, 17
 ordered, 84

- Clone Option, 84
- Ordering, 44
 - action, 90
 - first, 45
 - first-action, 45
 - id, 45
 - kind, 45
 - role, 90
 - rsc-role, 89
 - then, 45
 - then-action, 45
 - with-rsc-role, 89
- Ordering Constraints, 44, 45, 45, 45, 45, 45, 45, 45, 89, 89, 90, 90
 - symmetrical, 45
- other, 137

P

- Pacemaker, 133
- pcmk_action_limit, 107
 - Fencing, 107
- pcmk_delay_max, 107
 - Fencing, 107
- pcmk_host_argument, 107
 - Fencing, 107
- pcmk_host_check, 107
 - Fencing, 107
- pcmk_host_list, 107
 - Fencing, 107
- pcmk_host_map, 106
 - Fencing, 106
- pcmk_list_action, 108
 - Fencing, 108
- pcmk_list_retries, 108
 - Fencing, 108
- pcmk_list_timeout, 108
 - Fencing, 108
- pcmk_monitor_action, 109
 - Fencing, 109
- pcmk_monitor_retries, 109
 - Fencing, 109
- pcmk_monitor_timeout, 109
 - Fencing, 109
- pcmk_off_action, 108
 - Fencing, 108
- pcmk_off_retries, 108
 - Fencing, 108
- pcmk_off_timeout, 108
 - Fencing, 108
- pcmk_reboot_action, 107
 - Fencing, 107
- pcmk_reboot_retries, 108
 - Fencing, 108
- pcmk_reboot_timeout, 107
 - Fencing, 107
- pcmk_status_action, 109
 - Fencing, 109
- pcmk_status_retries, 109
 - Fencing, 109
- pcmk_status_timeout, 109
 - Fencing, 109
- pe-error-series-max, 20
 - Cluster Option, 20
- pe-input-series-max, 20
 - Cluster Option, 20
- pe-warn-series-max, 20
 - Cluster Option, 20
- Ping Resource
 - Option
 - dampen, 73
 - host_list, 73
 - multiplier, 73
- Ping Resource Option, 73, 73, 73
- priority, 32, 106
 - Fencing, 106
 - Resource Option, 32
- promote, 136
 - OCF Action, 136
- promote_resource, 92
 - Notification Environment Variable, 92
- promote_undef, 92
 - Notification Environment Variable, 92
- Property
 - cib-last-written, 17
 - class, 31
 - cluster-infrastructure, 18
 - dc-uuid, 17
 - dc-version, 18
 - enabled, 37, 37
 - expected-quorum-votes, 18
 - have-quorum, 17
 - id, 31, 36, 84, 88
 - interval, 36
 - name, 36
 - on-fail, 37
 - pcmk_action_limit, 107
 - pcmk_delay_max, 107
 - pcmk_host_argument, 107
 - pcmk_host_check, 107
 - pcmk_host_list, 107
 - pcmk_host_map, 106
 - pcmk_list_action, 108
 - pcmk_list_retries, 108
 - pcmk_list_timeout, 108
 - pcmk_monitor_action, 109
 - pcmk_monitor_retries, 109
 - pcmk_monitor_timeout, 109
 - pcmk_off_action, 108

- pcmk_off_retries, 108
- pcmk_off_timeout, 108
- pcmk_reboot_action, 107
- pcmk_reboot_retries, 108
- pcmk_reboot_timeout, 107
- pcmk_status_action, 109
- pcmk_status_retries, 109
- pcmk_status_timeout, 109
- priority, 106
- provider, 31
- role, 37
- stonith-timeout, 106, 107, 108
- timeout, 36
- type, 31
- provider, 31
 - Resource, 31

Q

- Querying
 - Cluster Option, 21
- Querying Options, 21
- queue-time, 123
 - Action Status, 123

R

- rc, 58
- rc-code, 123
 - Action Status, 123
- Reattach, 145
- Reattach Upgrade, 145
- recipient, 58
- Remote administration, 69
- Remote connection, 69
- Remote Connection
 - Option
 - remote-clear-port, 69
 - remote-tls-port, 69
- Remote Connection Option, 69, 69
- remote-clear-port, 69
 - Remote Connection Option, 69
- remote-tls-port, 69
 - Remote Connection Option, 69
- Remove Cluster Node, 25, 26
 - Corosync, 25
 - Heartbeat, 26
- remove-after-stop, 20
 - Cluster Option, 20
- Replace Cluster Node, 25, 26
 - Corosync, 25
 - Heartbeat, 26
- require-all, 48
 - Resource Sets, 48
- requires, 32

- Resource Option, 32
- Resource, 27, 31, 31, 31, 31
 - Action, 36
 - Alerts, 55
 - class, 27
 - Constraint
 - Attribute Expression, 62
 - Date Specification, 63
 - Date/Time Expression, 63
 - Duration, 64
 - Rule, 61
 - Constraints, 41
 - Colocation, 46
 - Location, 42
 - Ordering, 44
 - Group Property
 - id, 82
 - Location
 - Determine by Rules, 66
 - Location Relative to other Resources, 46
 - LSB, 28
 - Moving, 70
 - Nagios Plugins, 30
 - OCF, 28
 - Option
 - failure-timeout, 33
 - is-managed, 32
 - migration-threshold, 33
 - multiple-active, 33
 - priority, 32
 - requires, 32
 - resource-stickiness, 32
 - target-role, 32
 - Property
 - class, 31
 - id, 31
 - provider, 31
 - type, 31
 - Start Order, 44
 - STONITH, 30
 - System Services, 30
 - Systemd, 29
 - Upstart, 29
- Resource Discovery, 43
 - Location Constraints, 43
- Resource Option, 32, 32, 32, 32, 32, 33, 33, 33
- Resource Sets, 48, 48, 48, 48, 48, 48
 - action, 48
 - id, 48
 - require-all, 48
 - role, 48
 - score, 48
 - sequential, 48
- resource-stickiness, 32

- Clones, 85
- Groups, 83
- Multi-State, 91
- Resource Option, 32
- Resources, 28, 28, 28, 28, 29, 29, 30, 30, 30, 70
 - Clones, 83
 - Groups, 81
 - Multi-state, 87
- Return Code
 - 0
 - OCF_SUCCESS, 136
 - 1
 - OCF_ERR_GENERIC, 137
 - 2
 - OCF_ERR_ARGS, 137
 - 3
 - OCF_ERR_UNIMPLEMENTED, 137
 - 4
 - OCF_ERR_PERM, 137
 - 5
 - OCF_ERR_INSTALLED, 137
 - 6
 - OCF_ERR_CONFIGURED, 137
 - 7
 - OCF_NOT_RUNNING, 137
 - 8
 - OCF_RUNNING_MASTER, 137
 - 9
 - OCF_FAILED_MASTER, 137
 - OCF_ERR_ARGS, 137
 - OCF_ERR_CONFIGURED, 137
 - OCF_ERR_GENERIC, 137
 - OCF_ERR_INSTALLED, 137
 - OCF_ERR_PERM, 137
 - OCF_ERR_UNIMPLEMENTED, 137
 - OCF_FAILED_MASTER, 91, 137
 - OCF_NOT_RUNNING, 91, 137
 - OCF_RUNNING_MASTER, 91, 137
 - OCF_SUCCESS, 91, 136
 - other, 137
- role, 37, 48, 61, 90
 - Action Property, 37
 - Constraint Rule, 61
 - Ordering Constraints, 90
 - Resource Sets, 48
- Rolling, 145
- Rolling Upgrade, 145
- rsc, 42, 46, 58
 - Colocation Constraints, 46
 - Location Constraints, 42
- rsc-role, 89
 - Ordering Constraints, 89
- Rule, 61
 - boolean-op, 61

- Controlling Cluster Options, 68
- Determine Resource Location, 66
- role, 61
- score, 61
- score-attribute, 61

S

- score, 42, 47, 48, 61
 - Colocation Constraints, 47
 - Constraint Rule, 61
 - Location Constraints, 42
 - Resource Sets, 48
- score-attribute, 61
 - Constraint Rule, 61
- sequence, 58
- sequential, 48
 - Resource Sets, 48
- Setting
 - Cluster Option, 21
- Setting Options, 21
- Setting Options with Rules, 68
- Shutdown, 145
- Shutdown Upgrade, 145
- shutdown-escalation, 20
 - Cluster Option, 20
- slave_resource, 92
 - Notification Environment Variable, 92
- slave_uname, 92
 - Notification Environment Variable, 92
- soft, 136
 - OCF error, 136
- start, 63, 135
 - Constraint Expression, 63
 - OCF Action, 135
- Start Order, 44
- start-failure-is-fatal, 19, 72
 - Cluster Option, 19
- startup-fencing, 20
 - Cluster Option, 20
- start_resource, 86, 92
 - Notification Environment Variable, 86, 92
- start_uname, 86, 92
 - Notification Environment Variable, 86, 92
- status, 58
- Status, 121
 - call-id, 123
 - crm-debug-origin, 122, 124
 - crmd, 122
 - crm_feature_set, 124
 - exec-time, 123
 - expected, 122
 - ha, 122
 - id, 121, 123
 - interval, 123

- in_ccm, 122
- join, 122
- last-rc-change, 123
- last-run, 123
- op-digest, 124
- op-status, 123
- operation, 123
- queue-time, 123
- rc-code, 123
- transition-key, 124
- transition-magic, 124
- uname, 121
- Status of a Node, 121
- STONITH, 30
 - Configuration, 105
 - Resources, 30
- stonith-action, 19
 - Cluster Option, 19
- stonith-enabled, 19
 - Cluster Option, 19
- stonith-timeout, 19, 106, 107, 108
 - Cluster Option, 19
 - Fencing, 106, 107, 108
- stop, 135
 - OCF Action, 135
- stop-all-resources, 19
 - Cluster Option, 19
- stop-orphan-actions, 19
 - Cluster Option, 19
- stop-orphan-resources, 19
 - Cluster Option, 19
- stop_resource, 86, 92
 - Notification Environment Variable, 86, 92
- stop_uname, 86, 92
 - Notification Environment Variable, 86, 92
- Switching between Stacks, 145
- symmetric-cluster, 19
 - Cluster Option, 19
- symmetrical, 45
 - Ordering Constraints, 45
- Symmetrical Opt-Out, 44
- Symmetrical Opt-Out Clusters, 44
- System Service
 - Resources, 30
- System Services, 30
- Systemd, 29
 - Resources, 29

T

- target, 113
 - fencing-level, 113
- target-attribute, 113, 113
 - fencing-level, 113, 113
- target-pattern, 113

- fencing-level, 113
- target-role, 32
 - Resource Option, 32
- target_rc, 58
- task, 58
- then, 45
 - Ordering Constraints, 45
- then-action, 45
 - Ordering Constraints, 45
- Time Based Expressions, 63
- timeout, 36, 56
 - Action Property, 36
- timestamp, 58
- timestamp-format, 56
- transition-key, 124
 - Action Status, 124
- transition-magic, 124
 - Action Status, 124
- type, 31, 62, 86, 92
 - Constraint Expression, 62
 - Notification Environment Variable, 86, 92
 - Resource, 31

U

- uname, 121
 - Node Status, 121
- Upgrade
 - Reattach, 145
 - Rolling, 145
 - Shutdown, 145
- Upgrade manually, 150
- Upgrading, 149
- Upgrading the Configuration, 149
- Upstart, 29
 - Resources, 29

V

- Validate Configuration, 151
- Validate XML, 151
- validate-all, 135
 - OCF Action, 135
- validate-with, 17
 - Cluster Option, 17
- value, 62
 - Constraint Expression, 62
- Verify, 149
 - Configuration, 149
- version, 58

W

- weekdays, 64
 - Date Specification, 64
- weeks, 64

- Date Specification, 64
- weekyears, 64
 - Date Specification, 64
- with-rsc, 47
 - Colocation Constraints, 47
- with-rsc-role, 89
 - Ordering Constraints, 89

X

- XML
 - Convert, 151

Y

- yeardays, 64
 - Date Specification, 64
- years, 64
 - Date Specification, 64