# Pacemaker Explained

*Release 2.1.9*

the Pacemaker project contributors

Nov 04, 2024

# CONTENTS

*Configuring Pacemaker Clusters*

CHAPTER

# ONE

## ABSTRACT

This document definitively explains Pacemaker's features and capabilities, particularly the XML syntax used in Pacemaker's Cluster Information Base (CIB).

CHAPTER

TWO

# TABLE OF CONTENTS

## 2.1 Introduction

### 2.1.1 The Scope of this Document

This document is intended to be an exhaustive reference for configuring Pacemaker. To achieve this, it focuses on the XML syntax used to configure the CIB.

For those that are allergic to XML, multiple higher-level front-ends (both command-line and GUI) are available. These tools will not be covered in this document, though the concepts explained here should make the functionality of these tools more easily understood.

Users may be interested in other parts of the Pacemaker documentation set, such as *Clusters from Scratch*, a step-by-step guide to setting up an example cluster, and *Pacemaker Administration*, a guide to maintaining a cluster.

### 2.1.2 What Is Pacemaker?

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

Pacemaker's key features include:

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

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

- Unified, scriptable cluster management tools

**Note: Fencing**

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

Pacemaker represents fence devices as a special class of resource.

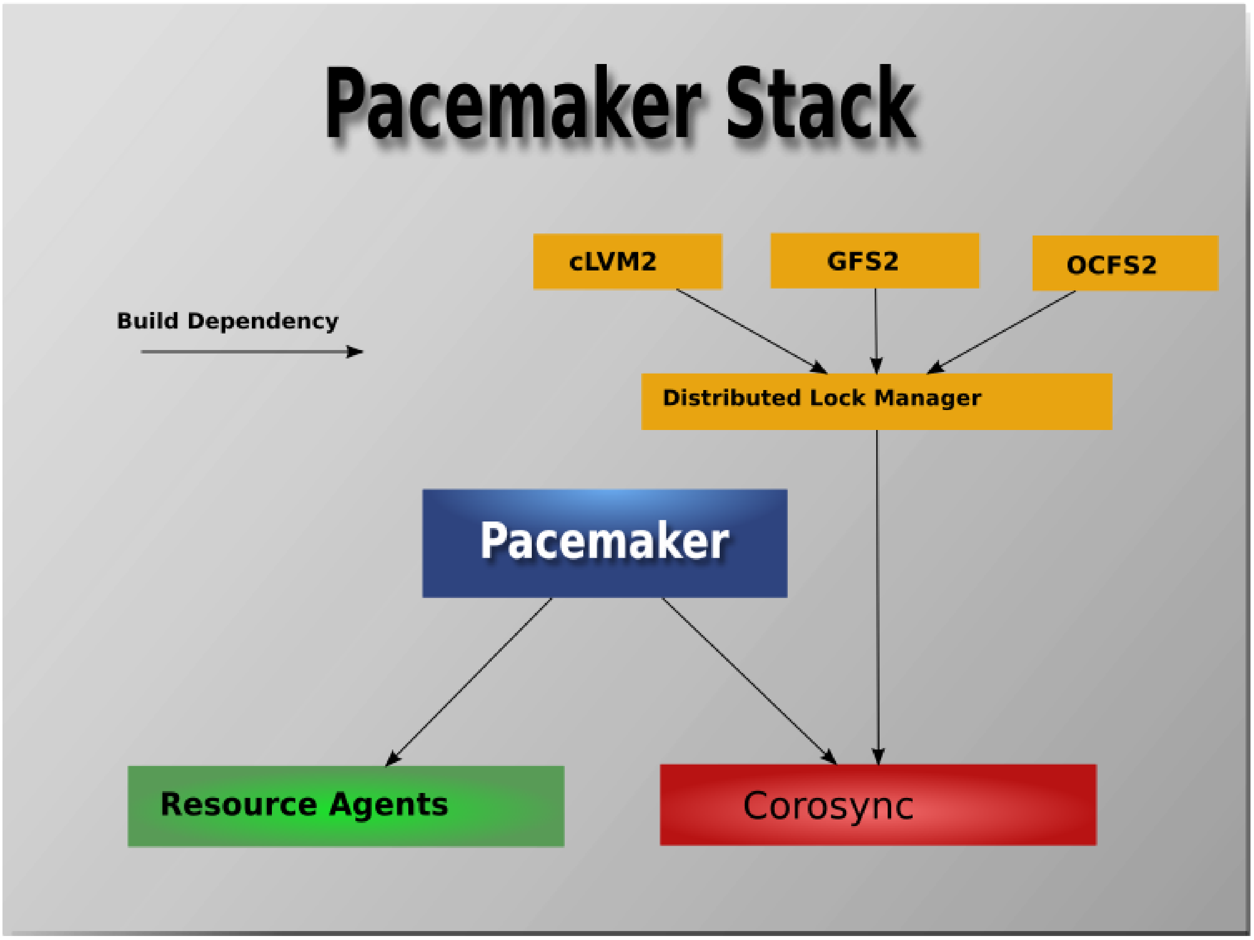
A cluster cannot safely recover from certain failure conditions, such as an unresponsive node, without fencing.

### Cluster Architecture

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

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

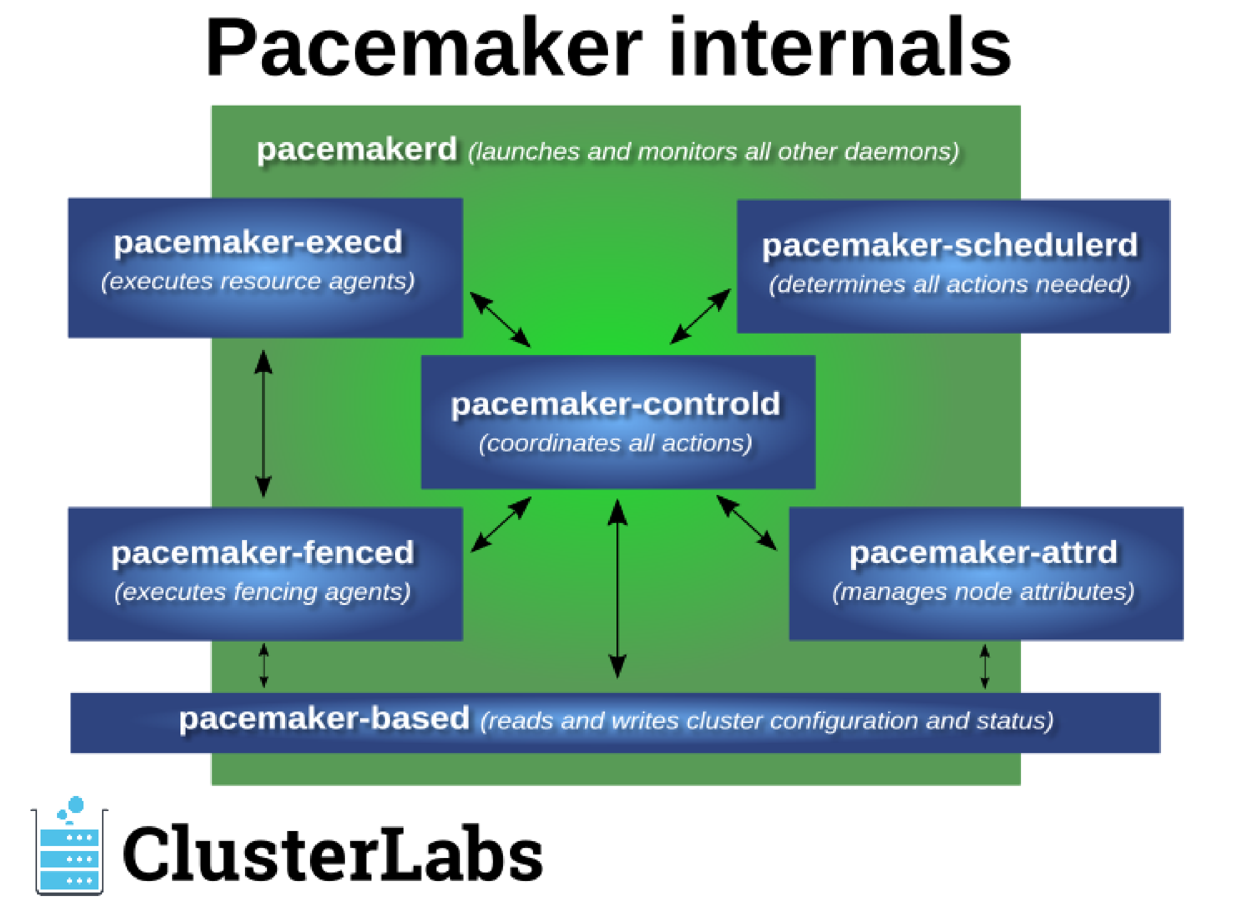
Most managed services are not, themselves, cluster-aware. However, many popular open-source cluster filesystems make use of a common *Distributed Lock Manager* (DLM), which makes direct use of Corosync for its messaging and membership capabilities and Pacemaker for the ability to fence nodes.




### Pacemaker Architecture

Pacemaker itself is composed of multiple daemons that work together:

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




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

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

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

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

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

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

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

Pacemaker centralizes cluster decision-making by electing one of the controller instances as the *Designated Controller* (*DC*). Should the elected DC process (or the node it is on) fail, a new one is quickly established. The DC responds to cluster events by taking a current snapshot of the CIB, feeding it to the scheduler, then asking the executors (either directly on the local node, or via requests to controller peers on other nodes) and the fencer to execute any necessary actions.

**Note: Old daemon names**

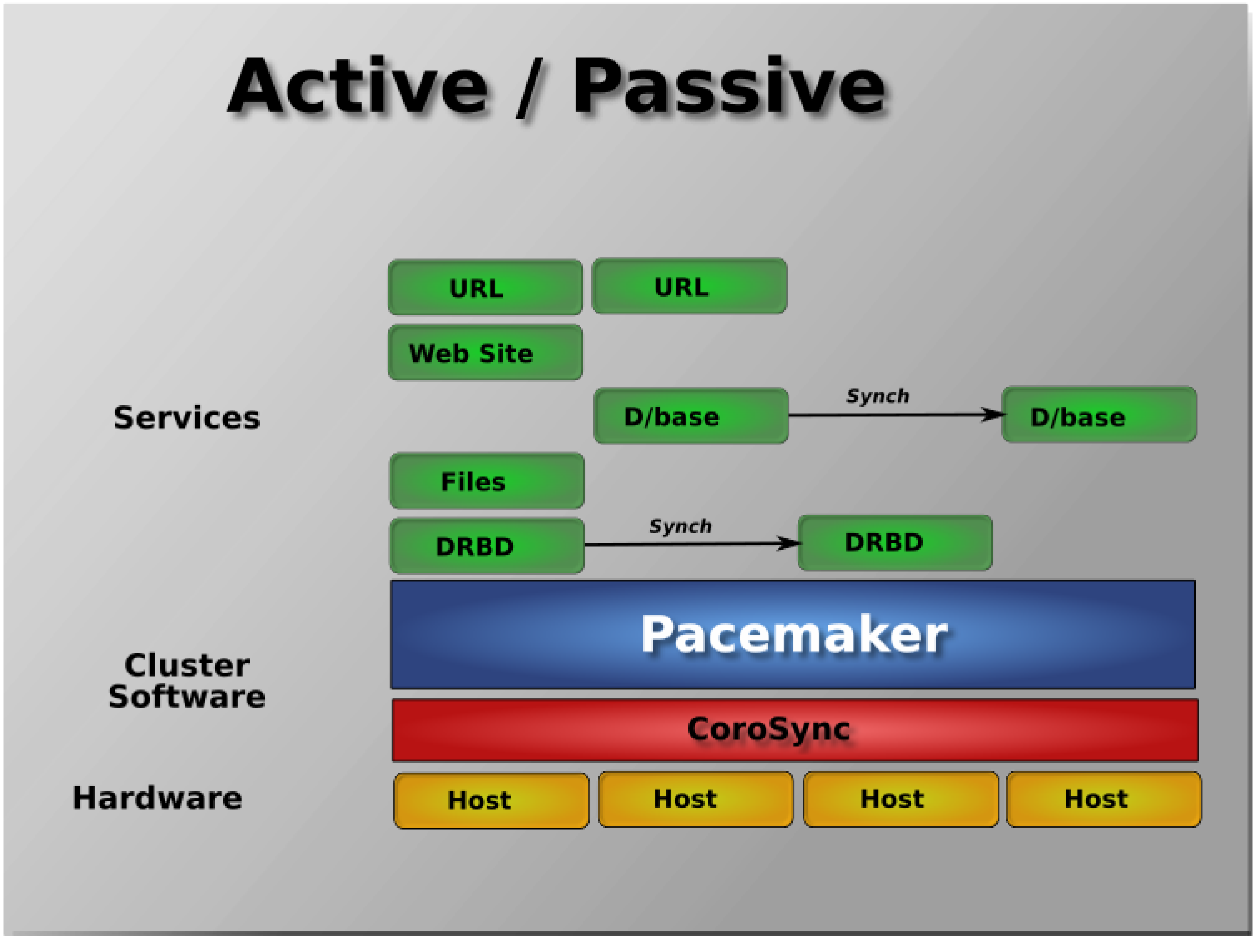
The Pacemaker daemons were renamed in version 2.0. You may still find references to the old names, especially in documentation targeted to version 1.1.

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

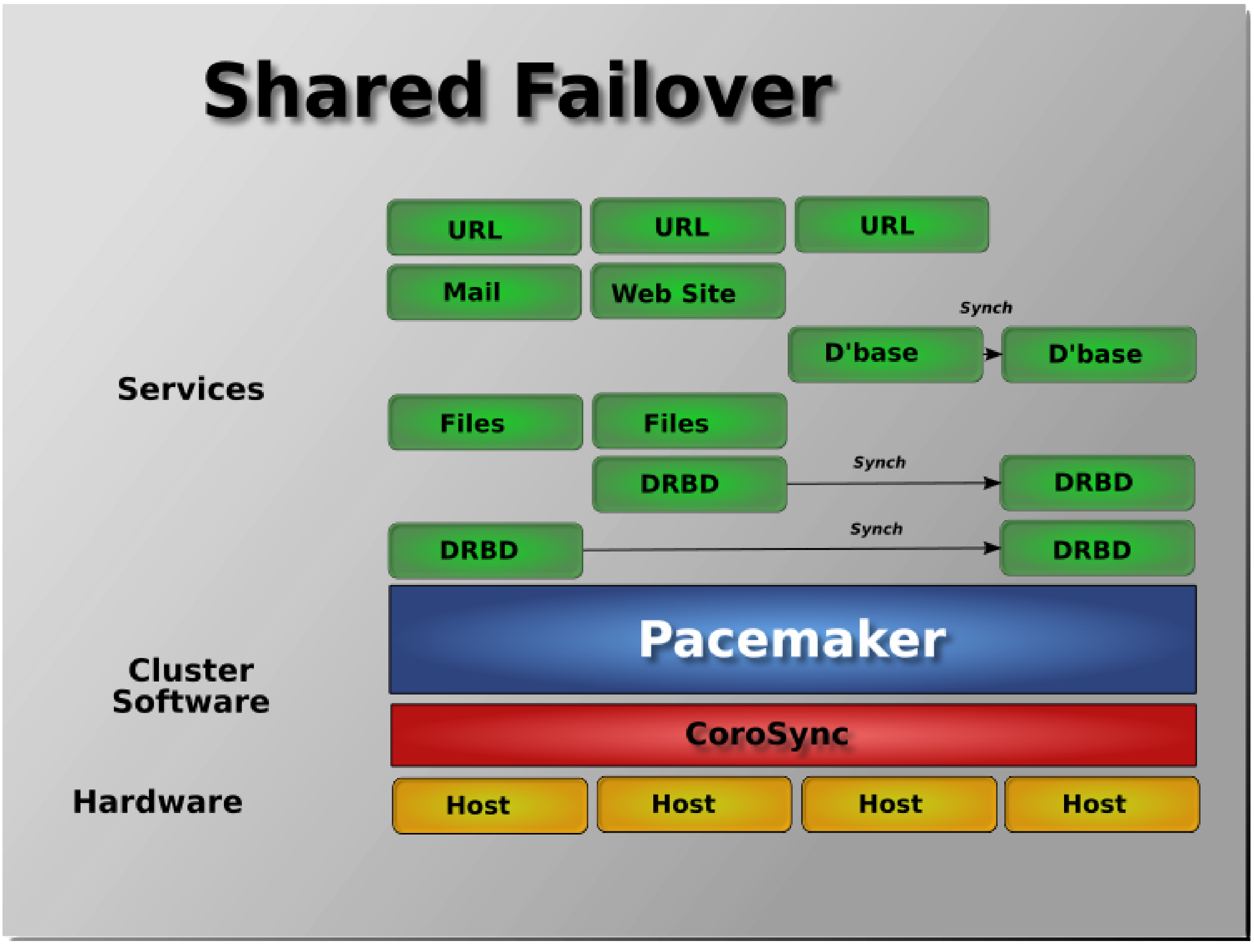
### Node Redundancy Designs

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

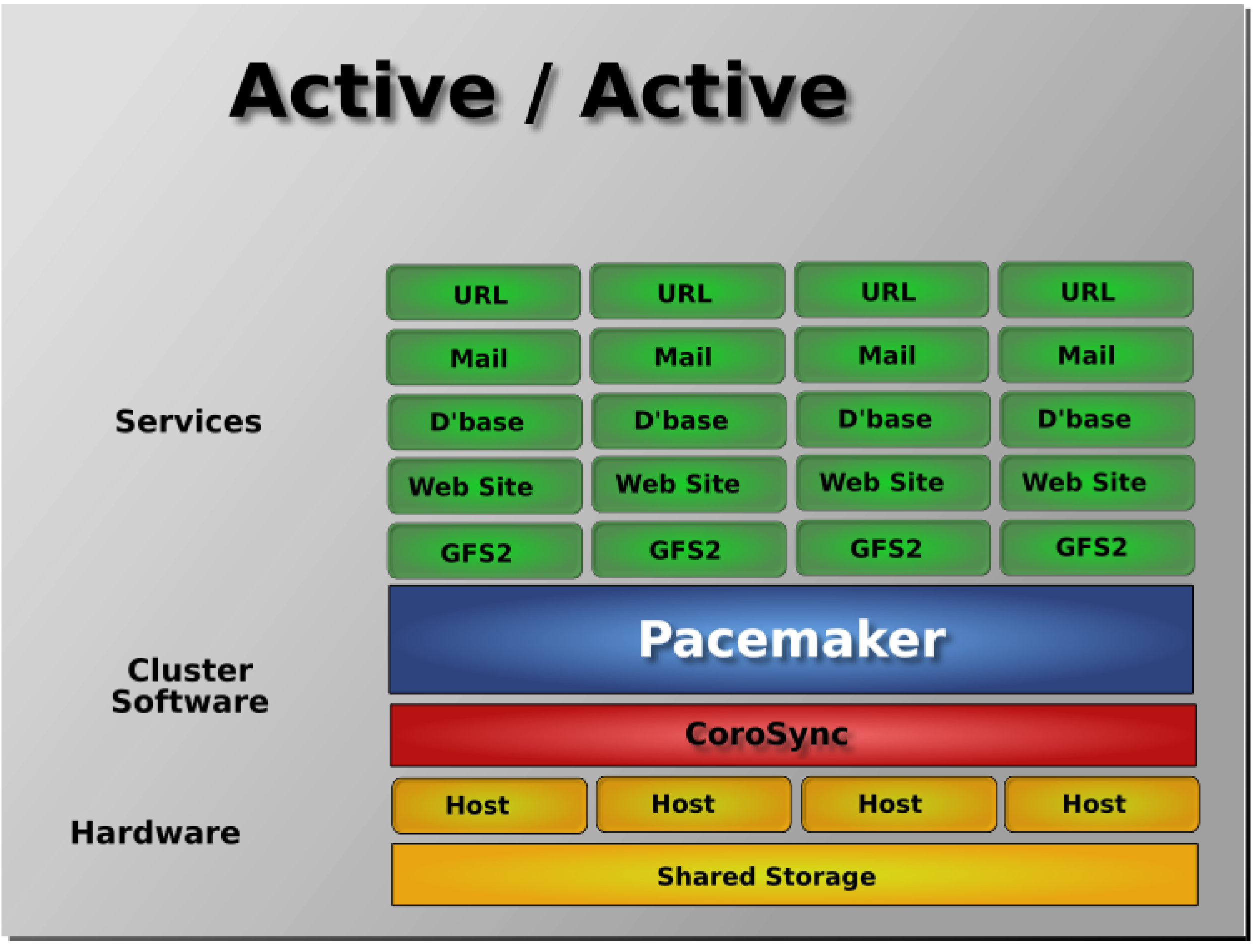
Active/passive clusters with two (or more) nodes using Pacemaker and DRBD are a cost-effective high-availability solution for many situations. One of the nodes provides the desired services, and if it fails, the other node takes over.




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




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




## 2.2 Host-Local Configuration

**Note:** Directory and file paths below may differ on your system depending on your Pacemaker build settings. Check your Pacemaker configuration file to find the correct paths.

### 2.2.1 Configuration Value Types

Throughout this document, configuration values will be designated as having one of the following types:

Table 1: **Configuration Value Types**

| Type | Description |
|---|---|
| boolean | Case-insensitive text value where `1`, `yes`, `y`, `on`, and `true` evaluate as true and `0`, `no`, `n`, `off`, `false`, and unset evaluate as false |
| date/time | Textual timestamp like `Sat Dec 21 11:47:45 2013` |
| duration | A nonnegative time duration, specified either like a *timeout* or an ISO 8601 duration. A duration may be up to approximately 49 days but is intended for much smaller time periods. |

Continued on next page

Table 1 – continued from previous page

| Type | Description |
|---|---|
| enumeration | Text that must be one of a set of defined values (which will be listed in the description) |
| epoch_time | Time as the integer number of seconds since the Unix epoch, `1970-01-01 00:00:00 +0000 (UTC)`. |
| id | A text string starting with a letter or underbar, followed by any combination of letters, numbers, dashes, dots, and/or underbars; when used for a property named `id`, the string must be unique across all `id` properties in the CIB |
| integer | 32-bit signed integer value (-2,147,483,648 to 2,147,483,647) |
| ISO 8601 | An ISO 8601 date/time. |
| nonnegative integer | 32-bit nonnegative integer value (0 to 2,147,483,647) |
| percentage | Floating-point number followed by an optional percent sign ('%') |
| port | Integer TCP port number (0 to 65535) |
| range | A range may be a single nonnegative integer or a dash-separated range of nonnegative integers. Either the first or last value may be omitted to leave the range open-ended. Examples: `0`, `3-`, `-5`, `4-6`. |
| score | A Pacemaker score can be an integer between -1,000,000 and 1,000,000, or a string alias: `INFINITY` or `+INFINITY` is equivalent to 1,000,000, `-INFINITY` is equivalent to -1,000,000, and `red`, `yellow`, and `green` are equivalent to integers as described in *Tracking Node Health*. |
| text | A text string |
| timeout | A time duration, specified as a bare number (in which case it is considered to be in seconds) or a number with a unit (`ms` or `msec` for milliseconds, `us` or `usec` for microseconds, `s` or `sec` for seconds, `m` or `min` for minutes, `h` or `hr` for hours) optionally with whitespace before and/or after the number. |
| version | Version number (any combination of alphanumeric characters, dots, and dashes, starting with a number). |

### Scores

Scores are integral to how Pacemaker 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.

Score addition and subtraction follow these rules:

- Any value (including `INFINITY`) - `INFINITY` = `-INFINITY`
- `INFINITY` + any value other than `-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).

### 2.2.2 Local Options

Pacemaker supports several host-local configuration options. These options can be configured on each node in the main Pacemaker configuration file (`/etc/sysconfig/pacemaker`) in the format `<NAME>="<VALUE>"`. They work by setting environment variables when Pacemaker daemons start up.

Table 2: **Local Options**

| Name | Type | Default | Description |
|---|---|---|---|
| CIB_pam_service | *text* | login | PAM service to use for remote CIB client authentication (passed to `pam_start`). |
| PCMK_logfacility | *enumeration* | daemon | Enable logging via the system log or journal, using the specified log facility. Messages sent here are of value to all Pacemaker administrators. This can be disabled using `none`, but that is not recommended. Allowed values:<br>• `none`<br>• `daemon`<br>• `user`<br>• `local0`<br>• `local1`<br>• `local2`<br>• `local3`<br>• `local4`<br>• `local5`<br>• `local6`<br>• `local7` |
| PCMK_logpriority | *enumeration* | notice | Unless system logging is disabled using `PCMK_logfacility=none`, messages of the specified log severity and higher will be sent to the system log. The default is appropriate for most installations. Allowed values:<br>• `emerg`<br>• `alert`<br>• `crit`<br>• `error`<br>• `warning`<br>• `notice`<br>• `info`<br>• `debug` |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| PCMK_logfile | *text* | /var/log/pacemaker/pacemaker.log | Unless set to `none`, more detailed log messages will be sent to the specified file (in addition to the system log, if enabled). These messages may have extended information, and will include messages of info severity. This log is of more use to developers and advanced system administrators, and when reporting problems. Note: The default is `/var/log/pcmk-init.log` (inside the container) for bundled container nodes; this would typically be mapped to a different path on the host running the container. |
| PCMK_logfile_mode | *text* | 0660 | Pacemaker will set the permissions on the detail log to this value (see `chmod(1)`). |
| PCMK_debug | *enumeration* | no | Whether to send debug severity messages to the detail log. This may be set for all subsystems (`yes` or `no`) or for specific (comma- separated) subsystems. Allowed subsystems are:<br>• `pacemakerd`<br>• `pacemaker-attrd`<br>• `pacemaker-based`<br>• `pacemaker-controld`<br>• `pacemaker-execd`<br>• `pacemaker-fenced`<br>• `pacemaker-schedulerd`<br>Example: `PCMK_debug="pacemakerd, pacemaker-execd"` |
| PCMK_stderr | *boolean* | no | *Advanced Use Only:* Whether to send daemon log messages to stderr. This would be useful only during troubleshooting, when starting Pacemaker manually on the command line.<br>Setting this option in the configuration file is pointless, since the file is not read when starting Pacemaker manually. However, it can be set directly as an environment variable on the command line. |
| PCMK_trace_functions | *text* | | *Advanced Use Only:* Send debug and trace severity messages from these (comma-separated) source code functions to the detail log.<br>Example: `PCMK_trace_functions="func1, func2"` |
| PCMK_trace_files | *text* | | *Advanced Use Only:* Send debug and trace severity messages from all functions in these (comma-separated) source file names to the detail log.<br>Example: `PCMK_trace_files="file1.c, file2.c"` |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| PCMK_trace_formats | *text* | | *Advanced Use Only:* Send trace severity messages that are generated by these (comma-separated) format strings in the source code to the detail log.<br>Example: `PCMK_trace_formats="Error: %s (%d)"` |
| PCMK_trace_tags | *text* | | *Advanced Use Only:* Send debug and trace severity messages related to these (comma-separated) resource IDs to the detail log.<br>Example: `PCMK_trace_tags="client-ip, dbfs"` |
| PCMK_blackbox | *enumeration* | no | *Advanced Use Only:* Enable blackbox logging globally (`yes` or `no`) or by subsystem. A blackbox contains a rolling buffer of all logs (of all severities). Blackboxes are stored under `/var/lib/pacemaker/blackbox` by default, by default, and their contents can be viewed using the `qb-blackbox(8)` command.<br>The blackbox recorder can be enabled at start using this variable, or at runtime by sending a Pacemaker subsystem daemon process a `SIGUSR1` or `SIGTRAP` signal, and disabled by sending `SIGUSR2` (see `kill(1)`). The blackbox will be written after a crash, assertion failure, or `SIGTRAP` signal.<br>See *PCMK_debug* for allowed subsystems.<br>Example: `PCMK_blackbox="pacemakerd, pacemaker-execd"` |
| PCMK_trace_blackbox | *enumeration* | | *Advanced Use Only:* Write a blackbox whenever the message at the specified function and line is logged. Multiple entries may be comma-separated.<br>Example: `PCMK_trace_blackbox="remote.c:144,remote.c:149"` |
| PCMK_node_start_state | *enumeration* | default | By default, the local host will join the cluster in an online or standby state when Pacemaker first starts depending on whether it was previously put into standby mode. If this variable is set to `standby` or `online`, it will force the local host to join in the specified state. |
| PCMK_node_action_limit | *nonnegative integer* | | If set, this overrides the *node-action-limit* cluster option on this node to specify the maximum number of jobs that can be scheduled on this node (or 0 to use twice the number of CPU cores). |
| PCMK_shutdown_delay | *timeout* | | Specify a delay before shutting down `pacemakerd` after shutting down all other Pacemaker daemons. |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| PCMK_fail_fast | *boolean* | no | By default, if a Pacemaker subsystem crashes, the main `pacemakerd` process will attempt to restart it. If this variable is set to `yes`, `pacemakerd` will panic the local host instead. |
| PCMK_panic_action | *enumeration* | reboot | Pacemaker will panic the local host under certain conditions. By default, this means rebooting the host. This variable can change that behavior: if `crash`, trigger a kernel crash (useful if you want a kernel dump to investigate); if `sync-reboot` or `sync-crash`, synchronize filesystems before rebooting the host or triggering a kernel crash. The sync values are more likely to preserve log messages, but with the risk that the host may be left active if the synchronization hangs. |
| PCMK_authkey_location | *text* | /etc/pacemaker/authkey | Use the contents of this file as the authorization key to use with Pacemaker Remote connections. This file must be readable by Pacemaker daemons (that is, it must allow read permissions to either the `hacluster` user or the `haclient` group), and its contents must be identical on all nodes. |
| PCMK_remote_address | *text* | | By default, if the Pacemaker Remote service is run on the local node, it will listen for connections on all IP addresses. This may be set to one address to listen on instead, as a resolvable hostname or as a numeric IPv4 or IPv6 address. When resolving names or listening on all addresses, IPv6 will be preferred if available. When listening on an IPv6 address, IPv4 clients will be supported via IPv4-mapped IPv6 addresses.<br>Example: `PCMK_remote_address="192.0.2.1"` |
| PCMK_remote_port | *port* | 3121 | Use this TCP port number for Pacemaker Remote node connections. This value must be the same on all nodes. |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| PCMK_remote_pid1 | *enumeration* | default | *Advanced Use Only:* When a bundle resource's `run-command` option is left to default, Pacemaker Remote runs as PID 1 in the bundle's containers. When it does so, it loads environment variables from the container's `/etc/pacemaker/pcmk-init.env` and performs the PID 1 responsibility of reaping dead subprocesses.<br>This option controls whether those actions are performed when Pacemaker Remote is not running as PID 1. It is intended primarily for developer testing but can be useful when `run-command` is set to a separate, custom PID 1 process that launches Pacemaker Remote.<br>• `full`: Pacemaker Remote loads environment variables from `/etc/pacemaker/pcmk-init.env` and reaps dead subprocesses.<br>• `vars`: Pacemaker Remote loads environment variables from `/etc/pacemaker/pcmk-init.env` but does not reap dead subprocesses.<br>• `default`: Pacemaker Remote performs neither action.<br>If Pacemaker Remote is running as PID 1, this option is ignored, and the behavior is the same as for `full`. |
| PCMK_tls_priorities | *text* | NORMAL | *Advanced Use Only:* These GnuTLS cipher priorities will be used for TLS connections (whether for Pacemaker Remote connections or remote CIB access, when enabled). See:<br>https://gnutls.org/manual/html_node/Priority-Strings.html<br>Pacemaker will append `":+ANON-DH"` for remote CIB access and `":+DHE-PSK:+PSK"` for Pacemaker Remote connections, as they are required for the respective functionality.<br>Example: `PCMK_tls_priorities="SECURE128:+SECURE192"` |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| PCMK_dh_min_bits | *nonnegative integer* | 0 (no minimum) | *Advanced Use Only:* Set a lower bound on the bit length of the prime number generated for Diffie-Hellman parameters needed by TLS connections. The default is no minimum.<br>The server (Pacemaker Remote daemon, or CIB manager configured to accept remote clients) will use this value to provide a floor for the value recommended by the GnuTLS library. The library will only accept a limited number of specific values, which vary by library version, so setting these is recommended only when required for compatibility with specific client versions.<br>Clients (connecting cluster nodes or remote CIB commands) will require that the server use a prime of at least this size. This is recommended only when the value must be lowered in order for the client's GnuTLS library to accept a connection to an older server. |
| PCMK_dh_max_bits | *nonnegative integer* | 0 (no maximum) | *Advanced Use Only:* Set an upper bound on the bit length of the prime number generated for Diffie-Hellman parameters needed by TLS connections. The default is no maximum.<br>The server (Pacemaker Remote daemon, or CIB manager configured to accept remote clients) will use this value to provide a ceiling for the value recommended by the GnuTLS library. The library will only accept a limited number of specific values, which vary by library version, so setting these is recommended only when required for compatibility with specific client versions.<br>Clients do not use `PCMK_dh_max_bits`. |
| PCMK_ipc_type | *enumeration* | shared-mem | *Advanced Use Only:* Force use of a particular IPC method. Allowed values:<br>• `shared-mem`<br>• `socket`<br>• `posix`<br>• `sysv` |
| PCMK_ipc_buffer | *nonnegative integer* | 131072 | *Advanced Use Only:* Specify an IPC buffer size in bytes. This can be useful when connecting to large clusters that result in messages exceeding the default size (which will also result in log messages referencing this variable). |

Continued on next page

Table 2 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| PCMK_cluster_type | *enumeration* | corosync | *Advanced Use Only:* Specify the cluster layer to be used. If unset, Pacemaker will detect and use a supported cluster layer, if available. Currently, `"corosync"` is the only supported cluster layer. If multiple layers are supported in the future, this will allow overriding Pacemaker's automatic detection to select a specific one. |
| PCMK_schema_directory | *text* | /usr/share/pacemaker | *Advanced Use Only:* Specify an alternate location for RNG schemas and XSL transforms. |
| PCMK_remote_schema_directory | *text* | /var/lib/pacemaker/schemas | *Advanced Use Only:* Specify an alternate location on Pacemaker Remote nodes for storing newer RNG schemas and XSL transforms fetched from the cluster. |
| PCMK_valgrind_enabled | *enumeration* | no | *Advanced Use Only:* Whether subsystem daemons should be run under `valgrind`. Allowed values are the same as for `PCMK_debug`. |
| PCMK_callgrind_enabled | *enumeration* | no | *Advanced Use Only:* Whether subsystem daemons should be run under `valgrind` with the `callgrind` tool enabled. Allowed values are the same as for `PCMK_debug`. |
| SBD_SYNC_RESOURCE_STARTUP | *boolean* | | If true, `pacemakerd` waits for a ping from `sbd` during startup before starting other Pacemaker daemons, and during shutdown after stopping other Pacemaker daemons but before exiting. Default value is set based on the `--with-sbd-sync-default` configure script option. |
| SBD_WATCHDOG_TIMEOUT | *duration* | | If the `stonith-watchdog-timeout` cluster property is set to a negative or invalid value, use double this value as the default if positive, or use 0 as the default otherwise. This value must be greater than the value of `stonith-watchdog-timeout` if both are set. |
| VALGRIND_OPTS | *text* | | *Advanced Use Only:* Pass these options to valgrind, when enabled (see `valgrind(1)`). `"--vgdb=no"` should usually be specified because `pacemaker-execd` can lower privileges when executing commands, which would otherwise leave a bunch of unremovable files in `/tmp`. |

## 2.3 Cluster-Wide Configuration

### 2.3.1 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:

**An empty configuration**

```
<cib crm_feature_set="3.6.0" validate-with="pacemaker-3.5" epoch="1" num_updates="0" admin_epoch=
→"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` element. Certain fundamental settings are defined as attributes of this element.
  - `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 executor (pacemaker-execd 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.

### 2.3.2 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: **CIB Properties**

| Name | Type | Default | Description |
|---|---|---|---|
| admin_epoch | *nonnegative integer* | 0 | When a node joins the cluster, the cluster asks the node with the highest (`admin_epoch`, `epoch`, `num_updates`) tuple to replace the configuration on all the nodes – which makes 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. |
| epoch | *nonnegative integer* | 0 | The cluster increments this every time the CIB's configuration section is updated. |
| num_updates | *nonnegative integer* | 0 | The cluster increments this every time the CIB's configuration or status sections are updated, and resets it to 0 when epoch changes. |
| validate-with | *enumeration* | | Determines the type of XML validation that will be done on the configuration. Allowed values are `none` (in which case the cluster will not require that updates conform to expected syntax) and the base names of schema files installed on the local machine (for example, "pacemaker-3.9") |
| remote-tls-port | *port* | | If set, the CIB manager will listen for anonymously encrypted remote connections on this port, to allow CIB administration from hosts not in the cluster. No key is used, so this should be used only on a protected network where man-in-the-middle attacks can be avoided. |
| remote-clear-port | *port* | | If set to a TCP port number, the CIB manager will listen for remote connections on this port, to allow for CIB administration from hosts not in the cluster. No encryption is used, so this should be used only on a protected network. |
| cib-last-written | *date/time* | | Indicates when the configuration was last written to disk. Maintained by the cluster; for informational purposes only. |
| have-quorum | *boolean* | | Indicates whether the cluster has quorum. If false, the cluster's response is determined by `no-quorum-policy` (see below). Maintained by the cluster. |
| dc-uuid | *text* | | Node ID of the cluster's current designated controller (DC). Used and maintained by the cluster. |
| execution-date | *epoch time* | | Time to use when evaluating rules. |

### 2.3.3 Cluster Options

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

They are grouped into sets within the `crm_config` section. In advanced configurations, there may be more than one set. (This will be described later in the chapter on *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 pacemaker-schedulerd` and `man pacemaker-controld` commands.

Table 4: **Cluster Options**

| Name | Type | Default | Description |
|---|---|---|---|
| cluster-name | *text* | | An (optional) name for the cluster as a whole. This is mostly for users' convenience for use as desired in administration, but can be used in the Pacemaker configuration in *Rules* (as the `#cluster-name` *node attribute*). It may also be used by higher-level tools when displaying cluster information, and by certain resource agents (for example, the `ocf:heartbeat:GFS2` agent stores the cluster name in filesystem meta-data). |
| dc-version | *version* | *detected* | Version of Pacemaker on the cluster's designated controller (DC). Maintained by the cluster, and intended for diagnostic purposes. |
| cluster-infrastructure | *text* | *detected* | The messaging layer with which Pacemaker is currently running. Maintained by the cluster, and intended for informational and diagnostic purposes. |
| no-quorum-policy | *enumeration* | stop | What to do when the cluster does not have quorum. Allowed values:<br>• `ignore`: continue all resource management<br>• `freeze`: continue resource management, but don't recover resources from nodes not in the affected partition<br>• `stop`: stop all resources in the affected cluster partition<br>• `demote`: demote promotable resources and stop all other resources in the affected cluster partition *(since 2.0.5)*<br>• `fence`: fence all nodes in the affected cluster partition *(since 2.1.9)*<br>• `suicide`: same as `fence` *(deprecated since 2.1.9)* |
| batch-limit | *integer* | 0 | The maximum number of actions that the cluster may execute in parallel across all nodes. The ideal value will depend on the speed and load of your network and cluster nodes. If zero, the cluster will impose a dynamically calculated limit only when any node has high load. If -1, the cluster will not impose any limit. |
| migration-limit | *integer* | -1 | The number of *live migration* actions that the cluster is allowed to execute in parallel on a node. A value of -1 means unlimited. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| load-threshold | *percentage* | 80% | Maximum amount of system load that should be used by cluster nodes. The cluster will slow down its recovery process when the amount of system resources used (currently CPU) approaches this limit. |
| node-action-limit | *integer* | 0 | Maximum number of jobs that can be scheduled per node. If nonpositive or invalid, double the number of cores is used as the maximum number of jobs per node. *PCMK_node_action_limit* overrides this option on a per-node basis. |
| symmetric-cluster | *boolean* | true | If true, resources can run on any node by default. If false, a resource is allowed to run on a node only if a *location constraint* enables it. |
| stop-all-resources | *boolean* | false | Whether all resources should be disallowed from running (can be useful during maintenance or troubleshooting) |
| stop-orphan-resources | *boolean* | true | Whether resources that have been deleted from the configuration should be stopped. This value takes precedence over *is-managed* (that is, even unmanaged resources will be stopped when orphaned if this value is `true`). |
| stop-orphan-actions | *boolean* | true | Whether recurring *operations* that have been deleted from the configuration should be cancelled |
| start-failure-is-fatal | *boolean* | true | Whether a failure to start a resource on a particular node prevents further start attempts on that node. If `false`, the cluster will decide whether the node is still eligible based on the resource's current failure count and `migration-threshold`. |
| enable-startup-probes | *boolean* | true | Whether the cluster should check the pre-existing state of resources when the cluster starts |
| maintenance-mode | *boolean* | false | If true, the cluster will not start or stop any resource in the cluster, and any recurring operations (expect those specifying `role` as `Stopped`) will be paused. If true, this overrides the *maintenance* node attribute, *is-managed* and *maintenance* resource meta-attributes, and *enabled* operation meta-attribute. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| stonith-enabled | *boolean* | true | Whether the cluster is allowed to fence nodes (for example, failed nodes and nodes with resources that can't be stopped).<br>If true, at least one fence device must be configured before resources are allowed to run.<br>If false, unresponsive nodes are immediately assumed to be running no resources, and resource recovery on online nodes starts without any further protection (which can mean *data loss* if the unresponsive node still accesses shared storage, for example). See also the *requires* resource meta-attribute. |
| stonith-action | *enumeration* | reboot | Action the cluster should send to the fence agent when a node must be fenced. Allowed values are `reboot`, `off`, and (for legacy agents only) `poweroff`. |
| stonith-timeout | *duration* | 60s | How long to wait for `on`, `off`, and `reboot` fence actions to complete by default. |
| stonith-max-attempts | *score* | 10 | How many times fencing can fail for a target before the cluster will no longer immediately re-attempt it. Any value below 1 will be ignored, and the default will be used instead. |
| have-watchdog | *boolean* | *detected* | Whether watchdog integration is enabled. This is set automatically by the cluster according to whether SBD is detected to be in use. User-configured values are ignored. The value *true* is meaningful if diskless SBD is used and *stonith-watchdog-timeout* is nonzero. In that case, if fencing is required, watchdog-based self-fencing will be performed via SBD without requiring a fencing resource explicitly configured. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| stonith-watchdog-timeout | *timeout* | 0 | If nonzero, and the cluster detects `have-watchdog` as `true`, then watchdog-based self-fencing will be performed via SBD when fencing is required.<br>If this is set to a positive value, lost nodes are assumed to achieve self-fencing within this much time.<br>This does not require a fencing resource to be explicitly configured, though a fence_watchdog resource can be configured, to limit use to specific nodes.<br>If this is set to 0 (the default), the cluster will never assume watchdog-based self-fencing.<br>If this is set to a negative value, the cluster will use twice the local value of the `SBD_WATCHDOG_TIMEOUT` environment variable if that is positive, or otherwise treat this as 0.<br>**Warning:** When used, this timeout must be larger than `SBD_WATCHDOG_TIMEOUT` on all nodes that use watchdog-based SBD, and Pacemaker will refuse to start on any of those nodes where this is not true for the local value or SBD is not active. When this is set to a negative value, `SBD_WATCHDOG_TIMEOUT` must be set to the same value on all nodes that use SBD, otherwise data corruption or loss could occur. |
| concurrent-fencing | *boolean* | false | Whether the cluster is allowed to initiate multiple fence actions concurrently. Fence actions initiated externally, such as via the `stonith_admin` tool or an application such as DLM, or by the fencer itself such as recurring device monitors and `status` and `list` commands, are not limited by this option. |
| fence-reaction | *enumeration* | stop | How should a cluster node react if notified of its own fencing? A cluster node may receive notification of a "succeeded" fencing that targeted it if fencing is misconfigured, or if fabric fencing is in use that doesn't cut cluster communication. Allowed values are `stop` to attempt to immediately stop Pacemaker and stay stopped, or `panic` to attempt to immediately reboot the local node, falling back to stop on failure. The default is likely to be changed to `panic` in a future release. *(since 2.0.3)* |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| priority-fencing-delay | *duration* | 0 | Apply this delay to any fencing targeting the lost nodes with the highest total resource priority in case we don't have the majority of the nodes in our cluster partition, so that the more significant nodes potentially win any fencing match (especially meaningful in a split-brain of a 2-node cluster). A promoted resource instance takes the resource's priority plus 1 if the resource's priority is not 0. Any static or random delays introduced by `pcmk_delay_base` and `pcmk_delay_max` configured for the corresponding fencing resources will be added to this delay. This delay should be significantly greater than (safely twice) the maximum delay from those parameters. *(since 2.0.4)* |
| node-pending-timeout | *duration* | 0 | Fence nodes that do not join the controller process group within this much time after joining the cluster, to allow the cluster to continue managing resources. A value of 0 means never fence pending nodes. Setting the value to 2h means fence nodes after 2 hours. *(since 2.1.7)* |
| cluster-delay | *duration* | 60s | If the DC 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 within this time (beyond the action's own timeout). The ideal value will depend on the speed and load of your network and cluster nodes. |
| dc-deadtime | *duration* | 20s | How long to wait for a response from other nodes when electing a DC. The ideal value will depend on the speed and load of your network and cluster nodes. |
| cluster-ipc-limit | *nonnegative integer* | 500 | The maximum IPC message backlog before one cluster daemon will disconnect another. This is of use in large clusters, for which a good value is the number of resources in the cluster multiplied by the number of nodes. The default of 500 is also the minimum. Raise this if you see "Evicting client" log messages for cluster daemon process IDs. |
| pe-error-series-max | *integer* | -1 | The number of scheduler inputs resulting in errors to save. These inputs can be helpful during troubleshooting and when reporting issues. A negative value means save all inputs, and 0 means save none. |
| pe-warn-series-max | *integer* | 5000 | The number of scheduler inputs resulting in warnings to save. These inputs can be helpful during troubleshooting and when reporting issues. A negative value means save all inputs, and 0 means save none. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| pe-input-series-max | *integer* | 4000 | The number of "normal" scheduler inputs to save. These inputs can be helpful during troubleshooting and when reporting issues. A negative value means save all inputs, and 0 means save none. |
| enable-acl | *boolean* | false | Whether *access control lists* should be used to authorize CIB modifications |
| placement-strategy | *enumeration* | default | How the cluster should assign resources to nodes (see *Utilization and Placement Strategy*). Allowed values are `default`, `utilization`, `balanced`, and `minimal`. |
| node-health-strategy | *enumeration* | none | How the cluster should react to *node health* attributes. Allowed values are `none`, `migrate-on-red`, `only-green`, `progressive`, and `custom`. |
| node-health-base | *score* | 0 | The base health score assigned to a node. Only used when `node-health-strategy` is `progressive`. |
| node-health-green | *score* | 0 | The score to use for a node health attribute whose value is `green`. Only used when `node-health-strategy` is `progressive` or `custom`. |
| node-health-yellow | *score* | 0 | The score to use for a node health attribute whose value is `yellow`. Only used when `node-health-strategy` is `progressive` or `custom`. |
| node-health-red | *score* | -INFINITY | The score to use for a node health attribute whose value is `red`. Only used when `node-health-strategy` is `progressive` or `custom`. |
| cluster-recheck-interval | *duration* | 15min | Pacemaker is primarily event-driven, and looks ahead to know when to recheck the cluster for failure-timeout settings and most time-based rules *(since 2.0.3)*. However, it will also recheck the cluster after this amount of inactivity. This has two goals: rules with `date_spec` are only guaranteed to be checked this often, and it also serves as a fail-safe for some kinds of scheduler bugs. A value of 0 disables this polling. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| shutdown-lock | *boolean* | false | The default of false allows active resources to be recovered elsewhere when their node is cleanly shut down, which is what the vast majority of users will want. However, some users prefer to make resources highly available only for failures, with no recovery for clean shutdowns. If this option is true, resources active on a node when it is cleanly shut down are kept "locked" to that node (not allowed to run elsewhere) until they start again on that node after it rejoins (or for at most `shutdown-lock-limit`, if set). Stonith resources and Pacemaker Remote connections are never locked. Clone and bundle instances and the promoted role of promotable clones are currently never locked, though support could be added in a future release. Locks may be manually cleared using the `--refresh` option of `crm_resource` (both the resource and node must be specified; this works with remote nodes if their connection resource's `target-role` is set to `Stopped`, but not if Pacemaker Remote is stopped on the remote node without disabling the connection resource). *(since 2.0.4)* |
| shutdown-lock-limit | *duration* | 0 | If `shutdown-lock` is true, and this is set to a nonzero time duration, locked resources will be allowed to start after this much time has passed since the node shutdown was initiated, even if the node has not rejoined. (This works with remote nodes only if their connection resource's `target-role` is set to `Stopped`.) *(since 2.0.4)* |
| remove-after-stop | *boolean* | false | *Deprecated* Whether the cluster should remove resources from Pacemaker's executor after they are stopped. Values other than the default are, at best, poorly tested and potentially dangerous. This option is deprecated and will be removed in a future release. |
| startup-fencing | *boolean* | true | *Advanced Use Only:* Whether the cluster should fence unseen nodes at start-up. Setting this to false is unsafe, because the unseen nodes could be active and running resources but unreachable. `dc-deadtime` acts as a grace period before this fencing, since a DC must be elected to schedule fencing. |
| election-timeout | *duration* | 2min | *Advanced Use Only:* If a winner is not declared within this much time of starting an election, the node that initiated the election will declare itself the winner. |

Continued on next page

Table 4 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| shutdown-escalation | *duration* | 20min | *Advanced Use Only:* The controller will exit immediately if a shutdown does not complete within this much time. |
| join-integration-timeout | *duration* | 3min | *Advanced Use Only:* If you need to adjust this value, it probably indicates the presence of a bug. |
| join-finalization-timeout | *duration* | 30min | *Advanced Use Only:* If you need to adjust this value, it probably indicates the presence of a bug. |
| transition-delay | *duration* | 0s | *Advanced Use Only:* Delay cluster recovery for the configured interval to allow for additional or related events to occur. This can be useful if your configuration is sensitive to the order in which ping updates arrive. Enabling this option will slow down cluster recovery under all conditions. |

## 2.4 Cluster Nodes

### 2.4.1 Defining a Cluster Node

Each cluster node will have an entry in the `nodes` section containing at least an ID and a name. A cluster node's ID is defined by the cluster layer (Corosync).

**Example Corosync cluster node entry**

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

In normal circumstances, the admin should let the cluster populate this information automatically from the cluster layer.

#### Where Pacemaker Gets the Node Name

The name that Pacemaker uses for a node in the configuration does not have to be the same as its local hostname. Pacemaker uses the following for a Corosync node's name, in order of most preferred first:

- The value of `name` in the `nodelist` section of `corosync.conf`
- The value of `ring0_addr` in the `nodelist` section of `corosync.conf`
- The local hostname (value of `uname -n`)

If the cluster is running, the `crm_node -n` command will display the local node's name as used by the cluster.

If a Corosync `nodelist` is used, `crm_node --name-for-id` with a Corosync node ID will display the name used by the node with the given Corosync `nodeid`, for example:

```
crm_node --name-for-id 2
```

### 2.4.2 Node Attributes

Pacemaker allows node-specific values to be specified using *node attributes*. A node attribute has a name, and may have a distinct value for each node.

Node attributes come in two types, *permanent* and *transient*. Permanent node attributes are kept within the `node` entry, and keep their values even if the cluster restarts on a node. Transient node attributes are kept in the CIB's `status` section, and go away when the cluster stops on the node.

While certain node attributes have specific meanings to the cluster, they are mainly intended to allow administrators and resource agents to track any information desired.

For example, an administrator might choose to define node attributes for how much RAM and disk space each node has, which OS each uses, or which server room rack each node is in.

Users can configure *Rules* that use node attributes to affect where resources are placed.

#### Setting and querying node attributes

Node attributes can be set and queried using the `crm_attribute` and `attrd_updater` commands, so that the user does not have to deal with XML configuration directly.

Here is an example command to set a permanent node attribute, and the XML configuration that would be generated:

**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 id="1" uname="pcmk-1">
   <instance_attributes id="nodes-1-attributes">
     <nvpair id="nodes-1-kernel" name="kernel" value="3.10.0-862.14.4.el7.x86_64"/>
   </instance_attributes>
</node>
```

To read back the value that was just set:

```
# crm_attribute --type nodes --node pcmk-1 --name kernel --query
scope=nodes  name=kernel value=3.10.0-862.14.4.el7.x86_64
```

The `--type nodes` indicates that this is a permanent node attribute; `--type status` would indicate a transient node attribute.

**Warning:** Attribute values with newline or tab characters are currently displayed with newlines as `"\n"` and tabs as `"\t"`, when `crm_attribute` or `attrd_updater` query commands use `--output-as=text` or leave `--output-as` unspecified:

```
# crm_attribute -N node1 -n test_attr -v "$(echo -e "a\nb\tc")" -t status
# crm_attribute -N node1 -n test_attr --query -t status
scope=status  name=test_attr value=a\nb\tc
```

This format is deprecated. In a future release, the values will be displayed with literal whitespace characters:

```
# crm_attribute -N node1 -n test_attr --query -t status
scope=status  name=test_attr value=a
b c
```

Users should either avoid attribute values with newlines and tabs, or ensure that they can handle both formats.

However, it's best to use `--output-as=xml` when parsing attribute values from output. Newlines, tabs, and special characters are replaced with XML character references that a conforming XML processor can recognize and convert to literals *(since 2.1.8)*:

```
# crm_attribute -N node1 -n test_attr --query -t status --output-as=xml
<pacemaker-result api-version="2.35" request="crm_attribute -N laptop -n test_attr --query -t status --output-as=xml">
  <attribute name="test_attr" value="a&#10;b&#9;c" scope="status"/>
  <status code="0" message="OK"/>
</pacemaker-result>
```

### Special node attributes

Certain node attributes have special meaning to the cluster.

Node attribute names beginning with `#` are considered reserved for these special attributes. Some special attributes do not start with `#`, for historical reasons.

Certain special attributes are set automatically by the cluster, should never be modified directly, and can be used only within *Rules*; these are listed under *built-in node attributes*.

For true/false values, the cluster considers a value of "1", "y", "yes", "on", or "true" (case-insensitively) to be true, "0", "n", "no", "off", "false", or unset to be false, and anything else to be an error.

Table 5: **Node attributes with special significance**

| Name | Description |
|---|---|
| fail-count-* | Attributes whose names start with `fail-count-` are managed by the cluster to track how many times particular resource operations have failed on this node. These should be queried and cleared via the `crm_failcount` or `crm_resource --cleanup` commands rather than directly. |
| last-failure-* | Attributes whose names start with `last-failure-` are managed by the cluster to track when particular resource operations have most recently failed on this node. These should be cleared via the `crm_failcount` or `crm_resource --cleanup` commands rather than directly. |
| maintenance | If true, the cluster will not start or stop any resources on this node. Any resources active on the node become unmanaged, and any recurring operations for those resources (except those specifying `role` as `Stopped`) will be paused. The *maintenance-mode* cluster option, if true, overrides this. If this attribute is true, it overrides the *is-managed* and *maintenance* meta-attributes of affected resources and *enabled* meta-attribute for affected recurring actions. Pacemaker should not be restarted on a node that is in single-node maintenance mode. |
| probe_complete | This is managed by the cluster to detect when nodes need to be reprobed, and should never be used directly. |

Continued on next page

Table 5 – continued from previous page

| Name | Description |
|---|---|
| resource-discovery-enabled | If the node is a remote node, fencing is enabled, and this attribute is explicitly set to false (unset means true in this case), resource discovery (probes) will not be done on this node. This is highly discouraged; the `resource-discovery` location constraint property is preferred for this purpose. |
| shutdown | This is managed by the cluster to orchestrate the shutdown of a node, and should never be used directly. |
| site-name | If set, this will be used as the value of the `#site-name` node attribute used in rules. (If not set, the value of the `cluster-name` cluster option will be used as `#site-name` instead.) |
| standby | If true, the node is in standby mode. This is typically set and queried via the `crm_standby` command rather than directly. |
| terminate | If the value is true or begins with any nonzero number, the node will be fenced. This is typically set by tools rather than directly. |
| #digests-* | Attributes whose names start with `#digests-` are managed by the cluster to detect when *Unfencing* needs to be redone, and should never be used directly. |
| #node-unfenced | When the node was last unfenced (as seconds since the epoch). This is managed by the cluster and should never be used directly. |

### 2.4.3 Tracking Node Health

A node may be functioning adequately as far as cluster membership is concerned, and yet be "unhealthy" in some respect that makes it an undesirable location for resources. For example, a disk drive may be reporting SMART errors, or the CPU may be highly loaded.

Pacemaker offers a way to automatically move resources off unhealthy nodes.

#### Node Health Attributes

Pacemaker will treat any node attribute whose name starts with `#health` as an indicator of node health. Node health attributes may have one of the following values:

Table 6: **Allowed Values for Node Health Attributes**

| Value | Intended significance |
|---|---|
| `red` | This indicator is unhealthy |
| `yellow` | This indicator is becoming unhealthy |
| `green` | This indicator is healthy |
| *integer* | A numeric score to apply to all resources on this node (0 or positive is healthy, negative is unhealthy) |

#### Node Health Strategy

Pacemaker assigns a node health score to each node, as the sum of the values of all its node health attributes. This score will be used as a location constraint applied to this node for all resources.

The `node-health-strategy` cluster option controls how Pacemaker responds to changes in node health attributes, and how it translates `red`, `yellow`, and `green` to scores.

Allowed values are:

Table 7: **Node Health Strategies**

| Value | Effect |
|---|---|
| none | Do not track node health attributes at all. |
| migrate-on-red | Assign the value of `-INFINITY` to `red`, and 0 to `yellow` and `green`. This will cause all resources to move off the node if any attribute is `red`. |
| only-green | Assign the value of `-INFINITY` to `red` and `yellow`, and 0 to `green`. This will cause all resources to move off the node if any attribute is `red` or `yellow`. |
| progressive | Assign the value of the `node-health-red` cluster option to `red`, the value of `node-health-yellow` to `yellow`, and the value of `node-health-green` to `green`. Each node is additionally assigned a score of `node-health-base` (this allows resources to start even if some attributes are `yellow`). This strategy gives the administrator finer control over how important each value is. |
| custom | Track node health attributes using the same values as `progressive` for `red`, `yellow`, and `green`, but do not take them into account. The administrator is expected to implement a policy by defining *Rules* referencing node health attributes. |

### Exempting a Resource from Health Restrictions

If you want a resource to be able to run on a node even if its health score would otherwise prevent it, set the resource's `allow-unhealthy-nodes` meta-attribute to `true` *(available since 2.1.3)*.

This is particularly useful for node health agents, to allow them to detect when the node becomes healthy again. If you configure a health agent without this setting, then the health agent will be banned from an unhealthy node, and you will have to investigate and clear the health attribute manually once it is healthy to allow resources on the node again.

If you want the meta-attribute to apply to a clone, it must be set on the clone itself, not on the resource being cloned.

### Configuring Node Health Agents

Since Pacemaker calculates node health based on node attributes, any method that sets node attributes may be used to measure node health. The most common are resource agents and custom daemons.

Pacemaker provides examples that can be used directly or as a basis for custom code. The `ocf:pacemaker:HealthCPU`, `ocf:pacemaker:HealthIOWait`, and `ocf:pacemaker:HealthSMART` resource agents set node health attributes based on CPU and disk status.

To take advantage of this feature, add the resource to your cluster (generally as a cloned resource with a recurring monitor action, to continually check the health of all nodes). For example:

**Example HealthIOWait resource configuration**

```
<clone id="resHealthIOWait-clone">
  <primitive class="ocf" id="HealthIOWait" provider="pacemaker" type="HealthIOWait">
    <instance_attributes id="resHealthIOWait-instance_attributes">
      <nvpair id="resHealthIOWait-instance_attributes-red_limit" name="red_limit" value="30"/>
      <nvpair id="resHealthIOWait-instance_attributes-yellow_limit" name="yellow_limit" value="10
↪"/>
    </instance_attributes>
    <operations>
      <op id="resHealthIOWait-monitor-interval-5" interval="5" name="monitor" timeout="5"/>
      <op id="resHealthIOWait-start-interval-0s" interval="0s" name="start" timeout="10s"/>
      <op id="resHealthIOWait-stop-interval-0s" interval="0s" name="stop" timeout="10s"/>
    </operations>
  </primitive>
</clone>
```

The resource agents use `attrd_updater` to set proper status for each node running this resource, as a node attribute whose name starts with `#health` (for `HealthIOWait`, the node attribute is named `#health-iowait`).

When a node is no longer faulty, you can force the cluster to make it available to take resources without waiting for the next monitor, by setting the node health attribute to green. For example:

**Force node1 to be marked as healthy**

```
# attrd_updater --name "#health-iowait" --update "green" --node "node1"
```

## 2.5 Cluster Resources

### 2.5.1 What is a Cluster Resource?

A *resource* is a service managed by Pacemaker. The simplest type of resource, a *primitive*, is described in this chapter. More complex forms, such as groups and clones, are described in later chapters.

Every primitive has a *resource agent* that provides Pacemaker a standardized interface for managing the service. This allows Pacemaker to be agnostic about the services it manages. Pacemaker doesn't need to understand how the service works because it relies on the resource agent to do the right thing when asked.

Every resource has a *standard* (also called *class*) specifying the interface that its resource agent follows, and a *type* identifying the specific service being managed.

### 2.5.2 Resource Standards

Pacemaker can use resource agents complying with these standards, described in more detail below:

- ocf
- lsb
- systemd
- service
- stonith

- nagios *(deprecated since 2.1.6)*
- upstart *(deprecated since 2.1.0)*

Support for some standards is controlled by build options and so might not be available in any particular build of Pacemaker. The command `crm_resource --list-standards` will show which standards are supported by the local build.

### Open Cluster Framework

The Open Cluster Framework (OCF) Resource Agent API is a ClusterLabs standard for managing services. It is the most preferred since it is specifically designed for use in a Pacemaker cluster.

OCF agents are scripts that support a variety of actions including `start`, `stop`, and `monitor`. They may accept parameters, making them more flexible than other standards. The number and purpose of parameters is left to the agent, which advertises them via the `meta-data` action.

Unlike other standards, OCF agents have a *provider* as well as a standard and type.

For more information, see the "Resource Agents" chapter of *Pacemaker Administration* and the OCF standard.

### Systemd

Most Linux distributions use Systemd for system initialization and service management. *Unit files* specify how to manage services and are usually provided by the distribution.

Pacemaker can manage systemd services. Simply create a resource with `systemd` as the resource standard and the unit file name as the resource type. Do *not* run `systemctl enable` on the unit.

**Important:** Make sure that any systemd services to be controlled by the cluster are *not* enabled to start at boot.

### Linux Standard Base

*LSB* resource agents, also known as SysV-style, are scripts that provide start, stop, and status actions for a service.

They are provided by some operating system distributions. If a full path is not given, they are assumed to be located in a directory specified when your Pacemaker software was built (usually `/etc/init.d`).

In order to be used with Pacemaker, they must conform to the LSB specification as it relates to init scripts.

**Warning:** Some LSB scripts do not fully comply with the standard. For details on how to check whether your script is LSB-compatible, see the "Resource Agents" chapter of *Pacemaker Administration*. Common problems include:

- Not implementing the `status` action
- Not observing the correct exit status codes
- Starting a started resource returns an error
- Stopping a stopped resource returns an error

**Important:** Make sure the host is *not* configured to start any LSB services at boot that will be controlled by the cluster.

#### 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:

- an LSB init script
- a Systemd unit file
- an Upstart job

### STONITH

The `stonith` standard is used for managing fencing devices, discussed later in *Fencing*.

#### Nagios Plugins

Nagios Plugins are a way to monitor services. Pacemaker can use these as resources, to react to a change in the service's status.

To use plugins as resources, Pacemaker must have been built with support, and OCF-style meta-data for the plugins must be installed on nodes that can run them. Meta-data for several common plugins is provided by the nagios-agents-metadata project.

The supported parameters for such a resource are same as the long options of the plugin.

Start and monitor actions for plugin resources are implemented as invoking the plugin. A plugin result of "OK" (0) is treated as success, a result of "WARN" (1) is treated as a successful but degraded service, and any other result is considered a failure.

A plugin resource is not going to change its status after recovery by restarting the plugin, so using them alone does not make sense with `on-fail` set (or left to default) to `restart`. Another value could make sense, for example, if you want to fence or standby nodes that cannot reach some external service.

A more common use case for plugin resources is to configure them with a `container` meta-attribute set to the name of another resource that actually makes the service available, such as a virtual machine or container.

With `container` set, the plugin resource will automatically be colocated with the containing resource and ordered after it, and the containing resource will be considered failed if the plugin resource fails. This allows monitoring of a service inside a virtual machine or container, with recovery of the virtual machine or container if the service fails.

**Warning:** Nagios support is deprecated in Pacemaker. Support will be dropped entirely at the next major release of Pacemaker.

For monitoring a service inside a virtual machine or container, the recommended alternative is to configure the virtual machine as a guest node or the container as a *bundle*. For other use cases, or when the virtual machine or container image cannot be modified, the recommended alternative is to write a custom OCF agent for the service (which may even call the Nagios plugin as part of its status action).

#### Upstart

Some Linux distributions previously used Upstart for system initialization and service management. Pacemaker is able to manage services using Upstart if the local system supports them and support was enabled when your Pacemaker software was built.

The *jobs* that specify how services are managed are usually provided by the operating system distribution.

**Important:** Make sure the host is *not* configured to start any Upstart services at boot that will be controlled by the cluster.

**Warning:** Upstart support is deprecated in Pacemaker. Upstart is no longer actively maintained, and test platforms for it are no longer readily usable. Support will be dropped entirely at the next major release of Pacemaker.

### 2.5.3 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 8: **Properties of a Primitive Resource**

| Field | Description |
|---|---|
| id | Your name for the resource |
| class | The standard the resource agent conforms to. Allowed values: `lsb`, `ocf`, `service`, `stonith`, `systemd`, `nagios` *(deprecated since 2.1.6)*, and `upstart` *(deprecated since 2.1.0)* |
| description | A description of the Resource Agent, intended for local use. E.g. `IP address for website` |
| 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:

**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!

---

**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>
```

### 2.5.4 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.

#### 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 9: **Meta-attributes of a Primitive Resource**

| Name | Type | Default | Description |
|---|---|---|---|
| priority | *score* | 0 | If not all resources can be active, the cluster will stop lower-priority resources in order to keep higher-priority ones active. |
| critical | *boolean* | true | Use this value as the default for `influence` in all *colocation constraints* involving this resource, as well as in the implicit colocation constraints created if this resource is in a *group*. For details, see *Colocation Influence*. *(since 2.1.0)* |

Continued on next page

Table 9 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| target-role | *enumeration* | Started | What state should the cluster attempt to keep this resource in? Allowed values:<br>• `Stopped:` Force the resource to be stopped<br>• `Started:` Allow the resource to be started (and in the case of *promotable* clone resources, promoted if appropriate)<br>• `Unpromoted:` Allow the resource to be started, but only in the unpromoted role if the resource is *promotable*<br>• `Promoted:` Equivalent to `Started` |
| is-managed | *boolean* | true | If false, the cluster will not start, stop, promote, or demote the resource on any node. Recurring actions for the resource are unaffected. Maintenance mode overrides this setting. |
| maintenance | *boolean* | false | If true, the cluster will not start, stop, promote, or demote the resource on any node, and will pause any recurring monitors (except those specifying `role` as `Stopped`). If true, the *maintenance-mode* cluster option or *maintenance* node attribute overrides this. |
| resource-stickiness | *score* | 1 for individual clone instances, 0 for all other resources | A score that will be added to the current node when a resource is already active. This allows running resources to stay where they are, even if they would be placed elsewhere if they were being started from a stopped state. |
| requires | *enumeration* | `quorum` for resources with a `class` of `stonith`, otherwise `unfencing` if unfencing is active in the cluster, otherwise `fencing` if `stonith-enabled` is true, otherwise `quorum` | Conditions under which the resource can be started. Allowed values:<br>• `nothing:` The cluster can always start this resource.<br>• `quorum:` The cluster can start this resource only if a majority of the configured nodes are active.<br>• `fencing:` The cluster can start this resource only if a majority of the configured nodes are active *and* any failed or unknown nodes have been *fenced*.<br>• `unfencing:` The cluster can only start this resource if a majority of the configured nodes are active *and* any failed or unknown nodes have been fenced *and* only on nodes that have been *unfenced*. |

Continued on next page

Table 9 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| migration-threshold | *score* | 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 specifies `on-fail` as `restart` (the default), and additionally for failed `start` operations, if the cluster property `start-failure-is-fatal` is `false`. |
| failure-timeout | *duration* | 0 | Ignore previously failed resource actions after this much time has passed without new failures (potentially allowing the resource back to the node on which it failed, if it previously reached its `migration-threshold` there). A value of 0 indicates that failures do not expire. **WARNING:** If this value is low, and pending cluster activity prevents the cluster from responding to a failure within that time, then the failure will be ignored completely and will not cause recovery of the resource, even if a recurring action continues to report failure. It should be at least greater than the longest *action timeout* for all resources in the cluster. A value in hours or days is reasonable. |

Continued on next page

Table 9 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| multiple-active | *enumeration* | stop_start | What should the cluster do if it ever finds the resource active on more than one node? Allowed values:<br>• `block`: mark the resource as unmanaged<br>• `stop_only`: stop all active instances and leave them that way<br>• `stop_start`: stop all active instances and start the resource in one location only<br>• `stop_unexpected`: stop all active instances except where the resource should be active (this should be used only when extra instances are not expected to disrupt existing instances, and the resource agent's monitor of an existing instance is capable of detecting any problems that could be caused; note that any resources ordered after this will still need to be restarted) *(since 2.1.3)* |
| allow-migrate | *boolean* | true for `ocf:pacemaker:remote` resources, false otherwise | Whether the cluster should try to "live migrate" this resource when it needs to be moved (see *Migrating Resources*) |
| allow-unhealthy-nodes | *boolean* | false | Whether the resource should be able to run on a node even if the node's health score would otherwise prevent it (see *Tracking Node Health*) *(since 2.1.3)* |
| container-attribute-target | *enumeration* | | Specific to bundle resources; see *Bundle Node Attributes* |
| remote-node | *text* | | The name of the Pacemaker Remote guest node this resource is associated with, if any. If specified, this both enables the resource as a guest node and defines the unique name used to identify the guest node. The guest must be configured to run the Pacemaker Remote daemon when it is started. **WARNING:** This value cannot overlap with any resource or node IDs. |
| remote-addr | *text* | value of `remote-node` | If `remote-node` is specified, the IP address or hostname used to connect to the guest via Pacemaker Remote. The Pacemaker Remote daemon on the guest must be configured to accept connections on this address. |

Continued on next page

Table 9 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| remote-port | *port* | 3121 | If `remote-node` is specified, the port on the guest used for its Pacemaker Remote connection. The Pacemaker Remote daemon on the guest must be configured to listen on this port. |
| remote-connect-timeout | *timeout* | 60s | If `remote-node` is specified, how long before a pending guest connection will time out. |
| remote-allow-migrate | *boolean* | true | If `remote-node` is specified, this acts as the `allow-migrate` meta-attribute for the implicit remote connection resource (`ocf:pacemaker:remote`). |

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:

**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>
```

In addition to the cluster-defined meta-attributes described above, you may also configure arbitrary meta-attributes of your own choosing. Most commonly, this would be done for use in *rules*. For example, an IT department might define a custom meta-attribute to indicate which company department each resource is intended for. To reduce the chance of name collisions with cluster-defined meta-attributes added in the future, it is recommended to use a unique, organization-specific prefix for such attributes.

### 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`).

### Resource Instance Attributes

The resource agents of some resource standards (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:

**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.

**Displaying the metadata for the Dummy resource agent template**

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

```xml
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="Dummy" version="2.0">
<version>1.1</version>

<longdesc lang="en">
This is a dummy OCF resource agent. It does absolutely nothing except keep track
of whether it is running or not, and can be configured so that actions fail or
take a long time. Its purpose is primarily for testing, and to serve as a
template for resource agent writers.
</longdesc>
<shortdesc lang="en">Example stateless resource agent</shortdesc>

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

<parameter name="passwd" reloadable="1">
<longdesc lang="en">
Fake password field
</longdesc>
<shortdesc lang="en">Password</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="fake" reloadable="1">
<longdesc lang="en">
Fake attribute that can be changed to cause a reload
</longdesc>
<shortdesc lang="en">Fake attribute that can be changed to cause a reload</shortdesc>
<content type="string" default="dummy" />
</parameter>

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

<parameter name="fail_start_on" reloadable="1">
<longdesc lang="en">
Start, migrate_from, and reload-agent actions will return failure if running on
the host specified here, but the resource will run successfully anyway (future
monitor calls will find it running). This can be used to test on-fail=ignore.
</longdesc>
<shortdesc lang="en">Report bogus start failure on specified host</shortdesc>
<content type="string" default="" />
</parameter>
<parameter name="envfile" reloadable="1">
<longdesc lang="en">
If this is set, the environment will be dumped to this file for every call.
</longdesc>
<shortdesc lang="en">Environment dump file</shortdesc>
<content type="string" default="" />
</parameter>

</parameters>
```

## 2.6 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.

Operations may be explicitly configured for two purposes: to override defaults for options (such as timeout) that the cluster will use whenever it initiates the operation, and to run an operation on a recurring basis (for example, to monitor the resource for failure).

**An OCF resource with a non-default start timeout**

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

Pacemaker identifies operations by a combination of name and interval, so this combination must be unique for each resource. That is, you should not configure two operations for the same resource with the same name and interval.

### 2.6.1 Operation Properties

The `id`, `name`, `interval`, and `role` operation properties may be specified only as XML attributes of the `op` element. Other operation properties may be specified in any of the following ways, from highest precedence to lowest:

- directly in the `op` element as an XML attribute
- in an `nvpair` element within a `meta_attributes` element within the `op` element
- in an `nvpair` element within a `meta_attributes` element within *operation defaults*

If not specified, the default from the table below is used.

Table 10: **Operation Properties**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique identifier for the XML element *(required)* |
| name | *text* | | An action name supported by the resource agent *(required)* |

Continued on next page

Table 10 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| interval | *duration* | 0 | If this is a positive value, Pacemaker will schedule recurring instances of this operation at the given interval (which makes sense only with *name* set to *monitor*). If this is 0, Pacemaker will apply other properties configured for this operation to instances that are scheduled as needed during normal cluster operation. *(required)* |
| role | *enumeration* | | If this is set, the operation configuration applies only on nodes where the cluster expects the resource to be in the specified role. This makes sense only for recurring monitors. Allowed values: `Started`, `Stopped`, and in the case of *promotable clone resources*, `Unpromoted` and `Promoted`. |
| timeout | *timeout* | 20s | If resource agent execution does not complete within this amount of time, the action will be considered failed. **Note:** timeouts for fencing agents are handled specially (see the *Fencing* chapter). |

Continued on next page

Table 10 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| on-fail | *enumeration* | • If `name` is `stop`: `fence` if *stonith-enabled* is true, otherwise `block`<br>• If `name` is `demote`: `on-fail` of the `monitor` action with `role` set to `Promoted`, if present, enabled, and configured to a value other than `demote`, or `restart` otherwise<br>• Otherwise: `restart` | How the cluster should respond to a failure of this action. Allowed values:<br>• `ignore`: Pretend the resource did not fail<br>• `block`: Do not perform any further operations on the resource<br>• `stop`: Stop the resource and leave it stopped<br>• `demote`: Demote the resource, without a full restart. This is valid only for `promote` actions, and for `monitor` actions with both a nonzero `interval` and `role` set to `Promoted`; for any other action, a configuration error will be logged, and the default behavior will be used. *(since 2.0.5)*<br>• `restart`: Stop the resource, and start it again if allowed (possibly on a different node)<br>• `fence`: Fence the node on which the resource failed<br>• `standby`: Put the node on which the resource failed in standby mode (forcing *all* resources away) |
| enabled | *boolean* | true | If `false`, ignore this operation definition. This does not suppress all actions of this type, but is typically used to pause a recurring monitor. This can complement the resource being unmanaged (*is-managed* set to `false`), which does not stop recurring operations. Maintenance mode, which does stop configured monitors, overrides this setting. |
| record-pending | *boolean* | true | Operation results are always recorded when the operation completes (successful or not). If this is `true`, operations will also be recorded when initiated, so that status output can indicate that the operation is in progress. |

**Note:** Only one action can be configured for any given combination of `name` and `interval`.

**Note:** When `on-fail` is set to `demote`, recovery from failure by a successful demote causes the cluster to recalculate whether and where a new instance should be promoted. The node with the failure is eligible, so if promotion scores have not changed, it will be promoted again.

There is no direct equivalent of `migration-threshold` for the promoted role, but the same effect can be achieved with a location constraint using a *rule* with a node attribute expression for the resource's fail count.

For example, to immediately ban the promoted role from a node with any failed promote or promoted instance monitor:

```
<rsc_location id="loc1" rsc="my_primitive">
    <rule id="rule1" score="-INFINITY" role="Promoted" boolean-op="or">
      <expression id="expr1" attribute="fail-count-my_primitive#promote_0"
        operation="gte" value="1"/>
      <expression id="expr2" attribute="fail-count-my_primitive#monitor_10000"
        operation="gte" value="1"/>
    </rule>
</rsc_location>
```

This example assumes that there is a promotable clone of the `my_primitive` resource (note that the primitive name, not the clone name, is used in the rule), and that there is a recurring 10-second-interval monitor configured for the promoted role (fail count attributes specify the interval in milliseconds).

### 2.6.2 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[1]. You must configure `monitor` operations explicitly to perform these checks.

**An OCF resource with a recurring health check**

```
<primitive id="Public-IP" class="ocf" type="IPaddr" provider="heartbeat">
  <operations>
     <op id="Public-IP-start" name="start" timeout="60s"/>
     <op id="Public-IP-monitor" 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>
```

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).

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

**Note:** Currently, monitors with `role=Stopped` are not implemented for *clone* resources.

### 2.6.3 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.

### 2.6.4 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.

**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>
```

### 2.6.5 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.

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

```xml
<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>
```

### 2.6.6 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 of an OCF resource with a disabled health check**

```xml
<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"/>'
```

### 2.6.7 Specifying When Recurring Actions are Performed

By default, recurring actions are scheduled relative to when the resource started. In some cases, you might prefer that a recurring action start relative to a specific date and time. For example, you might schedule an in-depth monitor to run once every 24 hours, and want it to run outside business hours.

To do this, set the operation's `interval-origin`. The cluster uses this point to calculate the correct `start-delay` such that the operation will occur at `interval-origin` plus a multiple of the operation interval.

For example, if the recurring 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 after 11 hours and 28 minutes.

The value specified for `interval` and `interval-origin` can be any date/time conforming to the ISO8601 standard. By way of example, to specify an operation that would run on the first Monday of 2021 and every Monday after that, you would add:

**Example recurring action that runs relative to base date/time**

```
<op id="intensive-monitor" name="monitor" interval="P7D" interval-origin="2021-W01-1"/>
```

### 2.6.8 Handling Resource Failure

By default, Pacemaker will attempt to recover failed resources by restarting them. However, failure recovery is highly configurable.

#### Failure Counts

Pacemaker tracks resource failures for each combination of node, resource, and operation (start, stop, monitor, etc.).

You can query the fail count for a particular node, resource, and/or operation using the `crm_failcount` command. For example, to see how many times the 10-second monitor for `myrsc` has failed on `node1`, run:

```
# crm_failcount --query -r myrsc -N node1 -n monitor -I 10s
```

If you omit the node, `crm_failcount` will use the local node. If you omit the operation and interval, `crm_failcount` will display the sum of the fail counts for all operations on the resource.

You can use `crm_resource --cleanup` or `crm_failcount --delete` to clear fail counts. For example, to clear the above monitor failures, run:

```
# crm_resource --cleanup -r myrsc -N node1 -n monitor -I 10s
```

If you omit the resource, `crm_resource --cleanup` will clear failures for all resources. If you omit the node, it will clear failures on all nodes. If you omit the operation and interval, it will clear the failures for all operations on the resource.

**Note:** Even when cleaning up only a single operation, all failed operations will disappear from the status display. This allows us to trigger a re-check of the resource's current status.

Higher-level tools may provide other commands for querying and clearing fail counts.

The `crm_mon` tool shows the current cluster status, including any failed operations. To see the current fail counts for any failed resources, call `crm_mon` with the `--failcounts` option. This shows the fail counts per resource (that is, the sum of any operation fail counts for the resource).

#### Failure Response

Normally, if a running resource fails, pacemaker will try to stop it and start it again. Pacemaker will choose the best location to start it each time, which may be the same node that it failed on.

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 meta-attribute.[2]

If you define `migration-threshold` to *N* for a resource, it will be banned from the original node after *N* failures there.

**Note:** The `migration-threshold` is per *resource*, even though fail counts are tracked per *operation*. The operation fail counts are added together to compare against the `migration-threshold`.

By default, fail counts remain until manually cleared by an administrator using `crm_resource --cleanup` or `crm_failcount --delete` (hopefully after first fixing the failure's cause). It is possible to have fail counts expire automatically by setting the `failure-timeout` resource meta-attribute.

**Important:** A successful operation does not clear past failures. If a recurring monitor operation fails once, succeeds many times, then fails again days later, its fail count is 2. Fail counts are cleared only by manual intervention or failure timeout.

For example, setting `migration-threshold` to 2 and `failure-timeout` to `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.

**Note:** `failure-timeout` is measured since the most recent failure. That is, older failures do not individually time out and lower the fail count. Instead, all failures are timed out simultaneously (and the fail count is reset to 0) if there is no new failure for the timeout period.

There are two exceptions to the migration threshold: 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 fail count 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 fencing is enabled, then the cluster will fence the node in order to be able to start the resource elsewhere. If fencing is disabled, 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 any failure timeout or clearing.

### 2.6.9 Reloading an Agent After a Definition Change

The cluster automatically detects changes to the configuration of active resources. The cluster's normal response is to stop the service (using the old definition) and start it again (with the new definition). This works, but some resource agents 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:

- Implement the `reload-agent` action. What it should do depends completely on your application!

  **Note:** Resource agents may also implement a `reload` action to make the managed service reload its own *native* configuration. This is different from `reload-agent`, which makes effective changes in the

[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.

resource's *Pacemaker* configuration (specifically, the values of the agent's reloadable parameters).

- Advertise the `reload-agent` operation in the `actions` section of its meta-data.
- Set the `reloadable` attribute to 1 in the `parameters` section of its meta-data for any parameters eligible to be reloaded after a change.

Once these requirements are satisfied, the cluster will automatically know to reload the resource (instead of restarting) when a reloadable parameter changes.

**Note:** Metadata will not be re-read unless the resource needs to be started. If you edit the agent of an already active resource to set a parameter reloadable, the resource may restart the first time the parameter value changes.

**Note:** If both a reloadable and non-reloadable parameter are changed simultaneously, the resource will be restarted.

### 2.6.10 Migrating Resources

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

However, some types of resources, such as many virtual machines, 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 live 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. Even 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 is it 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 agent standard must be OCF.
- 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 meta-data.
- 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 scheduler is not yet able to model this situation correctly and so takes the safer (if less optimal) path.

Also, if a migratable resource depends on a non-migratable resource, and both need to be moved, the migratable resource will be restarted.

## 2.7 Resource Constraints

### 2.7.1 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.

#### Location Properties

Table 11: **Attributes of a rsc_location Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for the constraint (required) |
| rsc | *id* | | The name of the resource to which this constraint applies. A location constraint must either have a `rsc`, have a `rsc-pattern`, or contain at least one resource set. |
| rsc-pattern | *text* | | A pattern matching the names of resources to which this constraint applies. The syntax is the same as POSIX extended regular expressions, with the addition of an initial `!` indicating that resources *not* matching the pattern are selected. If the regular expression contains submatches, and the constraint contains a *rule*, the submatches can be referenced as `%1` through `%9` in the rule's `score-attribute` or a rule expression's `attribute` (see *Specifying location scores using pattern submatches*). A location constraint must either have a `rsc`, have a `rsc-pattern`, or contain at least one resource set. |

Continued on next page

Table 11 – continued from previous page

<table>
<tr><th>Name</th><th>Type</th><th>Default</th><th>Description</th></tr>
<tr><td>node</td><td>text</td><td></td><td>The name of the node to which this constraint applies. A location constraint must either have a <code>node</code> and <code>score</code>, or contain at least one rule.</td></tr>
<tr><td>score</td><td>score</td><td></td><td>Positive values indicate a preference for running the affected resource(s) on <code>node</code> – the higher the value, the stronger the preference. Negative values indicate the resource(s) should avoid this node (a value of -INFINITY changes “should” to “must”). A location constraint must either have a <code>node</code> and <code>score</code>, or contain at least one rule.</td></tr>
<tr><td>role</td><td>enumeration</td><td><code>Started</code></td><td>This is significant only for promotable clones, is allowed only if <code>rsc</code> or <code>rsc-pattern</code> is set, and is ignored if the constraint contains a rule. Allowed values:<ul><li><code>Started</code> or <code>Unpromoted</code>: The constraint affects the location of all instances of the resource. (A promoted instance must start in the unpromoted role before being promoted, so any location requirement for unpromoted instances also affects promoted instances.)</li><li><code>Promoted</code>: The constraint does not affect the location of instances, but instead affects which of the instances will be promoted.</li></ul></td></tr>
<tr><td>resource-discovery</td><td>enumeration</td><td>always</td><td>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 of nodes, limiting resource discovery to allowed nodes can significantly boost performance. Allowed values:<ul><li><code>always:</code> Always perform resource discovery for the specified resource on this node.</li><li><code>never:</code> Never perform resource discovery for the specified resource on this node. This option should generally be used with a -INFINITY score, although that is not strictly required.</li><li><code>exclusive:</code> Perform resource discovery for the specified resource only on this node (and other nodes similarly marked as <code>exclusive</code>). Multiple location constraints using <code>exclusive</code> discovery for the same resource across different nodes creates a subset of nodes resource-discovery is exclusive to. If a resource is marked for <code>exclusive</code> discovery on one or more nodes, that resource is only allowed to be placed within that subset of nodes.</li></ul></td></tr>
</table>

> **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).

### 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.

**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>
```

### 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.

**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-do-not-run" rsc="Webserver" node="sles-2" score="-INFINITY"/>
    <rsc_location id="loc-3-do-not-run" rsc="Database" node="sles-1" score="-INFINITY"/>
    <rsc_location id="loc-4" rsc="Database" node="sles-2" score="200"/>
</constraints>
```

### 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.

**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.

#### Specifying locations using pattern matching

A location constraint can affect all resources whose IDs match a given pattern. The following example bans resources named **ip-httpd**, **ip-asterisk**, **ip-gateway**, etc., from **node1**.

**Location constraint banning all resources matching a pattern from one node**

```
<constraints>
    <rsc_location id="ban-ips-from-node1" rsc-pattern="ip-.*" node="node1" score="-INFINITY"/>
</constraints>
```

### 2.7.2 Specifying the Order in which Resources Should Start/Stop

*Ordering constraints* tell the cluster the order in which certain resource actions should occur.

**Important:** Ordering constraints affect *only* the ordering of resource actions; 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 *Placing Resources Relative to other Resources*), or alternatively, a group (see *Groups - A Syntactic Shortcut*).

#### Ordering Properties

Table 12: **Attributes of a rsc_order Element**

| 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`. |

Continued on next page

Table 12 – continued from previous page

| Field | Default | Description |
|---|---|---|
| kind | Mandatory | How to enforce the constraint. Allowed values:<br>• `Mandatory`: `then-action` will never be initiated for the `then` resource unless and until `first-action` successfully completes for the `first` resource.<br>• `Optional`: The constraint applies only if both specified resource actions are scheduled in the same transition (that is, in response to the same cluster state). This means that `then-action` is allowed on the `then` resource regardless of the state of the `first` resource, but if both actions happen to be scheduled at the same time, they will be ordered.<br>• `Serialize`: Ensure that the specified actions are never performed concurrently for the specified resources. `First-action` and `then-action` can be executed in either order, but one must complete before the other can be initiated. An example use case is when resource start-up puts a high load on the host. |
| symmetrical | TRUE for `Mandatory` and `Optional` kinds. FALSE for `Serialize` kind. | 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). `Serialize` orders cannot be symmetrical. |

`Promote` and `demote` apply to *promotable* clone resources.

### 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:

**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.

### Symmetric and asymmetric ordering

A mandatory symmetric ordering of "start A then start B" implies not only that the start actions must be ordered, but that B is not allowed to be active unless A is active. For example, if the ordering is added to the configuration when A is stopped (due to target-role, failure, etc.) and B is already active, then B will be stopped.

By contrast, asymmetric ordering of "start A then start B" means the stops can occur in either order, which implies that B *can* remain active in the same situation.

### 2.7.3 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.

**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 *Specifying the Order in which Resources Should Start/Stop*) *and* a colocation constraint, or alternatively, a group (see *Groups - A Syntactic Shortcut*).

#### Colocation Properties

Table 13: **Attributes of a rsc_colocation Constraint**

| Field | Default | Description |
|---|---|---|
| id | | A unique name for the constraint (required). |
| rsc | | The name of a resource that should be located relative to `with-rsc`. A colocation constraint must either contain at least one *resource set*, or specify both `rsc` and `with-rsc`. |
| 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`. A colocation constraint must either contain at least one *resource set*, or specify both `rsc` and `with-rsc`. |
| node-attribute | #uname | If `rsc` and `with-rsc` are specified, this node attribute must be the same on the node running `rsc` and the node running `with-rsc` for the constraint to be satisfied. (For details, see *Colocation by Node Attribute*.) |
| score | 0 | 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". |
| rsc-role | Started | If `rsc` and `with-rsc` are specified, and `rsc` is a *promotable clone*, the constraint applies only to `rsc` instances in this role. Allowed values: `Started`, `Stopped`, `Promoted`, `Unpromoted`. For details, see *Promotable Clone Constraints*. |
| with-rsc-role | Started | If `rsc` and `with-rsc` are specified, and `with-rsc` is a *promotable clone*, the constraint applies only to `with-rsc` instances in this role. Allowed values: `Started`, `Stopped`, `Promoted`, `Unpromoted`. For details, see *Promotable Clone Constraints*. |

Continued on next page

[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.

Table 13 – continued from previous page

| Field | Default | Description |
|---|---|---|
| influence | value of `critical` meta-attribute for `rsc` | Whether to consider the location preferences of `rsc` when `with-rsc` is already active. Allowed values: `true`, `false`. For details, see *Colocation Influence*. *(since 2.1.0)* |

### 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:

**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"`.

**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.

### Advisory Placement

If mandatory placement is about "must" and "must not", then advisory placement is the "I'd prefer if" alternative.

For colocation constraints with scores greater than **-INFINITY** and less than **INFINITY**, the cluster will try to accommodate your wishes, but may ignore them if other factors outweigh the colocation score. Those factors might include other constraints, resource stickiness, failure thresholds, whether other resources would be prevented from being active, etc.

**Advisory colocation constraint for two resources**

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

### Colocation by Node Attribute

The `node-attribute` property of a colocation constraints allows you to express the requirement, "these resources must be on similar nodes".

As an example, imagine that you have two Storage Area Networks (SANs) that are not controlled by the cluster, and each node is connected to one or the other. You may have two resources **r1** and **r2** such that **r2** needs to use the same SAN as **r1**, but doesn't necessarily have to be on the same exact node. In such a case, you could define a *node attribute* named **san**, with the value **san1** or **san2** on each node as appropriate. Then, you could colocate **r2** with **r1** using `node-attribute` set to **san**.

### Colocation Influence

By default, if A is colocated with B, the cluster will take into account A's preferences when deciding where to place B, to maximize the chance that both resources can run.

For a detailed look at exactly how this occurs, see Colocation Explained.

However, if `influence` is set to `false` in the colocation constraint, this will happen only if B is inactive and needing to be started. If B is already active, A's preferences will have no effect on placing B.

An example of what effect this would have and when it would be desirable would be a nonessential reporting tool colocated with a resource-intensive service that takes a long time to start. If the reporting tool fails enough times to reach its migration threshold, by default the cluster will want to move both resources to another node if possible. Setting `influence` to `false` on the colocation constraint would mean that the reporting tool would be stopped in this situation instead, to avoid forcing the service to move.

The `critical` resource meta-attribute is a convenient way to specify the default for all colocation constraints and groups involving a particular resource.

**Note:** If a noncritical resource is a member of a group, all later members of the group will be treated as noncritical, even if they are marked as (or left to default to) critical.

## 2.7.4 Resource Sets

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

**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 *Ordering Sets of Resources*), `rsc_colocation` (see *Colocating Sets of Resources*), and `rsc_ticket` (see *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 14: **Attributes of a resource_set Element**

| Field | Default | Description |
|---|---|---|
| id | | A unique name for the set (required) |
| 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. With the current implementation, the cluster may continue even if only one member of the set is started, but if more than one member of the set is starting at the same time, the cluster will still wait until all of those have started before continuing (this may change in future versions). Meaningful within `rsc_order`. |
| role | | The constraint applies only to resource set members that are *Promotable clones* in this role. Meaningful within `rsc_location`, `rsc_colocation` and `rsc_ticket`. Allowed values: `Started`, `Promoted`, `Unpromoted`. For details, see *Promotable Clone Constraints*. |
| action | start | The action that applies to *all members* of the set. Meaningful within `rsc_order`. Allowed values: `start`, `stop`, `promote`, `demote`. |
| score | | *Advanced use only.* Use a specific score for this set within the constraint. |

## 2.7.5 Ordering Sets of Resources

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

**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>
```

**Visual representation of the four resources' start order for the above constraints**


### Ordered Set

To simplify this situation, *Resource Sets* can be used within ordering constraints:

**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**.

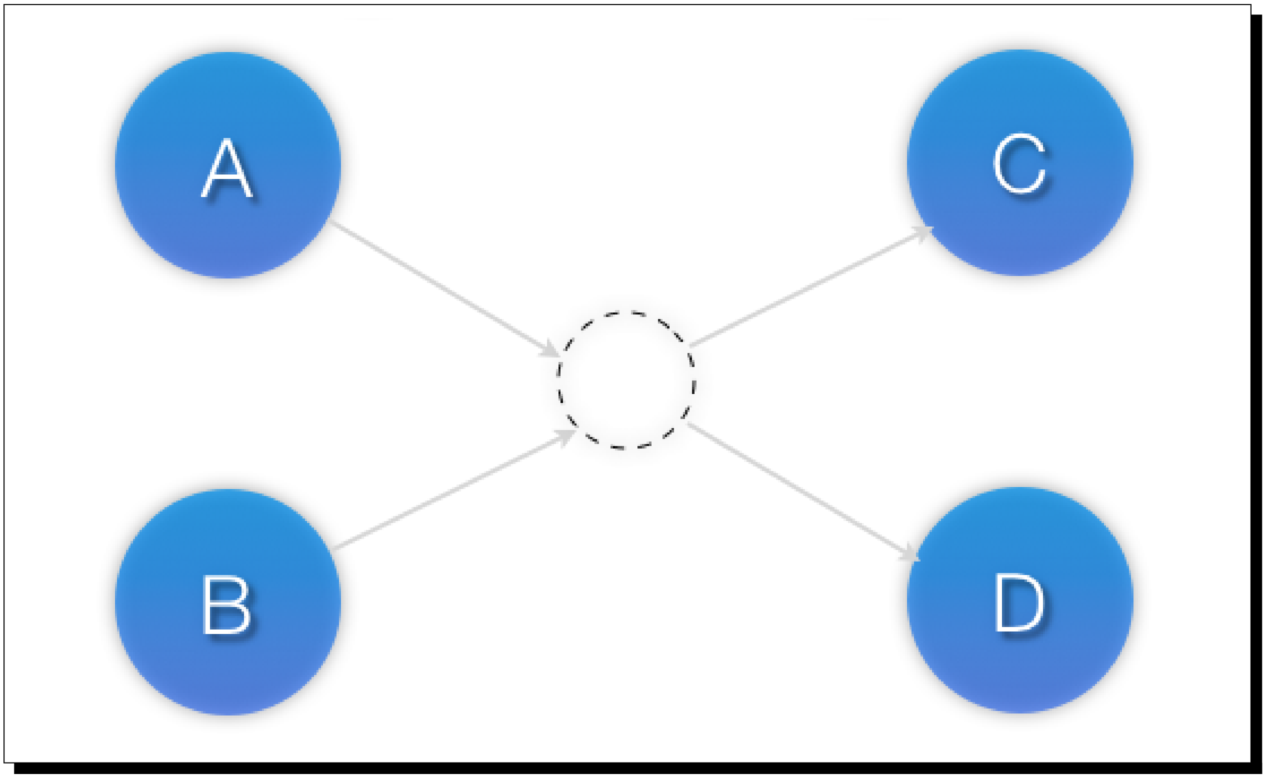
### Ordering Multiple Sets

The syntax can be expanded to allow sets of resources to be ordered relative to each other, where the members of each individual set may be ordered or unordered (controlled by the `sequential` property). 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.

**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>
```

**Visual representation of the start order for two ordered sets of unordered resources**

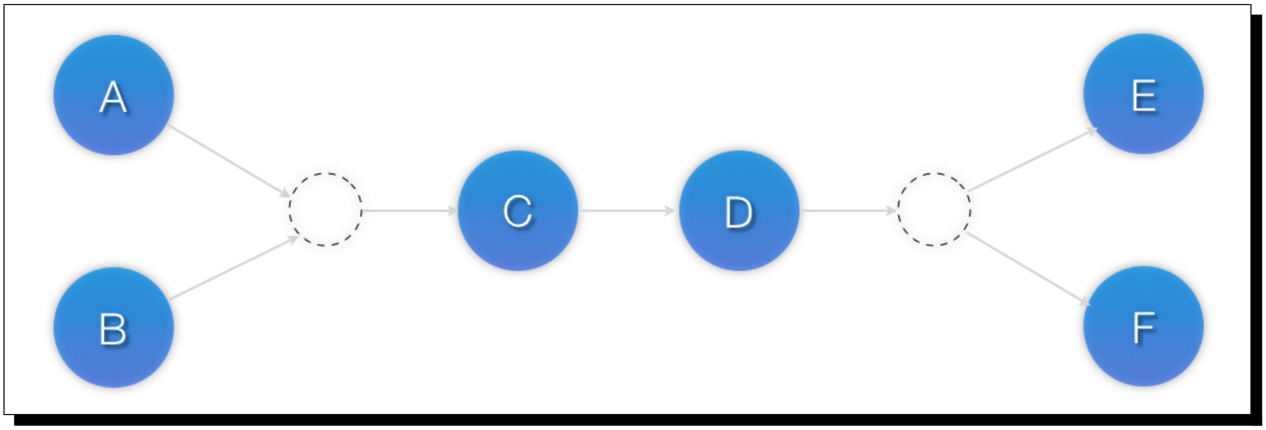



Of course either set – or both sets – of resources can also be internally ordered (by setting `sequential="true"`) and there is no limit to the number of sets that can be specified.

**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>
```

**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.

### 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.

**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.

### 2.7.6 Colocating Sets of Resources

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

The simplest way to do this is to define a resource group (see *Groups - A Syntactic Shortcut*), but that cannot always accurately express the desired relationships. For example, maybe the resources do not need to be ordered.

Another way would be to define each relationship as an individual constraint, but that causes a difficult-to-follow constraint explosion as the number of resources and combinations grow.

**Colocation chain as individual constraints, where A is placed first, then B, then C, then D**

```
<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 express complicated relationships with a simplified syntax[2], *resource sets* can be used within colocation constraints.

**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>
```

**Note:** Within a `resource_set`, the resources are listed in the order they are *placed*, which is the reverse of the order in which they are *colocated*. In the above example, resource **A** is placed before resource **B**, which is the same as saying resource **B** is colocated with resource **A**.

As with individual constraints, a resource that can't be active prevents any resource that must be colocated with it from being active. In both of the two previous examples, if **B** is unable to run, then both **C** and by inference **D** must remain stopped.

**Important:** If you use a higher-level tool, pay attention to how it exposes this functionality. Depending

[2] which is not the same as saying easy to follow

on the tool, creating a set **A B** may be equivalent to **A with B**, or **B with A**.

---

Resource sets can also be used to tell the cluster that entire *sets* of resources must be colocated relative to each other, while the individual members within any one set may or may not be colocated relative to each other (determined by the set's `sequential` property).

In the following example, resources **B**, **C**, and **D** will each be colocated with **A** (which will be placed first). **A** must be able to run in order for any of the resources to run, but any of **B**, **C**, or **D** may be stopped without affecting any of the others.

**Using colocated sets to specify a shared dependency**

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

---

**Note:** Pay close attention to the order in which resources and sets are listed. While the members of any one sequential set are placed first to last (i.e., the colocation dependency is last with first), multiple sets are placed last to first (i.e. the colocation dependency is first with last).

---

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

---

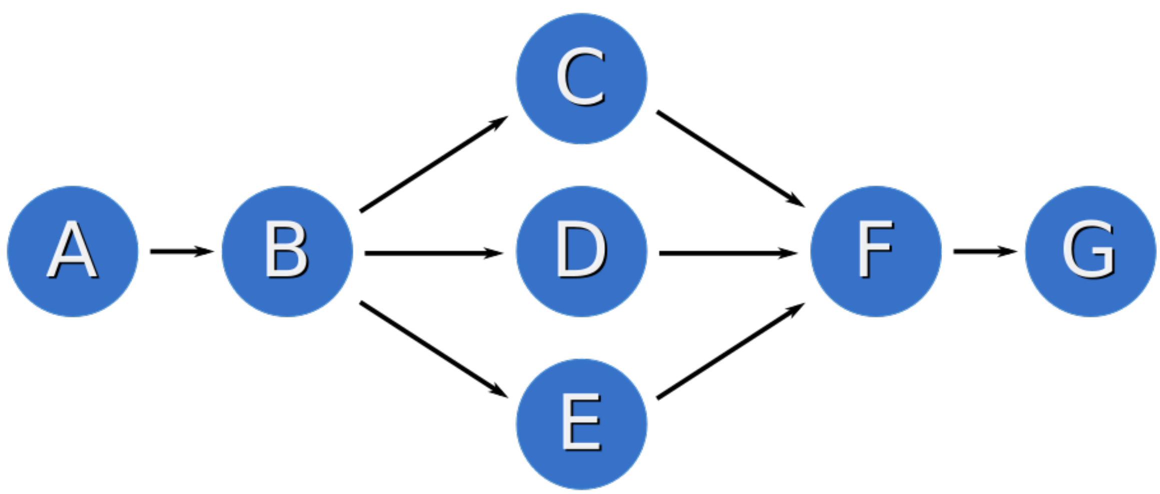
There is no inherent limit to the number and size of the sets used. The only thing that matters is that in order for any member of one set in the constraint to be active, all members of sets listed after it must also be active (and naturally on the same node); and if a set has `sequential="true"`, then in order for one member of that set to be active, all members listed before it must also be active.

If desired, you can restrict the dependency to instances of promotable clone resources that are in a specific role, using the set's `role` property.

**Colocation in which the members of the middle set have no interdependencies, and the last set listed applies only to promoted instances**

```
<constraints>
    <rsc_colocation id="coloc-1" score="INFINITY" >
      <resource_set id="colocated-set-1" sequential="true">
        <resource_ref id="F"/>
        <resource_ref id="G"/>
      </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="Promoted">
        <resource_ref id="A"/>
        <resource_ref id="B"/>
      </resource_set>
    </rsc_colocation>
</constraints>
```

**Visual representation of the above example (resources are placed from left to right)**




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

### 2.7.7 External Resource Dependencies

Sometimes, a resource will depend on services that are not managed by the cluster. An example might be a resource that requires a file system that is not managed by the cluster but mounted by systemd at boot time.

To accommodate this, the pacemaker systemd service depends on a normally empty target called `resource-agents-deps.target`. The system administrator may create a unit drop-in for that target specifying the dependencies, to ensure that the services are started before Pacemaker starts and stopped after Pacemaker stops.

Typically, this is accomplished by placing a unit file in the `/etc/systemd/system/resource-agents-deps.target.d` directory, with directives such as `Requires` and `After` specifying the dependencies as needed.

# 2.8 Fencing

## 2.8.1 What Is Fencing?

*Fencing* is the ability to make a node unable to run resources, even when that node is unresponsive to cluster commands.

Fencing is also known as *STONITH*, an acronym for "Shoot The Other Node In The Head", since the most common fencing method is cutting power to the node. Another method is "fabric fencing", cutting the node's access to some capability required to run resources (such as network access or a shared disk).

## 2.8.2 Why Is Fencing Necessary?

Fencing protects your data from being corrupted by malfunctioning nodes or unintentional concurrent access to shared resources.

Fencing protects against the "split brain" failure scenario, where cluster nodes have lost the ability to reliably communicate with each other but are still able to run resources. If the cluster just assumed that uncommunicative nodes were down, then multiple instances of a resource could be started on different nodes.

The effect of split brain depends on the resource type. For example, an IP address brought up on two hosts on a network will cause packets to randomly be sent to one or the other host, rendering the IP useless. For a database or clustered file system, the effect could be much more severe, causing data corruption or divergence.

Fencing is also used when a resource cannot otherwise be stopped. If a resource fails to stop on a node, it cannot be started on a different node without risking the same type of conflict as split-brain. Fencing the original node ensures the resource can be safely started elsewhere.

Users may also configure the `on-fail` property of *Resource Operations* or the `loss-policy` property of *ticket constraints* to `fence`, in which case the cluster will fence the resource's node if the operation fails or the ticket is lost.

## 2.8.3 Fence Devices

A *fence device* or *fencing device* is a special type of resource that provides the means to fence a node.

Examples of fencing devices include intelligent power switches and IPMI devices that accept SNMP commands to cut power to a node, and iSCSI controllers that allow SCSI reservations to be used to cut a node's access to a shared disk.

Since fencing devices will be used to recover from loss of networking connectivity to other nodes, it is essential that they do not rely on the same network as the cluster itself, otherwise that network becomes a single point of failure.

Since loss of a node due to power outage is indistinguishable from loss of network connectivity to that node, it is also essential that at least one fence device for a node does not share power with that node. For example, an on-board IPMI controller that shares power with its host should not be used as the sole fencing device for that host.

Since fencing is used to isolate malfunctioning nodes, no fence device should rely on its target functioning properly. This includes, for example, devices that ssh into a node and issue a shutdown command (such devices might be suitable for testing, but never for production).

### 2.8.4 Fence Agents

A *fence agent* or *fencing agent* is a `stonith`-class resource agent.

The fence agent standard provides commands (such as `off` and `reboot`) that the cluster can use to fence nodes. As with other resource agent classes, this allows a layer of abstraction so that Pacemaker doesn't need any knowledge about specific fencing technologies – that knowledge is isolated in the agent.

Pacemaker supports two fence agent standards, both inherited from no-longer-active projects:

- Red Hat Cluster Suite (RHCS) style: These are typically installed in `/usr/sbin` with names starting with `fence_`.
- Linux-HA style: These typically have names starting with `external/`. Pacemaker can support these agents using the **fence_legacy** RHCS-style agent as a wrapper, *if* support was enabled when Pacemaker was built, which requires the `cluster-glue` library.

### 2.8.5 When a Fence Device Can Be Used

Fencing devices do not actually "run" like most services. Typically, they just provide an interface for sending commands to an external device.

Additionally, fencing may be initiated by Pacemaker, by other cluster-aware software such as DRBD or DLM, or manually by an administrator, at any point in the cluster life cycle, including before any resources have been started.

To accommodate this, Pacemaker does not require the fence device resource to be "started" in order to be used. Whether a fence device is started or not determines whether a node runs any recurring monitor for the device, and gives the node a slight preference for being chosen to execute fencing using that device.

By default, any node can execute any fencing device. If a fence device is disabled by setting its `target-role` to `Stopped`, then no node can use that device. If a location constraint with a negative score prevents a specific node from "running" a fence device, then that node will never be chosen to execute fencing using the device. A node may fence itself, but the cluster will choose that only if no other nodes can do the fencing.

A common configuration scenario is to have one fence device per target node. In such a case, users often configure anti-location constraints so that the target node does not monitor its own device.

### 2.8.6 Limitations of Fencing Resources

Fencing resources have certain limitations that other resource classes don't:

- They may have only one set of meta-attributes and one set of instance attributes.
- If *Rules* are used to determine fencing resource options, these might be evaluated only when first read, meaning that later changes to the rules will have no effect. Therefore, it is better to avoid confusion and not use rules at all with fencing resources.

These limitations could be revisited if there is sufficient user demand.

### 2.8.7 Special Meta-Attributes for Fencing Resources

The table below lists special resource meta-attributes that may be set for any fencing resource.

Table 15: **Additional Properties of Fencing Resources**

| Field | Type | Default | Description |
|---|---|---|---|
| provides | string | | Any special capability provided by the fence device. Currently, only one such capability is meaningful: *unfencing*. |

### 2.8.8 Special Instance Attributes for Fencing Resources

The table below lists special instance attributes that may be set for any fencing resource (*not* meta-attributes, even though they are interpreted by Pacemaker rather than the fence agent). These are also listed in the man page for `pacemaker-fenced`.

Table 16: **Additional Properties of Fencing Resources**

| Name | Type | Default | Description |
|---|---|---|---|
| stonith-timeout | *timeout* | | This is not used by Pacemaker (see the `pcmk_reboot_timeout`, `pcmk_off_timeout`, etc., properties instead), but it may be used by Linux-HA fence agents. |
| pcmk_host_map | *text* | | A mapping of node names to ports for devices that do not understand the node names. For example, `node1:1;node2:2,3` tells the cluster to use port 1 for `node1` and ports 2 and 3 for `node2`. If `pcmk_host_check` is explicitly set to `static-list`, either this or `pcmk_host_list` must be set. The port portion of the map may contain special characters such as spaces if preceded by a backslash *(since 2.1.2)*. |
| pcmk_host_list | *text* | | Comma-separated list of nodes that can be targeted by this device (for example, `node1,node2,node3`). If pcmk_host_check is `static-list`, either this or `pcmk_host_map` must be set. |
| pcmk_host_check | *text* | See *Default Check Type* | The method Pacemaker should use to determine which nodes can be targeted by this device. Allowed values:<br>• `static-list:` targets are listed in the `pcmk_host_list` or `pcmk_host_map` attribute<br>• `dynamic-list:` query the device via the agent's `list` action<br>• `status:` query the device via the agent's `status` action<br>• `none:` assume the device can fence any node |

Continued on next page

Table 16 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| pcmk_delay_max | *duration* | 0s | Enable a delay of no more than the time specified before executing fencing actions. Pacemaker derives the overall delay by taking the value of pcmk_delay_base and adding a random delay value such that the sum is kept below this maximum. This is sometimes used in two-node clusters to ensure that the nodes don't fence each other at the same time. |
| pcmk_delay_base | *text* | 0s | Enable a static delay before executing fencing actions. This can be used, for example, in two-node clusters to ensure that the nodes don't fence each other, by having separate fencing resources with different values. The node that is fenced with the shorter delay will lose a fencing race. The overall delay introduced by pacemaker is derived from this value plus a random delay such that the sum is kept below the maximum delay. A single device can have different delays per node using a host map *(since 2.1.2)*, for example `node1:0s;node2:5s.` |
| pcmk_action_limit | *integer* | 1 | The maximum number of actions that can be performed in parallel on this device. A value of -1 means unlimited. Node fencing actions initiated by the cluster (as opposed to an administrator running the `stonith_admin` tool or the fencer running recurring device monitors and `status` and `list` commands) are additionally subject to the `concurrent-fencing` cluster property. |
| pcmk_host_argument | *text* | `port` otherwise `plug` if supported according to the metadata of the fence agent | *Advanced use only.* Which parameter should be supplied to the fence agent to identify the node to be fenced. Some devices support neither the standard `plug` nor the deprecated `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 | *text* | `reboot` | *Advanced use only.* 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 | *timeout* | 60s | *Advanced use only.* Specify an alternate timeout (in seconds) 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. |

Continued on next page

Table 16 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| pcmk_reboot_retries | *integer* | 2 | *Advanced use only.* The maximum number of times to retry the `reboot` 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_off_action | *text* | `off` | *Advanced use only.* 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 | *timeout* | 60s | *Advanced use only.* Specify an alternate timeout (in seconds) 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 | *Advanced use only.* 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 | *text* | `list` | *Advanced use only.* 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 | *timeout* | 60s | *Advanced use only.* Specify an alternate timeout (in seconds) 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 | *Advanced use only.* 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. |

Continued on next page

Table 16 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| pcmk_monitor_action | *text* | `monitor` | *Advanced use only.* The command to send to the resource agent in order to report extended 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 | *timeout* | 60s | *Advanced use only.* Specify an alternate timeout (in seconds) 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 | *Advanced use only.* 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 | *text* | `status` | *Advanced use only.* 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 | *timeout* | 60s | *Advanced use only.* Specify an alternate timeout (in seconds) 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 | *Advanced use only.* 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. |

### 2.8.9 Default Check Type

If the user does not explicitly configure `pcmk_host_check` for a fence device, a default value appropriate to other configured parameters will be used:

- If either `pcmk_host_list` or `pcmk_host_map` is configured, `static-list` will be used;
- otherwise, if the fence device supports the `list` action, and the first attempt at using `list` succeeds, `dynamic-list` will be used;

- otherwise, if the fence device supports the `status` action, `status` will be used;
- otherwise, `none` will be used.

### 2.8.10 Unfencing

With fabric fencing (such as cutting network or shared disk access rather than power), it is expected that the cluster will fence the node, and then a system administrator must manually investigate what went wrong, correct any issues found, then reboot (or restart the cluster services on) the node.

Once the node reboots and rejoins the cluster, some fabric fencing devices require an explicit command to restore the node's access. This capability is called *unfencing* and is typically implemented as the fence agent's `on` command.

If any cluster resource has `requires` set to `unfencing`, then that resource will not be probed or started on a node until that node has been unfenced.

### 2.8.11 Fencing and Quorum

In general, a cluster partition may execute fencing only if the partition has quorum, and the `stonith-enabled` cluster property is set to true. However, there are exceptions:

- The requirements apply only to fencing initiated by Pacemaker. If an administrator initiates fencing using the `stonith_admin` command, or an external application such as DLM initiates fencing using Pacemaker's C API, the requirements do not apply.
- A cluster partition without quorum is allowed to fence any active member of that partition. As a corollary, this allows a `no-quorum-policy` of `suicide` to work.
- If the `no-quorum-policy` cluster property is set to `ignore`, then quorum is not required to execute fencing of any node.

### 2.8.12 Fencing Timeouts

Fencing timeouts are complicated, since a single fencing operation can involve many steps, each of which may have a separate timeout.

Fencing may be initiated in one of several ways:

- An administrator may initiate fencing using the `stonith_admin` tool, which has a `--timeout` option (defaulting to 2 minutes) that will be used as the fence operation timeout.
- An external application such as DLM may initiate fencing using the Pacemaker C API. The application will specify the fence operation timeout in this case, which might or might not be configurable by the user.
- The cluster may initiate fencing itself. In this case, the `stonith-timeout` cluster property (defaulting to 1 minute) will be used as the fence operation timeout.

However fencing is initiated, the initiator contacts Pacemaker's fencer (`pacemaker-fenced`) to request fencing. This connection and request has its own timeout, separate from the fencing operation timeout, but usually happens very quickly.

The fencer will contact all fencers in the cluster to ask what devices they have available to fence the target node. The fence operation timeout will be used as the timeout for each of these queries.

Once a fencing device has been selected, the fencer will check whether any action-specific timeout has been configured for the device, to use instead of the fence operation timeout. For example, if `stonith-timeout` is 60 seconds, but the fencing device has `pcmk_reboot_timeout` configured as 90 seconds, then a timeout of 90 seconds will be used for reboot actions using that device.

A device may have retries configured, in which case the timeout applies across all attempts. For example, if a device has `pcmk_reboot_retries` configured as 2, and the first reboot attempt fails, the second attempt will only have whatever time is remaining in the action timeout after subtracting how much time the first attempt used. This means that if the first attempt fails due to using the entire timeout, no further attempts will be made. There is currently no way to configure a per-attempt timeout.

If more than one device is required to fence a target, whether due to failure of the first device or a fencing topology with multiple devices configured for the target, each device will have its own separate action timeout.

For all of the above timeouts, the fencer will generally multiply the configured value by 1.2 to get an actual value to use, to account for time needed by the fencer's own processing.

Separate from the fencer's timeouts, some fence agents have internal timeouts for individual steps of their fencing process. These agents often have parameters to configure these timeouts, such as `login-timeout`, `shell-timeout`, or `power-timeout`. Many such agents also have a `disable-timeout` parameter to ignore their internal timeouts and just let Pacemaker handle the timeout. This causes a difference in retry behavior. If `disable-timeout` is not set, and the agent hits one of its internal timeouts, it will report that as a failure to Pacemaker, which can then retry. If `disable-timeout` is set, and Pacemaker hits a timeout for the agent, then there will be no time remaining, and no retry will be done.

### 2.8.13 Fence Devices Dependent on Other Resources

In some cases, a fence device may require some other cluster resource (such as an IP address) to be active in order to function properly.

This is obviously undesirable in general: fencing may be required when the depended-on resource is not active, or fencing may be required because the node running the depended-on resource is no longer responding.

However, this may be acceptable under certain conditions:

- The dependent fence device should not be able to target any node that is allowed to run the depended-on resource.
- The depended-on resource should not be disabled during production operation.
- The `concurrent-fencing` cluster property should be set to `true`. Otherwise, if both the node running the depended-on resource and some node targeted by the dependent fence device need to be fenced, the fencing of the node running the depended-on resource might be ordered first, making the second fencing impossible and blocking further recovery. With concurrent fencing, the dependent fence device might fail at first due to the depended-on resource being unavailable, but it will be retried and eventually succeed once the resource is brought back up.

Even under those conditions, there is one unlikely problem scenario. The DC always schedules fencing of itself after any other fencing needed, to avoid unnecessary repeated DC elections. If the dependent fence device targets the DC, and both the DC and a different node running the depended-on resource need to be fenced, the DC fencing will always fail and block further recovery. Note, however, that losing a DC node entirely causes some other node to become DC and schedule the fencing, so this is only a risk when a stop or other operation with `on-fail` set to `fencing` fails on the DC.

### 2.8.14 Configuring Fencing

Higher-level tools can provide simpler interfaces to this process, but using Pacemaker command-line tools, this is how you could configure a fence device.

1. Find the correct driver:

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

**Note:** You may have to install packages to make fence agents available on your host. Searching your available packages for `fence-` is usually helpful. Ensure the packages providing the fence agents you require are installed on every cluster node.

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 *Special Instance Attributes for Fencing Resources* 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 *Special Instance Attributes for Fencing Resources* for details.
6. If the device does not expect the target to be specified with the `port` parameter, you may also need to set the special `pcmk_host_argument` parameter. See *Special Instance Attributes for Fencing Resources* for details.
7. Upload it into the CIB using cibadmin:

```
# cibadmin --create --scope resources --xml-file stonith.xml
```

8. Set `stonith-enabled` to true:

```
# crm_attribute --type crm_config --name stonith-enabled --update true
```

9. Once the stonith resource is running, you can test it by executing the following, replacing `$NODE_NAME` with the name of the node to fence (although you might want to stop the cluster on that machine first):

```
# stonith_admin --reboot $NODE_NAME
```

### Example Fencing Configuration

For this example, we assume we have a cluster node, `pcmk-1`, whose IPMI controller is reachable at the IP address 192.0.2.1. The IPMI controller uses the username `testuser` and the password `abc123`.

1. Looking at what's installed, we may see a variety of available agents:

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

```
(... some output omitted ...)
fence_idrac
fence_ilo3
fence_ilo4
fence_ilo5
fence_imm
```

(continues on next page)

(continued from previous page)

```
fence_ipmilan
(... some output omitted ...)
```

Perhaps after some reading some man pages and doing some Internet searches, we might decide `fence_ipmilan` is our best choice.

2. Next, we would check what parameters `fence_ipmilan` provides:

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

```
<resource-agent name="fence_ipmilan" shortdesc="Fence agent for IPMI">
  <symlink name="fence_ilo3" shortdesc="Fence agent for HP iLO3"/>
  <symlink name="fence_ilo4" shortdesc="Fence agent for HP iLO4"/>
  <symlink name="fence_ilo5" shortdesc="Fence agent for HP iLO5"/>
  <symlink name="fence_imm" shortdesc="Fence agent for IBM Integrated Management Module"/>
  <symlink name="fence_idrac" shortdesc="Fence agent for Dell iDRAC"/>
  <longdesc>fence_ipmilan is an I/O Fencing agentwhich can be used with machines controlled␣
↪by IPMI.This agent calls support software ipmitool (http://ipmitool.sf.net/). WARNING! This␣
↪fence agent might report success before the node is powered off. You should use -m/method␣
↪onoff if your fence device works correctly with that option.</longdesc>
  <vendor-url/>
  <parameters>
    <parameter name="action" unique="0" required="0">
      <getopt mixed="-o, --action=[action]"/>
      <content type="string" default="reboot"/>
      <shortdesc lang="en">Fencing action</shortdesc>
    </parameter>
    <parameter name="auth" unique="0" required="0">
      <getopt mixed="-A, --auth=[auth]"/>
      <content type="select">
        <option value="md5"/>
        <option value="password"/>
        <option value="none"/>
      </content>
      <shortdesc lang="en">IPMI Lan Auth type.</shortdesc>
    </parameter>
    <parameter name="cipher" unique="0" required="0">
      <getopt mixed="-C, --cipher=[cipher]"/>
      <content type="string"/>
      <shortdesc lang="en">Ciphersuite to use (same as ipmitool -C parameter)</shortdesc>
    </parameter>
    <parameter name="hexadecimal_kg" unique="0" required="0">
      <getopt mixed="--hexadecimal-kg=[key]"/>
      <content type="string"/>
      <shortdesc lang="en">Hexadecimal-encoded Kg key for IPMIv2 authentication</shortdesc>
    </parameter>
    <parameter name="ip" unique="0" required="0" obsoletes="ipaddr">
      <getopt mixed="-a, --ip=[ip]"/>
      <content type="string"/>
      <shortdesc lang="en">IP address or hostname of fencing device</shortdesc>
    </parameter>
    <parameter name="ipaddr" unique="0" required="0" deprecated="1">
      <getopt mixed="-a, --ip=[ip]"/>
      <content type="string"/>
      <shortdesc lang="en">IP address or hostname of fencing device</shortdesc>
    </parameter>
```

(continues on next page)

(continued from previous page)

```
    <parameter name="ipport" unique="0" required="0">
      <getopt mixed="-u, --ipport=[port]"/>
      <content type="integer" default="623"/>
      <shortdesc lang="en">TCP/UDP port to use for connection with device</shortdesc>
    </parameter>
    <parameter name="lanplus" unique="0" required="0">
      <getopt mixed="-P, --lanplus"/>
      <content type="boolean" default="0"/>
      <shortdesc lang="en">Use Lanplus to improve security of connection</shortdesc>
    </parameter>
    <parameter name="login" unique="0" required="0" deprecated="1">
      <getopt mixed="-l, --username=[name]"/>
      <content type="string"/>
      <shortdesc lang="en">Login name</shortdesc>
    </parameter>
    <parameter name="method" unique="0" required="0">
      <getopt mixed="-m, --method=[method]"/>
      <content type="select" default="onoff">
        <option value="onoff"/>
        <option value="cycle"/>
      </content>
      <shortdesc lang="en">Method to fence</shortdesc>
    </parameter>
    <parameter name="passwd" unique="0" required="0" deprecated="1">
      <getopt mixed="-p, --password=[password]"/>
      <content type="string"/>
      <shortdesc lang="en">Login password or passphrase</shortdesc>
    </parameter>
    <parameter name="passwd_script" unique="0" required="0" deprecated="1">
      <getopt mixed="-S, --password-script=[script]"/>
      <content type="string"/>
      <shortdesc lang="en">Script to run to retrieve password</shortdesc>
    </parameter>
    <parameter name="password" unique="0" required="0" obsoletes="passwd">
      <getopt mixed="-p, --password=[password]"/>
      <content type="string"/>
      <shortdesc lang="en">Login password or passphrase</shortdesc>
    </parameter>
    <parameter name="password_script" unique="0" required="0" obsoletes="passwd_script">
      <getopt mixed="-S, --password-script=[script]"/>
      <content type="string"/>
      <shortdesc lang="en">Script to run to retrieve password</shortdesc>
    </parameter>
    <parameter name="plug" unique="0" required="0" obsoletes="port">
      <getopt mixed="-n, --plug=[ip]"/>
      <content type="string"/>
      <shortdesc lang="en">IP address or hostname of fencing device (together with --port-as-
↪ip)</shortdesc>
    </parameter>
    <parameter name="port" unique="0" required="0" deprecated="1">
      <getopt mixed="-n, --plug=[ip]"/>
      <content type="string"/>
      <shortdesc lang="en">IP address or hostname of fencing device (together with --port-as-
↪ip)</shortdesc>
    </parameter>
    <parameter name="privlvl" unique="0" required="0">
```

(continues on next page)

(continued from previous page)

```
      <getopt mixed="-L, --privlvl=[level]"/>
      <content type="select" default="administrator">
        <option value="callback"/>
        <option value="user"/>
        <option value="operator"/>
        <option value="administrator"/>
      </content>
      <shortdesc lang="en">Privilege level on IPMI device</shortdesc>
    </parameter>
    <parameter name="target" unique="0" required="0">
      <getopt mixed="--target=[targetaddress]"/>
      <content type="string"/>
      <shortdesc lang="en">Bridge IPMI requests to the remote target address</shortdesc>
    </parameter>
    <parameter name="username" unique="0" required="0" obsoletes="login">
      <getopt mixed="-l, --username=[name]"/>
      <content type="string"/>
      <shortdesc lang="en">Login name</shortdesc>
    </parameter>
    <parameter name="quiet" unique="0" required="0">
      <getopt mixed="-q, --quiet"/>
      <content type="boolean"/>
      <shortdesc lang="en">Disable logging to stderr. Does not affect --verbose or --debug-
↪file or logging to syslog.</shortdesc>
    </parameter>
    <parameter name="verbose" unique="0" required="0">
      <getopt mixed="-v, --verbose"/>
      <content type="boolean"/>
      <shortdesc lang="en">Verbose mode</shortdesc>
    </parameter>
    <parameter name="debug" unique="0" required="0" deprecated="1">
      <getopt mixed="-D, --debug-file=[debugfile]"/>
      <content type="string"/>
      <shortdesc lang="en">Write debug information to given file</shortdesc>
    </parameter>
    <parameter name="debug_file" unique="0" required="0" obsoletes="debug">
      <getopt mixed="-D, --debug-file=[debugfile]"/>
      <content type="string"/>
      <shortdesc lang="en">Write debug information to given file</shortdesc>
    </parameter>
    <parameter name="version" unique="0" required="0">
      <getopt mixed="-V, --version"/>
      <content type="boolean"/>
      <shortdesc lang="en">Display version information and exit</shortdesc>
    </parameter>
    <parameter name="help" unique="0" required="0">
      <getopt mixed="-h, --help"/>
      <content type="boolean"/>
      <shortdesc lang="en">Display help and exit</shortdesc>
    </parameter>
    <parameter name="delay" unique="0" required="0">
      <getopt mixed="--delay=[seconds]"/>
      <content type="second" default="0"/>
      <shortdesc lang="en">Wait X seconds before fencing is started</shortdesc>
    </parameter>
    <parameter name="ipmitool_path" unique="0" required="0">
```

(continues on next page)

(continued from previous page)

```
      <getopt mixed="--ipmitool-path=[path]"/>
      <content type="string" default="/usr/bin/ipmitool"/>
      <shortdesc lang="en">Path to ipmitool binary</shortdesc>
    </parameter>
    <parameter name="login_timeout" unique="0" required="0">
      <getopt mixed="--login-timeout=[seconds]"/>
      <content type="second" default="5"/>
      <shortdesc lang="en">Wait X seconds for cmd prompt after login</shortdesc>
    </parameter>
    <parameter name="port_as_ip" unique="0" required="0">
      <getopt mixed="--port-as-ip"/>
      <content type="boolean"/>
      <shortdesc lang="en">Make "port/plug" to be an alias to IP address</shortdesc>
    </parameter>
    <parameter name="power_timeout" unique="0" required="0">
      <getopt mixed="--power-timeout=[seconds]"/>
      <content type="second" default="20"/>
      <shortdesc lang="en">Test X seconds for status change after ON/OFF</shortdesc>
    </parameter>
    <parameter name="power_wait" unique="0" required="0">
      <getopt mixed="--power-wait=[seconds]"/>
      <content type="second" default="2"/>
      <shortdesc lang="en">Wait X seconds after issuing ON/OFF</shortdesc>
    </parameter>
    <parameter name="shell_timeout" unique="0" required="0">
      <getopt mixed="--shell-timeout=[seconds]"/>
      <content type="second" default="3"/>
      <shortdesc lang="en">Wait X seconds for cmd prompt after issuing command</shortdesc>
    </parameter>
    <parameter name="retry_on" unique="0" required="0">
      <getopt mixed="--retry-on=[attempts]"/>
      <content type="integer" default="1"/>
      <shortdesc lang="en">Count of attempts to retry power on</shortdesc>
    </parameter>
    <parameter name="sudo" unique="0" required="0" deprecated="1">
      <getopt mixed="--use-sudo"/>
      <content type="boolean"/>
      <shortdesc lang="en">Use sudo (without password) when calling 3rd party software</
↪shortdesc>
    </parameter>
    <parameter name="use_sudo" unique="0" required="0" obsoletes="sudo">
      <getopt mixed="--use-sudo"/>
      <content type="boolean"/>
      <shortdesc lang="en">Use sudo (without password) when calling 3rd party software</
↪shortdesc>
    </parameter>
    <parameter name="sudo_path" unique="0" required="0">
      <getopt mixed="--sudo-path=[path]"/>
      <content type="string" default="/usr/bin/sudo"/>
      <shortdesc lang="en">Path to sudo binary</shortdesc>
    </parameter>
  </parameters>
  <actions>
    <action name="on" automatic="0"/>
    <action name="off"/>
    <action name="reboot"/>
```

(continues on next page)

(continued from previous page)

```
    <action name="status"/>
    <action name="monitor"/>
    <action name="metadata"/>
    <action name="manpage"/>
    <action name="validate-all"/>
    <action name="diag"/>
    <action name="stop" timeout="20s"/>
    <action name="start" timeout="20s"/>
  </actions>
</resource-agent>
```

Once we've decided what parameter values we think we need, it is a good idea to run the fence agent's status action manually, to verify that our values work correctly:

```
# fence_ipmilan --lanplus -a 192.0.2.1 -l testuser -p abc123 -o status

Chassis Power is on
```

3. Based on that, we might create a fencing resource configuration like this in `stonith.xml` (or any file name, just use the same name with `cibadmin` later):

```
<primitive id="Fencing-pcmk-1" class="stonith" type="fence_ipmilan" >
  <instance_attributes id="Fencing-params" >
    <nvpair id="Fencing-lanplus" name="lanplus" value="1" />
    <nvpair id="Fencing-ip" name="ip" value="192.0.2.1" />
    <nvpair id="Fencing-password" name="password" value="testuser" />
    <nvpair id="Fencing-username" name="username" value="abc123" />
  </instance_attributes>
  <operations >
    <op id="Fencing-monitor-10m" interval="10m" name="monitor" timeout="300s" />
  </operations>
</primitive>
```

**Note:** Even though the man page shows that the `action` parameter is supported, we do not provide that in the resource configuration. Pacemaker will supply an appropriate action whenever the fence device must be used.

4. In this case, we don't need to configure `pcmk_host_map` because `fence_ipmilan` ignores the target node name and instead uses its `ip` parameter to know how to contact the IPMI controller.

5. We do need to let Pacemaker know which cluster node can be fenced by this device, since `fence_ipmilan` doesn't support the `list` action. Add a line like this to the agent's instance attributes:

```
<nvpair id="Fencing-pcmk_host_list" name="pcmk_host_list" value="pcmk-1" />
```

6. We don't need to configure `pcmk_host_argument` since `ip` is all the fence agent needs (it ignores the target name).

7. Make the configuration active:

```
# cibadmin --create --scope resources --xml-file stonith.xml
```

8. Set `stonith-enabled` to true (this only has to be done once):

```
# crm_attribute --type crm_config --name stonith-enabled --update true
```

9. Since our cluster is still in testing, we can reboot `pcmk-1` without bothering anyone, so we'll test our fencing configuration by running this from one of the other cluster nodes:

```
# stonith_admin --reboot pcmk-1
```

   Then we will verify that the node did, in fact, reboot.

We can repeat that process to create a separate fencing resource for each node.

With some other fence device types, a single fencing resource is able to be used for all nodes. In fact, we could do that with `fence_ipmilan`, using the `port-as-ip` parameter along with `pcmk_host_map`. Either approach is fine.

### 2.8.15 Fencing Topologies

Pacemaker supports fencing nodes with multiple devices through a feature called *fencing topologies*. Fencing topologies may be used to provide alternative devices in case one fails, or to require multiple devices to all be executed successfully in order to consider the node successfully fenced, or even a combination of the two.

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 values are 1 through 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 scheduler and/or the controller.

Some possible uses of topologies include:

- Try on-board IPMI, then an intelligent power switch if that fails
- Try fabric fencing of both disk and network, then fall back to power fencing if either fails
- Wait up to a certain time for a kernel dump to complete, then cut power to the node

Table 17: **Attributes of a fencing-level Element**

| Attribute | Description |
|---|---|
| id | A unique name for this element (required) |
| target | The name of a single node to which this level applies |
| target-pattern | An extended regular expression (as defined in POSIX) matching the names of nodes to which this level applies |
| target-attribute | The name of a node attribute that is set (to `target-value`) for nodes to which this level applies |
| target-value | The node attribute value (of `target-attribute`) that is set for nodes to which this level applies |
| index | The order in which to attempt the levels. Levels are attempted in ascending order *until one succeeds*. Valid values are 1 through 9. |
| devices | A comma-separated list of devices that must all be tried for this level |

**Note: Fencing topology with different devices for different nodes**

```
<cib crm_feature_set="3.6.0" validate-with="pacemaker-3.5" 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>
```

### Example Dual-Layer, Dual-Device Fencing Topologies

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

- 2 nodes (prod-mysql1 and prod-mysql2)
- the nodes have IPMI controllers reachable at 192.0.2.1 and 192.0.2.2
- the nodes each have two independent Power Supply Units (PSUs) connected to two independent Power Distribution Units (PDUs) reachable at 198.51.100.1 (port 10 and port 11) and 203.0.113.1 (port 10 and port 11)
- fencing via the IPMI controller uses the `fence_ipmilan` agent (1 fence device per controller, with each device targeting a separate node)
- fencing via the PDUs uses the `fence_apc_snmp` agent (1 fence device per PDU, with both devices targeting both nodes)
- a random delay is used to lessen the chance of a "death match"
- fencing topology is set to try IPMI fencing first then dual PDU fencing if that fails

In a node failure scenario, Pacemaker will first select `fence_ipmilan` to try to kill the faulty node. Using the fencing topology, if that method fails, it will then move on to selecting `fence_apc_snmp` twice (once for the first PDU, then again for the second PDU).

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

**Note: First fencing method: single IPMI device per target**

Each cluster node has it own dedicated IPMI controller that can be contacted 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-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-lanplus" name="lanplus" value="true"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-pcmk_host_list" name="pcmk_host_list"␣
↪value="prod-mysql1"/>
    <nvpair id="fence_prod-mysql1_ipmi-instance_attributes-pcmk_delay_max" name="pcmk_delay_max"␣
↪value="8s"/>
  </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-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-lanplus" name="lanplus" value="true"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-pcmk_host_list" name="pcmk_host_list"␣
↪value="prod-mysql2"/>
    <nvpair id="fence_prod-mysql2_ipmi-instance_attributes-pcmk_delay_max" name="pcmk_delay_max"␣
↪value="8s"/>
  </instance_attributes>
</primitive>
```

---

**Note: Second fencing method: dual PDU devices**

Each cluster node also has 2 distinct power supplies controlled by 2 distinct PDUs:

- Node 1: PDU 1 port 10 and PDU 2 port 10
- Node 2: PDU 1 port 11 and PDU 2 port 11

The matching fencing agents are configured as follows:

```
<primitive class="stonith" id="fence_apc1" type="fence_apc_snmp">
  <instance_attributes id="fence_apc1-instance_attributes">
    <nvpair id="fence_apc1-instance_attributes-ipaddr" name="ipaddr" value="198.51.100.1"/>
    <nvpair id="fence_apc1-instance_attributes-login" name="login" value="fencing"/>
    <nvpair id="fence_apc1-instance_attributes-passwd" name="passwd" value="fencing"/>
    <nvpair id="fence_apc1-instance_attributes-pcmk_host_list"
       name="pcmk_host_map" value="prod-mysql1:10;prod-mysql2:11"/>
    <nvpair id="fence_apc1-instance_attributes-pcmk_delay_max" name="pcmk_delay_max" value="8s"/>
  </instance_attributes>
</primitive>
<primitive class="stonith" id="fence_apc2" type="fence_apc_snmp">
  <instance_attributes id="fence_apc2-instance_attributes">
    <nvpair id="fence_apc2-instance_attributes-ipaddr" name="ipaddr" value="203.0.113.1"/>
    <nvpair id="fence_apc2-instance_attributes-login" name="login" value="fencing"/>
    <nvpair id="fence_apc2-instance_attributes-passwd" name="passwd" value="fencing"/>
    <nvpair id="fence_apc2-instance_attributes-pcmk_host_list"
       name="pcmk_host_map" value="prod-mysql1:10;prod-mysql2:11"/>
    <nvpair id="fence_apc2-instance_attributes-pcmk_delay_max" name="pcmk_delay_max" value="8s"/>
  </instance_attributes>
</primitive>
```

**Note: 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 id="level-1-1" target="prod-mysql1" index="1" devices="fence_prod-mysql1_ipmi" />
  <fencing-level id="level-1-2" target="prod-mysql1" index="2" devices="fence_apc1,fence_apc2"  />
  <fencing-level id="level-2-1" target="prod-mysql2" index="1" devices="fence_prod-mysql2_ipmi" />
  <fencing-level id="level-2-2" target="prod-mysql2" index="2" devices="fence_apc1,fence_apc2"  />
</fencing-topology>
```

In `fencing-topology`, the lowest `index` value for a target determines its first fencing method.

### 2.8.16 Remapping Reboots

When the cluster needs to reboot a node, whether because `stonith-action` is `reboot` or because a reboot was requested externally (such as by `stonith_admin --reboot`), it will remap that to other commands in two cases:

- If the chosen fencing device does not support the `reboot` command, the cluster will ask it to perform `off` instead.
- 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`).

## 2.9 Alerts

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

### 2.9.1 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.).

**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.

For more information about sample alert agents provided by Pacemaker and about developing custom alert agents, see the *Pacemaker Administration* document.

### 2.9.2 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.

**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.

### 2.9.3 Alert Meta-Attributes

As with resources, meta-attributes can be configured for alerts to change whether and how Pacemaker calls them.

Table 18: **Meta-Attributes of an Alert**

| Meta-Attribute | Default | Description |
|---|---|---|
| enabled | true | If false for an alert, the alert will not be used. If true for an alert and false for a particular recipient of that alert, that recipient will not be used. *(since 2.1.6)* |

Continued on next page

Table 18 – continued from previous page

| 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 and/or per recipient.

**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`.

### 2.9.4 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.

**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>
```

### 2.9.5 Alert Filters

By default, an alert agent will be called for node events, fencing events, and resource events. An agent may choose to ignore certain types of events, but there is still the overhead of calling it for those events. To eliminate that overhead, you may select which types of events the agent should receive.

Alert filters are configured within a `select` element inside an `alert` element.

Table 19: **Possible alert filters**

| Name | Events alerted |
|---|---|
| select_nodes | A node joins or leaves the cluster (whether at the cluster layer for cluster nodes, or via a remote connection for Pacemaker Remote nodes). |
| select_fencing | Fencing or unfencing of a node completes (whether successfully or not). |
| select_resources | A resource action other than meta-data completes (whether successfully or not). |
| select_attributes | A transient attribute value update is sent to the CIB. |

**Alert configuration to receive only node events and fencing events**

```
<configuration>
   <alerts>
      <alert id="my-alert" path="/path/to/my-script.sh">
         <select>
            <select_nodes />
            <select_fencing />
         </select>
         <recipient id="my-alert-recipient1"
                    value="someuser@example.com"/>
      </alert>
   </alerts>
</configuration>
```

With `<select_attributes>` (the only event type not enabled by default), the agent will receive alerts when a node attribute changes. If you wish the agent to be called only when certain attributes change, you can configure that as well.

**Alert configuration to be called when certain node attributes change**

```
<configuration>
   <alerts>
      <alert id="my-alert" path="/path/to/my-script.sh">
         <select>
            <select_attributes>
               <attribute id="alert-standby" name="standby" />
               <attribute id="alert-shutdown" name="shutdown" />
            </select_attributes>
         </select>
         <recipient id="my-alert-recipient1" value="someuser@example.com"/>
      </alert>
   </alerts>
</configuration>
```

Node attribute alerts are currently considered experimental. Alerts may be limited to attributes set via `attrd_updater`, and agents may be called multiple times with the same attribute value.

## 2.10 Rules

Rules make a configuration more dynamic, allowing values to depend on conditions such as time of day or the value of a node attribute. For example, rules can:

- Set a higher value for *resource-stickiness* during working hours to minimize downtime, and a lower value on weekends to allow resources to move to their most preferred locations when people aren't around
- Automatically place the cluster into maintenance mode during a scheduled maintenance window
- Restrict a particular department's resources to run on certain nodes, as determined by custom resource meta-attributes and node attributes

### 2.10.1 Rule Options

Each context that supports rules may contain a single `rule` element.

Table 20: **Attributes of a rule Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| boolean-op | *enumeration* | `and` | How to combine conditions if this rule contains more than one. Allowed values:<br>• `and`: the rule is satisfied only if all conditions are satisfied<br>• `or`: the rule is satisfied if any condition is satisfied |

### 2.10.2 Rule Conditions and Contexts

A `rule` element must contain one or more conditions. A condition is any of the following, which will be described in more detail later:

- a *date/time expression*
- a *node attribute expression*
- a *resource type expression*
- an *operation type expression*
- another `rule` (allowing for complex combinations of conditions)

Each type of condition is allowed only in certain contexts. Although any given context may contain only one `rule` element, that element may contain any number of conditions, including other `rule` elements.

Rules may be used in the following contexts, which also will be described in more detail later:

- a *location constraint*
- a *cluster_property_set* element (within the `crm_config` element)
- an *instance_attributes* element (within an `alert`, `bundle`, `clone`, `group`, `node`, `op`, `primitive`, `recipient`, or `template` element)
- a *meta_attributes* element (within an `alert`, `bundle`, `clone`, `group`, `op`, `op_defaults`, `primitive`, `recipient`, `rsc_defaults`, or `template` element)
- a *utilization* element (within a `node`, `primitive`, or `template` element)

### 2.10.3 Date/Time Expressions

The `date_expression` element configures a rule condition based on the current date and time. It is allowed in rules in any context.

It may contain a `date_spec` or `duration` element depending on the `operation` as described below.

Table 21: **Attributes of a date_expression Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| start | *ISO 8601* | | The beginning of the desired time range. Meaningful with an `operation` of `in_range` or `gt`. |
| end | *ISO 8601* | | The end of the desired time range. Meaningful with an `operation` of `in_range` or `lt`. |

Continued on next page

Table 21 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| operation | *enumeration* | `in_range` | Specifies how to compare the current date/time against a desired time range. Allowed values:<br>• `gt:` The expression is satisfied if the current date/time is after `start` (which is required)<br>• `lt:` The expression is satisfied if the current date/time is before `end` (which is required)<br>• `in_range:` The expression is satisfied if the current date/time is greater than or equal to `start` (if specified) and less than or equal to either `end` (if specified) or `start` plus the value of the *duration* element (if one is contained in the `date_expression`). At least one of `start` or `end` must be specified. If both `end` and `duration` are specified, `duration` is ignored.<br>• `date_spec:` The expression is satisfied if the current date/time matches the specification given in the contained *date_spec* element (which is required) |

### Date Specifications

A `date_spec` element is used within a `date_expression` to specify a combination of dates and times that satisfy the expression.

Table 22: **Attributes of a date_spec Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| seconds | *range* | | If this is set, the expression is satisfied only if the current time's second is within this range. Allowed integers: 0 to 59. |
| minutes | *range* | | If this is set, the expression is satisfied only if the current time's minute is within this range. Allowed integers: 0 to 59. |
| hours | *range* | | If this is set, the expression is satisfied only if the current time's hour is within this range. Allowed integers: 0 to 23 where 0 is midnight and 23 is 11 p.m. |
| monthdays | *range* | | If this is set, the expression is satisfied only if the current date's day of the month is in this range. Allowed integers: 1 to 31. |
| weekdays | *range* | | If this is set, the expression is satisfied only if the current date's ordinal day of the week is in this range. Allowed integers: 1-7 (where 1 is Monday and 7 is Sunday). |
| yeardays | *range* | | If this is set, the expression is satisfied only if the current date's ordinal day of the year is in this range. Allowed integers: 1-366. |
| months | *range* | | If this is set, the expression is satisfied only if the current date's month is in this range. Allowed integers: 1-12 where 1 is January and 12 is December. |

Continued on next page

Table 22 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| weeks | *range* | | If this is set, the expression is satisfied only if the current date's ordinal week of the year is in this range. Allowed integers: 1-53. |
| years | *range* | | If this is set, the expression is satisfied only if the current date's year according to the Gregorian calendar is in this range. |
| weekyears | *range* | | If this is set, the expression is satisfied only if the current date's year in which the week started (according to the ISO 8601 standard) is in this range. |
| moon | *range* | | If this is set, the expression is satisfied only if the current date's phase of the moon is in this range. Allowed values are 0 to 7 where 0 is the new moon and 4 is the full moon. *(deprecated since 2.1.6)* |

**Note:** Pacemaker can calculate when evaluation of a `date_expression` with an `operation` of `gt`, `lt`, or `in_range` will next change, and schedule a cluster re-check for that time. However, it does not do this for `date_spec`. Instead, it evaluates the `date_spec` whenever a cluster re-check naturally happens via a cluster event or the `cluster-recheck-interval` cluster option.

For example, if you have a `date_spec` enabling a resource from 9 a.m. to 5 p.m., and `cluster-recheck-interval` has been set to 5 minutes, then sometime between 9 a.m. and 9:05 a.m. the cluster would notice that it needs to start the resource, and sometime between 5 p.m. and 5:05 p.m. it would realize that it needs to stop the resource. The timing of the actual start and stop actions will further depend on factors such as any other actions the cluster may need to perform first, and the load of the machine.

### Durations

A `duration` element is used within a `date_expression` to calculate an ending value for `in_range` operations when `end` is not supplied.

Table 23: **Attributes of a duration Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| seconds | *integer* | 0 | Number of seconds to add to the total duration |
| minutes | *integer* | 0 | Number of minutes to add to the total duration |
| hours | *integer* | 0 | Number of hours to add to the total duration |
| days | *integer* | 0 | Number of days to add to the total duration |
| weeks | *integer* | 0 | Number of weeks to add to the total duration |
| months | *integer* | 0 | Number of months to add to the total duration |
| years | *integer* | 0 | Number of years to add to the total duration |

### Example Date/Time Expressions

**Satisfied if the current year is 2005**

```
<rule id="rule1" score="INFINITY">
   <date_expression id="date_expr1" start="2005-001" operation="in_range">
    <duration id="duration1" years="1"/>
   </date_expression>
</rule>
```

or equivalently:

```
<rule id="rule2" score="INFINITY">
   <date_expression id="date_expr2" operation="date_spec">
    <date_spec id="date_spec2" years="2005"/>
   </date_expression>
</rule>
```

**9 a.m. to 5 p.m. Monday through Friday**

```
<rule id="rule3" score="INFINITY">
   <date_expression id="date_expr3" operation="date_spec">
    <date_spec id="date_spec3" hours="9-16" weekdays="1-5"/>
   </date_expression>
</rule>
```

Note that the `16` matches all the way through `16:59:59`, because the numeric value of the hour still matches.

**9 a.m. to 6 p.m. Monday through Friday, or anytime Saturday**

```
<rule id="rule4" score="INFINITY" boolean-op="or">
   <date_expression id="date_expr4-1" operation="date_spec">
    <date_spec id="date_spec4-1" hours="9-16" weekdays="1-5"/>
   </date_expression>
   <date_expression id="date_expr4-2" operation="date_spec">
    <date_spec id="date_spec4-2" weekdays="6"/>
   </date_expression>
</rule>
```

**9 a.m. to 5 p.m. or 9 p.m. to 12 a.m. Monday through Friday**

```
<rule id="rule5" score="INFINITY" boolean-op="and">
   <rule id="rule5-nested1" score="INFINITY" boolean-op="or">
    <date_expression id="date_expr5-1" operation="date_spec">
     <date_spec id="date_spec5-1" hours="9-16"/>
    </date_expression>
    <date_expression id="date_expr5-2" operation="date_spec">
     <date_spec id="date_spec5-2" hours="21-23"/>
    </date_expression>
   </rule>
   <date_expression id="date_expr5-3" operation="date_spec">
    <date_spec id="date_spec5-3" weekdays="1-5"/>
   </date_expression>
</rule>
```

**Mondays in March 2005**

```
<rule id="rule6" score="INFINITY" boolean-op="and">
   <date_expression id="date_expr6-1" operation="date_spec">
    <date_spec id="date_spec6" 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 only the first second of 2005-04-01. You may wish to write `end` as `"2005-03-31T23:59:59"` to avoid confusion.

### 2.10.4 Node Attribute Expressions

The `expression` element configures a rule condition based on the value of a node attribute. It is allowed in rules in location constraints and in `instance_attributes` elements within `bundle`, `clone`, `group`, `op`, `primitive`, and `template` elements.

Table 24: **Attributes of an expression Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| attribute | *text* | | Name of the node attribute to test (required) |

Continued on next page

Table 24 – continued from previous page

| Name | Type | Default | Description |
|---|---|---|---|
| operation | *enumeration* | | The comparison to perform (required). Allowed values:<br>• `defined:` The expression is satisfied if the node has the named attribute<br>• `not_defined:` The expression is satisfied if the node does not have the named attribute<br>• `lt:` The expression is satisfied if the node attribute value is less than the reference value<br>• `gt:` The expression is satisfied if the node attribute value is greater than the reference value<br>• `lte:` The expression is satisfied if the node attribute value is less than or equal to the reference value<br>• `gte:` The expression is satisfied if the node attribute value is greater than or equal to the reference value<br>• `eq:` The expression is satisfied if the node attribute value is equal to the reference value<br>• `ne:` The expression is satisfied if the node attribute value is not equal to the reference value |
| type | *enumeration* | The default type for `lt`, `gt`, `lte`, and `gte` operations is `number` if either value contains a decimal point character, or `integer` otherwise. The default type for all other operations is `string`. If a numeric parse fails for either value, then the values are compared as type `string`. | How to interpret values. Allowed values are `string`, `integer` *(since 2.0.5)*, `number`, and `version`. `integer` truncates floating-point values if necessary before performing a 64-bit integer comparison. `number` performs a double-precision floating-point comparison *(32-bit integer before 2.0.5)*. |
| value | *text* | | Reference value to compare node attribute against (used only with, and required for, operations other than `defined` and `not_defined`) |
| value-source | *enumeration* | `literal` | How the reference value is obtained. Allowed values:<br>• `literal`: `value` contains the literal reference value to compare<br>• `param`: `value` contains the name of a resource parameter to compare (valid only in the context of a location constraint)<br>• `meta`: `value` is the name of a resource meta-attribute to compare (valid only in the context of a location constraint) |

In addition to custom node attributes defined by the administrator, the cluster defines special, built-in node attributes for each node that can also be used in rule expressions.

Table 25: **Built-in Node Attributes**

| Name | Description |
|---|---|
| #uname | *Node name* |
| #id | Node ID |
| #kind | Node type (`cluster` for cluster nodes, `remote` for Pacemaker Remote nodes created with the `ocf:pacemaker:remote` resource, and `container` for Pacemaker Remote guest nodes and bundle nodes) |
| #is_dc | `true` if this node is the cluster's Designated Controller (DC), `false` otherwise |
| #cluster-name | The value of the `cluster-name` cluster property, if set |
| #site-name | The value of the `site-name` node attribute, if set, otherwise identical to `#cluster-name` |

### 2.10.5 Resource Type Expressions

The `rsc_expression` element *(since 2.0.5)* configures a rule condition based on the agent used for a resource. It is allowed in rules in a `meta_attributes` element within a `rsc_defaults` or `op_defaults` element.

Table 26: **Attributes of a rsc_expression Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| class | *text* | | If this is set, the expression is satisfied only if the resource's agent standard matches this value |
| provider | *text* | | If this is set, the expression is satisfied only if the resource's agent provider matches this value |
| type | *text* | | If this is set, the expression is satisfied only if the resource's agent type matches this value |

#### Example Resource Type Expressions

**Satisfied for `ocf:heartbeat:IPaddr2` resources**

```
<rule id="rule1" score="INFINITY">
    <rsc_expression id="rule_expr1" class="ocf" provider="heartbeat" type="IPaddr2"/>
</rule>
```

**Satisfied for `stonith:fence_xvm` resources**

```
<rule id="rule2" score="INFINITY">
    <rsc_expression id="rule_expr2" class="stonith" type="fence_xvm"/>
</rule>
```

### 2.10.6 Operation Type Expressions

The `op_expression` element *(since 2.0.5)* configures a rule condition based on a resource operation name and interval. It is allowed in rules in a `meta_attributes` element within an `op_defaults` element.

Table 27: **Attributes of an op_expression Element**

| Name | Type | Default | Description |
|---|---|---|---|
| id | *id* | | A unique name for this element (required) |
| name | *text* | | The expression is satisfied only if the operation's name matches this value (required) |
| interval | *duration* | | If this is set, the expression is satisfied only if the operation's interval matches this value |

### Example Operation Type Expressions

**Expression is satisfied for all monitor actions**

```
<rule id="rule1" score="INFINITY">
    <op_expression id="rule_expr1" name="monitor"/>
</rule>
```

**Expression is satisfied for all monitor actions with a 10-second interval**

```
<rule id="rule2" score="INFINITY">
    <op_expression id="rule_expr2" name="monitor" interval="10s"/>
</rule>
```

## 2.10.7 Using Rules to Determine Resource Location

If a *location constraint* contains a rule, the cluster will apply the constraint to all nodes where the rule is satisfied. This acts as if identical location constraints without rules were defined for each of the nodes.

In the context of a location constraint, `rule` elements may take additional attributes. These have an effect only when set for the constraint's top-level `rule`; they are ignored if set on a subrule.

Table 28: **Extra Attributes of a rule Element in a Location Constraint**

| Name | Type | Default | Description |
|---|---|---|---|
| role | *enumeration* | `Started` | If this is set in the constraint's top-level rule, the constraint acts as if `role` were set to this in the `rsc_location` element. |
| score | *score* | | If this is set in the constraint's top-level rule, the constraint acts as if `score` were set to this in the `rsc_location` element. Only one of `score` and `score-attribute` may be set. |
| score-attribute | *text* | | If this is set in the constraint's top-level rule, the constraint acts as if `score` were set to the value of this node attribute on each node where the rule is satisfied. Only one of `score` and `score-attribute` may be set. |

Consider the following simple location constraint:

**Prevent resource webserver from running on node node3**

```
<rsc_location id="ban-apache-on-node3" rsc="webserver"
              score="-INFINITY" node="node3"/>
```

The same constraint can be written more verbosely using a rule:

**Prevent resource webserver from running on node node3 using a rule**

```
<rsc_location id="ban-apache-on-node3" rsc="webserver">
    <rule id="ban-apache-rule" score="-INFINITY">
      <expression id="ban-apache-expr" attribute="#uname"
        operation="eq" value="node3"/>
    </rule>
</rsc_location>
```

The advantage of using the expanded form is that one could add more expressions (for example, limiting the constraint to certain days of the week).

### Location Rules Based on Other Node Properties

The expanded form allows us to match node attributes other than its name. As an example, consider this configuration of custom node attributes specifying each node's CPU capacity:

**Sample node section with node attributes**

```
<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>
```

We can use a rule to prevent a resource from running on underpowered machines:

**Rule using a node attribute (to be used inside a location constraint)**

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

#### Using `score-attribute` Instead of `score`

When using `score-attribute` instead of `score`, each node matched by the rule has its score adjusted according to its value for the named node attribute.

In the previous example, if the location constraint rule used `score-attribute="cpu_mips"` instead of `score="-INFINITY"`, node `c001n01` would have its preference to run the resource increased by 1234 whereas node `c001n02` would have its preference increased by 5678.

#### Specifying location scores using pattern submatches

Location constraints may use *rsc-pattern* to apply the constraint to all resources whose IDs match the given pattern. The pattern may contain up to 9 submatches in parentheses, whose values may be used as `%1` through `%9` in a `rule` element's `score-attribute` or an `expression` element's `attribute`.

For example, the following configuration excerpt gives the resources **server-httpd** and **ip-httpd** a preference of 100 on node1 and 50 on node2, and **ip-gateway** a preference of -100 on node1 and 200 on node2.

**Location constraint using submatches**

```
<nodes>
   <node id="1" uname="node1">
      <instance_attributes id="node1-attrs">
         <nvpair id="node1-prefer-httpd" name="prefer-httpd" value="100"/>
         <nvpair id="node1-prefer-gateway" name="prefer-gateway" value="-100"/>
      </instance_attributes>
   </node>
   <node id="2" uname="node2">
      <instance_attributes id="node2-attrs">
         <nvpair id="node2-prefer-httpd" name="prefer-httpd" value="50"/>
         <nvpair id="node2-prefer-gateway" name="prefer-gateway" value="200"/>
      </instance_attributes>
   </node>
</nodes>
<resources>
   <primitive id="server-httpd" class="ocf" provider="heartbeat" type="apache"/>
   <primitive id="ip-httpd" class="ocf" provider="heartbeat" type="IPaddr2"/>
   <primitive id="ip-gateway" class="ocf" provider="heartbeat" type="IPaddr2"/>
</resources>
<constraints>
   <!-- The following constraint says that for any resource whose name
        starts with "server-" or "ip-", that resource's preference for a
        node is the value of the node attribute named "prefer-" followed
        by the part of the resource name after "server-" or "ip-",
        wherever such a node attribute is defined.
     -->
   <rsc_location id="location1" rsc-pattern="(server|ip)-(.*)">
      <rule id="location1-rule1" score-attribute="prefer-%2">
         <expression id="location1-rule1-expression1" attribute="prefer-%2" operation="defined"/>
      </rule>
   </rsc_location>
</constraints>
```

### 2.10.8 Using Rules to Define Options

Rules may be used to control a variety of options:

- *Cluster options* (as `cluster_property_set` elements)
- *Node attributes* (as `instance_attributes` or `utilization` elements inside a `node` element)
- *Resource options* (as `utilization`, `meta_attributes`, or `instance_attributes` elements inside a resource definition element or `op` , `rsc_defaults`, `op_defaults`, or `template` element)
- *Operation options* (as `meta_attributes` elements inside an `op` or `op_defaults` element)
- *Alert options* (as `instance_attributes` or `meta_attributes` elements inside an `alert` or `recipient` element)

#### Using Rules to Control Resource Options

Often some cluster nodes will be different from their peers. Sometimes, these differences (for example, 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` elements 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.

**Defining different resource options based on the node name**

```xml
<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>
```

Multiple `instance_attributes` elements are evaluated from highest score to lowest. If not supplied, the score defaults to zero. Objects with equal scores are processed in their listed order. If an `instance_attributes` object has no rule or a satisfied `rule`, then for any parameter the resource does not yet have a value for, the resource will use the value defined by the `instance_attributes`.

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

- `special-node1` has the highest score (3) and so is evaluated first; its rule is satisfied, so `interface` is set to `eth1`.
- `special-node2` is evaluated next with score 2, but its rule is not satisfied, so it is ignored.
- `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`.

### Using Rules to Control Resource Defaults

Rules can be used for resource and operation defaults.

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.

**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>
```

`rsc_expression` is valid within both `rsc_defaults` and `op_defaults`; `op_expression` is valid only within `op_defaults`.

**Default all IPaddr2 resources to stopped**

```
<rsc_defaults>
    <meta_attributes id="op-target-role">
        <rule id="op-target-role-rule" score="INFINITY">
            <rsc_expression id="op-target-role-expr" class="ocf" provider="heartbeat"
              type="IPaddr2"/>
        </rule>
        <nvpair id="op-target-role-nvpair" name="target-role" value="Stopped"/>
    </meta_attributes>
</rsc_defaults>
```

**Default all monitor action timeouts to 7 seconds**

```
<op_defaults>
    <meta_attributes id="op-monitor-defaults">
        <rule id="op-monitor-default-rule" score="INFINITY">
            <op_expression id="op-monitor-default-expr" name="monitor"/>
        </rule>
        <nvpair id="op-monitor-timeout" name="timeout" value="7s"/>
    </meta_attributes>
</op_defaults>
```

**Default the timeout on all 10-second-interval monitor actions on IPaddr2 resources to 8 seconds**

```
<op_defaults>
    <meta_attributes id="op-monitor-and">
        <rule id="op-monitor-and-rule" score="INFINITY">
            <rsc_expression id="op-monitor-and-rsc-expr" class="ocf" provider="heartbeat"
              type="IPaddr2"/>
            <op_expression id="op-monitor-and-op-expr" name="monitor" interval="10s"/>
        </rule>
        <nvpair id="op-monitor-and-timeout" name="timeout" value="8s"/>
    </meta_attributes>
</op_defaults>
```

#### 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 following example illustrates how to set `maintenance_mode` during a scheduled maintenance window. This will keep the cluster running but not monitor, start, or stop resources during this time.

**Schedule a maintenance window for 9 to 11 p.m. CDT Sept. 20, 2019**

```
<crm_config>
   <cluster_property_set id="cib-bootstrap-options">
     <nvpair id="bootstrap-stonith-enabled" name="stonith-enabled" value="1"/>
   </cluster_property_set>
   <cluster_property_set id="normal-set" score="10">
     <nvpair id="normal-maintenance-mode" name="maintenance-mode" value="false"/>
   </cluster_property_set>
   <cluster_property_set id="maintenance-window-set" score="1000">
     <nvpair id="maintenance-nvpair1" name="maintenance-mode" value="true"/>
     <rule id="maintenance-rule1" score="INFINITY">
       <date_expression id="maintenance-date1" operation="in_range"
         start="2019-09-20 21:00:00 -05:00" end="2019-09-20 23:00:00 -05:00"/>
     </rule>
   </cluster_property_set>
</crm_config>
```

**Important:** The `cluster_property_set` with an `id` set to “cib-bootstrap-options” will *always* have the highest priority, regardless of any scores. Therefore, rules in another `cluster_property_set` can never take effect for any properties listed in the bootstrap set.

## 2.11 Collective Resources

Pacemaker supports several types of *collective* resources, which consist of multiple, related resource instances.

### 2.11.1 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.

**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:

**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.

### Group Properties

Table 29: **Properties of a Group Resource**

| Field | Description |
| --- | --- |
| id | A unique name for the group |
| description | An optional description of the group, for the user's own purposes. E.g. `resources needed for website` |

### Group Options

Groups inherit the `priority`, `target-role`, and `is-managed` properties from primitive resources. See *Resource Options* for information about those properties.

### 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.

### Group Contents

Groups may only contain a collection of cluster resources (see *Resource Properties*). To refer to a child of a group resource, just use the child's `id` instead of the group's.

### Group Constraints

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

**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>
```

#### 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.

### 2.11.2 Clones - Resources That Can Have Multiple Active Instances

*Clone* resources are resources that can have more than one copy active at the same time. This allows you, for example, to run a copy of a daemon on every node. You can clone any primitive or group resource[1].

#### Anonymous versus Unique Clones

A clone resource is configured to be either *anonymous* or *globally unique*.

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

The instances of globally unique clones are distinct entities. All instances are launched identically, but one instance of the clone is not identical to any other instance, whether running on the same node or a different node. As an example, a cloned IP address can use special kernel functionality such that each instance handles a subset of requests for the same IP address.

#### Promotable clones

If a clone is *promotable*, its instances can perform a special role that Pacemaker will manage via the `promote` and `demote` actions of the resource agent.

Services that support such a special role have various terms for the special role and the default role: primary and secondary, master and replica, controller and worker, etc. Pacemaker uses the terms *promoted* and *unpromoted* to be agnostic to what the service calls them or what they do.

All that Pacemaker cares about is that an instance comes up in the unpromoted role when started, and the resource agent supports the `promote` and `demote` actions to manage entering and exiting the promoted role.

#### Clone Properties

Table 30: **Properties of a Clone Resource**

| Field | Description |
|---|---|
| id | A unique name for the clone |
| description | An optional description of the clone, for the user's own purposes. E.g. `IP address for website` |

#### Clone Options

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

[1] Of course, the service must support running multiple instances.

Table 31: **Clone-specific configuration options**

| Field | Default | Description |
|---|---|---|
| globally-unique | false | If **true**, each clone instance performs a distinct function |
| clone-max | 0 | The maximum number of clone instances that can be started across the entire cluster. If 0, the number of nodes in the cluster will be used. |
| clone-node-max | 1 | If `globally-unique` is **true**, the maximum number of clone instances that can be started on a single node |
| clone-min | 0 | Require at least this number of clone instances to be runnable before allowing resources depending on the clone to be runnable. A value of 0 means require all clone instances to be runnable. |
| notify | false | Call the resource agent's **notify** action for all active instances, before and after starting or stopping any clone instance. The resource agent must support this action. Allowed values: **false**, **true** |
| ordered | false | If **true**, clone instances must be started sequentially instead of in parallel. Allowed values: **false**, **true** |
| interleave | false | When this clone is ordered relative to another clone, if this option is **false** (the default), the ordering is relative to *all* instances of the other clone, whereas if this option is **true**, the ordering is relative only to instances on the same node. Allowed values: **false**, **true** |
| promotable | false | If **true**, clone instances can perform a special role that Pacemaker will manage via the resource agent's **promote** and **demote** actions. The resource agent must support these actions. Allowed values: **false**, **true** |
| promoted-max | 1 | If `promotable` is **true**, the number of instances that can be promoted at one time across the entire cluster |
| promoted-node-max | 1 | If `promotable` and `globally-unique` are **true**, the number of clone instances can be promoted at one time on a single node |

**Note: Deprecated Terminology**

In older documentation and online examples, you may see promotable clones referred to as *multi-state*, *stateful*, or *master/slave*; these mean the same thing as *promotable*. Certain syntax is supported for backward compatibility, but is deprecated and will be removed in a future version:

- Using a `master` tag, instead of a `clone` tag with the `promotable` meta-attribute set to `true`
- Using the `master-max` meta-attribute instead of `promoted-max`
- Using the `master-node-max` meta-attribute instead of `promoted-node-max`
- Using `Master` as a role name instead of `Promoted`
- Using `Slave` as a role name instead of `Unpromoted`

### Clone Contents

Clones must contain exactly one primitive or group resource.

**A clone that runs a web server on all nodes**

```
<clone id="apache-clone">
    <primitive id="apache" class="lsb" type="apache">
        <operations>
           <op id="apache-monitor" name="monitor" interval="30"/>
        </operations>
    </primitive>
</clone>
```

**Warning:** You should never reference the name of a clone's child (the primitive or group resource being cloned). If you think you need to do this, you probably need to re-evaluate your design.

#### Clone Instance Attribute

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

#### Clone Constraints

In most cases, a clone will have a single instance 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.

**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 node with an active instance of the clone. The cluster will choose an instance 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.

#### Promotable Clone Constraints

For promotable clone resources, the `first-action` and/or `then-action` fields for ordering constraints may be set to `promote` or `demote` to constrain the promoted role, and colocation constraints may contain `rsc-role` and/or `with-rsc-role` fields.

**Constraints involving promotable clone resources**

```
<constraints>
   <rsc_location id="db-prefers-node1" rsc="database" node="node1" score="500"/>
   <rsc_colocation id="backup-with-db-unpromoted" rsc="backup"
     with-rsc="database" with-rsc-role="Unpromoted"/>
   <rsc_colocation id="myapp-with-db-promoted" rsc="myApp"
     with-rsc="database" with-rsc-role="Promoted"/>
   <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 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 promotable clone resource means that it can run on any node with an active instance of the promotable clone resource that has the specified role (`Promoted` or `Unpromoted`). In the example above, the cluster will choose a location based on where database is running in the promoted role, and if there are multiple promoted instances it will also factor in **myApp**'s own location preferences when deciding which location to choose.

Colocation with regular clones and other promotable clone 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` promotable clone resource is (or will be) in the specified role. Placement is then performed as normal.

### Using Promotable Clone Resources in Colocation Sets

When a promotable clone is used in a *resource set* inside a colocation constraint, the resource set may take a `role` attribute.

In the following example, an instance of **B** may be promoted only on a node where **A** is in the promoted role. Additionally, resources **C** and **D** must be located on a node where both **A** and **B** are promoted.

**Colocate C and D with A's and B's promoted instances**

```
<constraints>
    <rsc_colocation id="coloc-1" score="INFINITY" >
      <resource_set id="colocated-set-example-1" sequential="true" role="Promoted">
        <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>
```

#### Using Promotable Clone Resources in Ordered Sets

When a promotable clone is used in a *resource set* inside an ordering constraint, the resource set may take an `action` attribute.

**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 until **A** has been promoted. Additionally, resources **C** and **D** must wait until **A** and **B** have been promoted before they can start.

#### Clone Stickiness

To achieve stable assignments, 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 instances 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.

**Important:** If `resource-stickiness` is set in the `rsc_defaults` section, it will apply to clone instances as well. This means an explicit `resource-stickiness` of 0 in `rsc_defaults` works differently from the implicit default used when `resource-stickiness` is not specified.

#### Monitoring Promotable Clone Resources

The usual monitor actions are insufficient to monitor a promotable clone 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 unpromoted role, and an additional one with `role="Promoted"` will cover the promoted role.

**Monitoring both states of a promotable clone resource**

```
<clone id="myPromotableRsc">
   <meta_attributes id="myPromotableRsc-meta">
       <nvpair name="promotable" value="true"/>
   </meta_attributes>
   <primitive id="myRsc" class="ocf" type="myApp" provider="myCorp">
    <operations>
     <op id="public-ip-unpromoted-check" name="monitor" interval="60"/>
     <op id="public-ip-promoted-check" name="monitor" interval="61" role="Promoted"/>
    </operations>
   </primitive>
</clone>
```

**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 promotable clone 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.

#### Determining Which Instance is Promoted

Pacemaker can choose a promotable clone instance to be promoted in one of two ways:

- Promotion scores: These are node attributes set via the `crm_attribute` command using the `--promotion` option, which generally would be called by the resource agent's start action if it supports promotable clones. This tool automatically detects both the resource and host, and should be used to set a preference for being promoted. Based on this, `promoted-max`, and `promoted-node-max`, the instance(s) with the highest preference will be promoted.
- Constraints: Location constraints can indicate which nodes are most preferred to be promoted.

**Explicitly preferring node1 to be promoted**

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

### 2.11.3 Bundles - Containerized Resources

Pacemaker supports a special syntax for launching a service inside a container with any infrastructure it requires: the *bundle*.

Pacemaker bundles support Docker, podman *(since 2.0.1)*, and rkt container technologies.[2]

[2] Docker is a trademark of Docker, Inc. No endorsement by or association with Docker, Inc. is implied.

**A bundle for a containerized web server**

```
<bundle id="httpd-bundle">
   <podman image="pcmk:http" replicas="3"/>
   <network ip-range-start="192.168.122.131"
            host-netmask="24"
            host-interface="eth0">
      <port-mapping id="httpd-port" port="80"/>
      </network>
   <storage>
      <storage-mapping id="httpd-syslog"
                       source-dir="/dev/log"
                       target-dir="/dev/log"
                       options="rw"/>
      <storage-mapping id="httpd-root"
                       source-dir="/srv/html"
                       target-dir="/var/www/html"
                       options="rw,Z"/>
      <storage-mapping id="httpd-logs"
                       source-dir-root="/var/log/pacemaker/bundles"
                       target-dir="/etc/httpd/logs"
                       options="rw,Z"/>
   </storage>
   <primitive class="ocf" id="httpd" provider="heartbeat" type="apache"/>
</bundle>
```

### Bundle Prerequisites

Before configuring a bundle in Pacemaker, the user must install the appropriate container launch technology (Docker, podman, or rkt), and supply a fully configured container image, on every node allowed to run the bundle.

Pacemaker will create an implicit resource of type **ocf:heartbeat:docker**, **ocf:heartbeat:podman**, or **ocf:heartbeat:rkt** to manage a bundle's container. The user must ensure that the appropriate resource agent is installed on every node allowed to run the bundle.

### Bundle Properties

Table 32: **XML Attributes of a bundle Element**

| Field | Description |
|---|---|
| id | A unique name for the bundle (required) |
| description | An optional description of the group, for the user's own purposes. E.g. `manages the container that runs the service` |

A bundle must contain exactly one `docker`, `podman`, or `rkt` element.

### Bundle Container Properties

Table 33: **XML attributes of a docker, podman, or rkt Element**

| Attribute | Default | Description |
|---|---|---|
| image | | Container image tag (required) |
| replicas | Value of `promoted-max` if that is positive, else 1 | A positive integer specifying the number of container instances to launch |
| replicas-per-host | 1 | A positive integer specifying the number of container instances allowed to run on a single node |
| promoted-max | 0 | A non-negative integer that, if positive, indicates that the containerized service should be treated as a promotable service, with this many replicas allowed to run the service in the promoted role |
| network | | If specified, this will be passed to the `docker run`, `podman run`, or `rkt run` command as the network setting for the container. |
| run-command | `/usr/sbin/pacemaker-remoted` if bundle contains a **primitive**, otherwise none | This command will be run inside the container when launching it ("PID 1"). If the bundle contains a **primitive**, this command *must* start `pacemaker-remoted` (but could, for example, be a script that does other stuff, too). |
| options | | Extra command-line options to pass to the `docker run`, `podman run`, or `rkt run` command |

**Note:** Considerations when using cluster configurations or container images from Pacemaker 1.1:

- If the container image has a pre-2.0.0 version of Pacemaker, set `run-command` to `/usr/sbin/pacemaker_remoted` (note the underbar instead of dash).
- `masters` is accepted as an alias for `promoted-max`, but is deprecated since 2.0.0, and support for it will be removed in a future version.

### Bundle Network Properties

A bundle may optionally contain one `<network>` element.

Table 34: **XML attributes of a network Element**

| Attribute | Default | Description |
|---|---|---|
| add-host | TRUE | If TRUE, and `ip-range-start` is used, Pacemaker will automatically ensure that `/etc/hosts` inside the containers has entries for each *replica name* and its assigned IP. |
| ip-range-start | | If specified, Pacemaker will create an implicit `ocf:heartbeat:IPaddr2` resource for each container instance, starting with this IP address, using up to `replicas` sequential addresses. These addresses can be used from the host's network to reach the service inside the container, though it is not visible within the container itself. Only IPv4 addresses are currently supported. |
| host-netmask | 32 | If `ip-range-start` is specified, the IP addresses are created with this CIDR netmask (as a number of bits). |
| host-interface | | If `ip-range-start` is specified, the IP addresses are created on this host interface (by default, it will be determined from the IP address). |
| control-port | 3121 | If the bundle contains a `primitive`, the cluster will use this integer TCP port for communication with Pacemaker Remote inside the container. Changing this is useful when the container is unable to listen on the default port, for example, when the container uses the host's network rather than `ip-range-start` (in which case `replicas-per-host` must be 1), or when the bundle may run on a Pacemaker Remote node that is already listening on the default port. Any `PCMK_remote_port` environment variable set on the host or in the container is ignored for bundle connections. |

**Note:** Replicas are named by the bundle id plus a dash and an integer counter starting with zero. For example, if a bundle named **httpd-bundle** has **replicas=2**, its containers will be named **httpd-bundle-0** and **httpd-bundle-1**.

Additionally, a `network` element may optionally contain one or more `port-mapping` elements.

Table 35: **Attributes of a port-mapping Element**

| Attribute | Default | Description |
| --- | --- | --- |
| id | | A unique name for the port mapping (required) |
| port | | If this is specified, connections to this TCP port number on the host network (on the container's assigned IP address, if `ip-range-start` is specified) will be forwarded to the container network. Exactly one of `port` or `range` must be specified in a `port-mapping`. |
| internal-port | value of `port` | If `port` and this are specified, connections to `port` on the host's network will be forwarded to this port on the container network. |
| range | | If this is specified, connections to these TCP port numbers (expressed as *first_port-last_port*) on the host network (on the container's assigned IP address, if `ip-range-start` is specified) will be forwarded to the same ports in the container network. Exactly one of `port` or `range` must be specified in a `port-mapping`. |

**Note:** If the bundle contains a `primitive`, Pacemaker will automatically map the `control-port`, so it is not necessary to specify that port in a `port-mapping`.

### Bundle Storage Properties

A bundle may optionally contain one `storage` element. A `storage` element has no properties of its own, but may contain one or more `storage-mapping` elements.

Table 36: **Attributes of a storage-mapping Element**

| Attribute | Default | Description |
| --- | --- | --- |
| id | | A unique name for the storage mapping (required) |
| source-dir | | The absolute path on the host's filesystem that will be mapped into the container. Exactly one of `source-dir` and `source-dir-root` must be specified in a `storage-mapping`. |
| source-dir-root | | The start of a path on the host's filesystem that will be mapped into the container, using a different subdirectory on the host for each container instance. The subdirectory will be named the same as the *replica name*. Exactly one of `source-dir` and `source-dir-root` must be specified in a `storage-mapping`. |
| target-dir | | The path name within the container where the host storage will be mapped (required) |
| options | | A comma-separated list of file system mount options to use when mapping the storage |

**Note:** Pacemaker does not define the behavior if the source directory does not already exist on the host. However, it is expected that the container technology and/or its resource agent will create the source directory in that case.

---

**Note:** If the bundle contains a `primitive`, Pacemaker will automatically map the equivalent of `source-dir=/etc/pacemaker/authkey target-dir=/etc/pacemaker/authkey` and `source-dir-root=/var/log/pacemaker/bundles target-dir=/var/log` into the container, so it is not necessary to specify those paths in a `storage-mapping`.

---

**Important:** The `PCMK_authkey_location` environment variable must not be set to anything other than the default of `/etc/pacemaker/authkey` on any node in the cluster.

---

**Important:** If SELinux is used in enforcing mode on the host, you must ensure the container is allowed to use any storage you mount into it. For Docker and podman bundles, adding "Z" to the mount options will create a container-specific label for the mount that allows the container access.

---

#### Bundle Primitive

A bundle may optionally contain one *primitive* resource. The primitive may have operations, instance attributes, and meta-attributes defined, as usual.

If a bundle contains a primitive resource, the container image must include the Pacemaker Remote daemon, and at least one of `ip-range-start` or `control-port` must be configured in the bundle. Pacemaker will create an implicit **ocf:pacemaker:remote** resource for the connection, launch Pacemaker Remote within the container, and monitor and manage the primitive resource via Pacemaker Remote.

If the bundle has more than one container instance (replica), the primitive resource will function as an implicit *clone* – a *promotable clone* if the bundle has `promoted-max` greater than zero.

---

**Note:** If you want to pass environment variables to a bundle's Pacemaker Remote connection or primitive, you have two options:

- Environment variables whose value is the same regardless of the underlying host may be set using the container element's `options` attribute.
- If you want variables to have host-specific values, you can use the *storage-mapping* element to map a file on the host as `/etc/pacemaker/pcmk-init.env` in the container *(since 2.0.3)*. Pacemaker Remote will parse this file as a shell-like format, with variables set as NAME=VALUE, ignoring blank lines and comments starting with "#".

---

**Important:** When a bundle has a `primitive`, Pacemaker on all cluster nodes must be able to contact Pacemaker Remote inside the bundle's containers.

- The containers must have an accessible network (for example, `network` should not be set to "none" with a `primitive`).
- The default, using a distinct network space inside the container, works in combination with `ip-range-start`. Any firewall must allow access from all cluster nodes to the `control-port` on the container IPs.

- If the container shares the host's network space (for example, by setting `network` to "host"), a unique `control-port` should be specified for each bundle. Any firewall must allow access from all cluster nodes to the `control-port` on all cluster and remote node IPs.

#### Bundle Node Attributes

If the bundle has a `primitive`, the primitive's resource agent may want to set node attributes such as *promotion scores*. However, with containers, it is not apparent which node should get the attribute.

If the container uses shared storage that is the same no matter which node the container is hosted on, then it is appropriate to use the promotion score on the bundle node itself.

On the other hand, if the container uses storage exported from the underlying host, then it may be more appropriate to use the promotion score on the underlying host.

Since this depends on the particular situation, the `container-attribute-target` resource meta-attribute allows the user to specify which approach to use. If it is set to `host`, then user-defined node attributes will be checked on the underlying host. If it is anything else, the local node (in this case the bundle node) is used as usual.

This only applies to user-defined attributes; the cluster will always check the local node for cluster-defined attributes such as `#uname`.

If `container-attribute-target` is `host`, the cluster will pass additional environment variables to the primitive's resource agent that allow it to set node attributes appropriately: `CRM_meta_container_attribute_target` (identical to the meta-attribute value) and `CRM_meta_physical_host` (the name of the underlying host).

**Note:** When called by a resource agent, the `attrd_updater` and `crm_attribute` commands will automatically check those environment variables and set attributes appropriately.

#### Bundle Meta-Attributes

Any meta-attribute set on a bundle will be inherited by the bundle's primitive and any resources implicitly created by Pacemaker for the bundle.

This includes options such as `priority`, `target-role`, and `is-managed`. See *Resource Options* for more information.

Bundles support clone meta-attributes including `notify`, `ordered`, and `interleave`.

#### Limitations of Bundles

Restarting pacemaker while a bundle is unmanaged or the cluster is in maintenance mode may cause the bundle to fail.

Bundles may not be explicitly cloned or included in groups. This includes the bundle's primitive and any resources implicitly created by Pacemaker for the bundle. (If `replicas` is greater than 1, the bundle will behave like a clone implicitly.)

Bundles do not have instance attributes, utilization attributes, or operations, though a bundle's primitive may have them.

A bundle with a primitive can run on a Pacemaker Remote node only if the bundle uses a distinct `control-port`.

# 2.12 Reusing Parts of the Configuration

Pacemaker provides multiple ways to simplify the configuration XML by reusing parts of it in multiple places.

Besides simplifying the XML, this also allows you to manipulate multiple configuration elements with a single reference.

## 2.12.1 Reusing Resource Definitions

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.

### 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.

**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.

**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:

**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.

**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
```

### Using Templates in Constraints

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

- `order` constraints (see *Specifying the Order in which Resources Should Start/Stop*)
- `colocation` constraints (see *Placing Resources Relative to other Resources*)
- `rsc_ticket` constraints (for multi-site clusters as described in *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 chapter:

```
<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.

### Using Templates in Resource Sets

Resource templates can also be referenced in resource sets.

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

```
<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 using a sequential resource set:

```
<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>
```

Or, if the resources referencing the template can run in parallel, then:

```
<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>
```

## 2.12.2 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:

**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 nonexistent `id` will cause a validation failure, as will attempting to remove a `rule` with an `id` that is referenced elsewhere.

Some rule syntax is allowed only in *certain contexts*. Validation cannot ensure that the referenced rule is allowed in the context of the rule containing `id-ref`, so such errors will be caught (and logged) only after the new configuration is accepted. It is the administrator's reponsibility to check for these.

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

**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>
```

`id-ref` can similarly be used with `resource_set` (in any constraint type), `nvpair`, and `operations`.

### 2.12.3 Tagging Configuration Elements

Pacemaker allows you to *tag* any configuration element that has an XML ID.

The main purpose of tagging is to support higher-level user interface tools; Pacemaker itself only uses tags within constraints. Therefore, what you can do with tags mostly depends on the tools you use.

### Configuring Tags

A tag is simply a named list of XML IDs.

**Tag referencing three resources**

```
<tags>
  <tag id="all-vms">
    <obj_ref id="vm1"/>
    <obj_ref id="vm2"/>
    <obj_ref id="vm3"/>
  </tag>
</tags>
```

What you can do with this new tag depends on what your higher-level tools support. For example, a tool might allow you to enable or disable all of the tagged resources at once, or show the status of just the tagged resources.

A single configuration element can be listed in any number of tags.

**Important:** If listing nodes in a tag, you must list the node's `id`, not name.

### Using Tags in Constraints and Resource Sets

Pacemaker itself only uses tags in constraints. If you supply a tag name instead of a resource name in any constraint, the constraint will apply to all resources listed in that tag.

**Constraint using a tag**

```
<rsc_order id="order1" first="storage" then="all-vms" kind="Mandatory" />
```

In the example above, assuming the `all-vms` tag is defined as in the previous example, the constraint will behave the same as:

**Equivalent constraints without tags**

```
<rsc_order id="order1-1" first="storage" then="vm1" kind="Mandatory" />
<rsc_order id="order1-2" first="storage" then="vm2" kind="Mandatory" />
<rsc_order id="order1-3" first="storage" then="vm3" kind="Mandatory" />
```

A tag may be used directly in the constraint, or indirectly by being listed in a *resource set* used in the constraint. When used in a resource set, an expanded tag will honor the set's `sequential` property.

### Filtering With Tags

The `crm_mon` tool can be used to display lots of information about the state of the cluster. On large or complicated clusters, this can include a lot of information, which makes it difficult to find the one thing you are interested in. The `--resource=` and `--node=` command line options can be used to filter results. In their most basic usage, these options take a single resource or node name. However, they can also be supplied with a tag name to display several objects at once.

For instance, given the following CIB section:

```
<resources>
  <primitive class="stonith" id="Fencing" type="fence_xvm"/>
  <primitive class="ocf" id="dummy" provider="pacemaker" type="Dummy"/>
  <group id="inactive-group">
    <primitive class="ocf" id="inactive-dummy-1" provider="pacemaker" type="Dummy"/>
    <primitive class="ocf" id="inactive-dummy-2" provider="pacemaker" type="Dummy"/>
  </group>
  <clone id="inactive-clone">
    <primitive id="inactive-dhcpd" class="lsb" type="dhcpd"/>
  </clone>
</resources>
<tags>
  <tag id="inactive-rscs">
    <obj_ref id="inactive-group"/>
    <obj_ref id="inactive-clone"/>
  </tag>
</tags>
```

The following would be output for `crm_mon --resource=inactive-rscs -r`:

```
Cluster Summary:
  * Stack: corosync
  * Current DC: cluster02 (version 2.0.4-1.e97f9675f.git.el7-e97f9675f) - partition with quorum
  * Last updated: Tue Oct 20 16:09:01 2020
  * Last change:  Tue May  5 12:04:36 2020 by hacluster via crmd on cluster01
  * 5 nodes configured
  * 27 resource instances configured (4 DISABLED)

Node List:
  * Online: [ cluster01 cluster02 ]

Full List of Resources:
  * Clone Set: inactive-clone [inactive-dhcpd] (disabled):
    * Stopped (disabled): [ cluster01 cluster02 ]
  * Resource Group: inactive-group (disabled):
    * inactive-dummy-1  (ocf::pacemaker:Dummy):  Stopped (disabled)
    * inactive-dummy-2  (ocf::pacemaker:Dummy):  Stopped (disabled)
```

## 2.13 Utilization and Placement Strategy

Pacemaker decides where a resource should run by assigning a score to every node, considering factors such as the resource's constraints and stickiness, then assigning the resource to the node with the highest score.

If more than one node has the highest score, Pacemaker by default chooses the one with the least number of assigned resources, or if that is also the same, the one listed first in the CIB. This results in simple load balancing.

Sometimes, simple load balancing is insufficient. Different resources can use significantly different amounts of a node's memory, CPU, and other capacities. Some combinations of resources may strain a node's capacity, causing them to fail or have degraded performance. Or, an administrator may prefer to concentrate resources rather than balance them, to minimize energy consumption by spare nodes.

Pacemaker offers flexibility by allowing you to configure *utilization attributes* specifying capacities that each node provides and each resource requires, as well as a *placement strategy*.

### 2.13.1 Utilization attributes

You can define any number of utilization attributes to represent capacities of interest (CPU, memory, I/O bandwidth, etc.). Their values must be integers.

The nature and units of the capacities are irrelevant to Pacemaker. It just makes sure that each node has sufficient capacity to run the resources assigned to it.

**Specifying CPU and RAM capacities of two nodes**

```xml
<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>
```

**Specifying CPU and RAM consumed by several resources**

```xml
<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>
```

Utilization attributes for a node may be permanent or *(since 2.1.6)* transient. Permanent attributes persist after Pacemaker is restarted, while transient attributes do not.

**Transient utilization attribute for node cluster-1**

```
<transient_attributes id="cluster-1">
  <utilization id="status-cluster-1">
    <nvpair id="status-cluster-1-cpu" name="cpu" value="1"/>
  </utilization>
</transient_attributes>
```

Utilization attributes may be configured only on primitive resources. Pacemaker will consider a collective resource's utilization based on the primitives it contains.

**Note:** Utilization is supported for bundles *(since 2.1.3)*, but only for bundles with an inner primitive.

### 2.13.2 Placement Strategy

The `placement-strategy` cluster option determines how utilization attributes are used. Its allowed values are:

- `default`: The cluster ignores utilization values, and places resources according to (from highest to lowest precedence) assignment scores, the number of resources already assigned to each node, and the order nodes are listed in the CIB.
- `utilization`: The cluster uses the same method as the default strategy to assign a resource to a node, but only nodes with sufficient free capacity to meet the resource's requirements are eligible.
- `balanced`: Only nodes with sufficient free capacity are eligible to run a resource, and the cluster load-balances based on the sum of resource utilization values rather than the number of resources.
- `minimal`: Only nodes with sufficient free capacity are eligible to run a resource, and the cluster concentrates resources on as few nodes as possible.

To look at it another way, when deciding where to run a resource, the cluster starts by considering all nodes, then applies these criteria one by one until a single node remains:

- If `placement-strategy` is `utilization`, `balanced`, or `minimal`, consider only nodes that have sufficient spare capacities to meet the resource's requirements.
- Consider only nodes with the highest score for the resource. Scores take into account factors such as the node's health; the resource's stickiness, failure count on the node, and migration threshold; and constraints.
- If `placement-strategy` is `balanced`, consider only nodes with the most free capacity.
- If `placement-strategy` is `default`, `utilization`, or `balanced`, consider only nodes with the least number of assigned resources.
- If more than one node is eligible after considering all other criteria, choose the one listed first in the CIB.

### 2.13.3 How Multiple Capacities Combine

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

If multiple utilization attributes have been defined, then the node that has the highest value in 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.

### 2.13.4 Order of Resource Assignment

When assigning resources to nodes, the cluster chooses the next one to assign by considering the following criteria one by one until a single resource is selected:

- Assign the resource with the highest *priority*.
- If any resources are already active, assign the one with the highest score on its current node. This avoids unnecessary resource shuffling.
- Assign the resource with the highest score on its preferred node.
- If more than one resource remains after considering all other criteria, assign the one of them that is listed first in the CIB.

**Note:** For bundles, only the priority set for the bundle itself matters. If the bundle contains a primitive, the primitive's priority is ignored.

### 2.13.5 Limitations

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

In a high-availability cluster, it is unacceptable to spend minutes, let alone hours or days, finding an optimal solution while services are down.

Instead of trying to solve the problem completely, Pacemaker uses a "best effort" algorithm. This arrives at a quick solution, but at the cost of possibly leaving some resources stopped unnecessarily.

Using the example configuration at the start of this chapter, and the balanced placement strategy:

- `rsc-small` would be assigned to `node1`
- `rsc-medium` would be assigned to `node2`
- `rsc-large` would remain inactive

That 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 maxed out even under normal conditions, 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 capacities advertised by the nodes.**

  Advertise slightly more resources than we physically have, on the (usually valid) assumption that resources will not always use 100% of their configured utilization. This practice is sometimes called *overcommitting*.

- **Specify resource priorities.**

  If the cluster is going to sacrifice services, it should be the ones you care about the least.

## 2.14 Access Control Lists (ACLs)

By default, the `root` user or any user in the `haclient` group can modify Pacemaker's CIB without restriction. Pacemaker offers *access control lists (ACLs)* to provide more fine-grained authorization.

**Important:** Being able to modify the CIB's resource section allows a user to run any executable file as root, by configuring it as an LSB resource with a full path.

### 2.14.1 ACL Prerequisites

In order to use ACLs:

- The `enable-acl` *cluster option* must be set to true.
- Desired users must have user accounts in the `haclient` group on all cluster nodes in the cluster.
- If your CIB was created before Pacemaker 1.1.12, it might need to be updated to the current schema (using `cibadmin --upgrade` or a higher-level tool equivalent) in order to use the syntax documented here.
- Prior to the 2.1.0 release, the Pacemaker software had to have been built with ACL support. If you are using an older release, your installation supports ACLs only if the output of the command `pacemakerd --features` contains `acls`. In newer versions, ACLs are always enabled.

### 2.14.2 ACL Configuration

ACLs are specified within an `acls` element of the CIB. The `acls` element may contain any number of `acl_role`, `acl_target`, and `acl_group` elements.

### 2.14.3 ACL Roles

An ACL *role* is a collection of permissions allowing or denying access to particular portions of the CIB. A role is configured with an `acl_role` element in the CIB `acls` section.

Table 37: **Properties of an acl_role element**

| Attribute | Description |
|---|---|
| id | A unique name for the role *(required)* |
| description | Arbitrary text (not used by Pacemaker) |

An `acl_role` element may contain any number of `acl_permission` elements.

Table 38: **Properties of an acl_permission element**

| Attribute | Description |
|---|---|
| id | A unique name for the permission *(required)* |
| description | Arbitrary text (not used by Pacemaker) |
| kind | The access being granted. Allowed values are `read`, `write`, and `deny`. A value of `write` grants both read and write access. |
| object-type | The name of an XML element in the CIB to which the permission applies. (Exactly one of `object-type`, `xpath`, and `reference` must be specified for a permission.) |
| attribute | If specified, the permission applies only to `object-type` elements that have this attribute set (to any value). If not specified, the permission applies to all `object-type` elements. May only be used with `object-type`. |
| reference | The ID of an XML element in the CIB to which the permission applies. (Exactly one of `object-type`, `xpath`, and `reference` must be specified for a permission.) |
| xpath | An XPath specification selecting an XML element in the CIB to which the permission applies. Attributes may be specified in the XPath to select particular elements, but the permissions apply to the entire element. (Exactly one of `object-type`, `xpath`, and `reference` must be specified for a permission.) |

**Important:**

- Permissions are applied to the selected XML element's entire XML subtree (all elements enclosed within it).
- Write permission grants the ability to create, modify, or remove the element and its subtree, and also the ability to create any "scaffolding" elements (enclosing elements that do not have attributes other than an ID).
- Permissions for more specific matches (more deeply nested elements) take precedence over more general ones.
- If multiple permissions are configured for the same match (for example, in different roles applied to the same user), any `deny` permission takes precedence, then `write`, then lastly `read`.

### 2.14.4 ACL Targets and Groups

ACL targets correspond to user accounts on the system.

Table 39: **Properties of an acl_target element**

| Attribute | Description |
|---|---|
| id | A unique identifier for the target (if `name` is not specified, this must be the name of the user account) *(required)* |
| name | If specified, the user account name (this allows you to specify a user name that is already used as the `id` for some other configuration element) *(since 2.1.5)* |

ACL groups correspond to groups on the system. Any role configured for these groups apply to all users in that group *(since 2.1.5)*.

Table 40: **Properties of an acl_group element**

| Attribute | Description |
|---|---|
| id | A unique identifier for the group (if `name` is not specified, this must be the group name) *(required)* |
| name | If specified, the group name (this allows you to specify a group name that is already used as the `id` for some other configuration element) |

Each `acl_target` and `acl_group` element may contain any number of `role` elements.

**Note:** If the system users and groups are defined by some network service (such as LDAP), the cluster itself will be unaffected by outages in the service, but affected users and groups will not be able to make changes to the CIB.

Table 41: **Properties of a role element**

| Attribute | Description |
|---|---|
| id | The `id` of an `acl_role` element that specifies permissions granted to the enclosing target or group. |

**Important:** The `root` and `hacluster` user accounts always have full access to the CIB, regardless of ACLs. For all other user accounts, when `enable-acl` is true, permission to all parts of the CIB is denied by default (permissions must be explicitly granted).

### 2.14.5 ACL Examples

```
<acls>

   <acl_role id="read_all">
       <acl_permission id="read_all-cib" kind="read" xpath="/cib" />
   </acl_role>

   <acl_role id="operator">

       <acl_permission id="operator-maintenance-mode" kind="write"
           xpath="//crm_config//nvpair[@name='maintenance-mode']" />

       <acl_permission id="operator-maintenance-attr" kind="write"
           xpath="//nvpair[@name='maintenance']" />

       <acl_permission id="operator-target-role" kind="write"
           xpath="//resources//meta_attributes/nvpair[@name='target-role']" />

       <acl_permission id="operator-is-managed" kind="write"
           xpath="//resources//nvpair[@name='is-managed']" />
```

(continues on next page)

(continued from previous page)

```
    <acl_permission id="operator-rsc_location" kind="write"
        object-type="rsc_location" />

</acl_role>

<acl_role id="administrator">
    <acl_permission id="administrator-cib" kind="write" xpath="/cib" />
</acl_role>

<acl_role id="minimal">

    <acl_permission id="minimal-standby" kind="read"
        description="allow reading standby node attribute (permanent or transient)"
        xpath="//instance_attributes/nvpair[@name='standby']"/>

    <acl_permission id="minimal-maintenance" kind="read"
        description="allow reading maintenance node attribute (permanent or transient)"
        xpath="//nvpair[@name='maintenance']"/>

    <acl_permission id="minimal-target-role" kind="read"
        description="allow reading resource target roles"
        xpath="//resources//meta_attributes/nvpair[@name='target-role']"/>

    <acl_permission id="minimal-is-managed" kind="read"
        description="allow reading resource managed status"
        xpath="//resources//meta_attributes/nvpair[@name='is-managed']"/>

    <acl_permission id="minimal-deny-instance-attributes" kind="deny"
        xpath="//instance_attributes"/>

    <acl_permission id="minimal-deny-meta-attributes" kind="deny"
        xpath="//meta_attributes"/>

    <acl_permission id="minimal-deny-operations" kind="deny"
        xpath="//operations"/>

    <acl_permission id="minimal-deny-utilization" kind="deny"
        xpath="//utilization"/>

    <acl_permission id="minimal-nodes" kind="read"
        description="allow reading node names/IDs (attributes are denied separately)"
        xpath="/cib/configuration/nodes"/>

    <acl_permission id="minimal-resources" kind="read"
        description="allow reading resource names/agents (parameters are denied separately)"
        xpath="/cib/configuration/resources"/>

    <acl_permission id="minimal-deny-constraints" kind="deny"
        xpath="/cib/configuration/constraints"/>

    <acl_permission id="minimal-deny-topology" kind="deny"
        xpath="/cib/configuration/fencing-topology"/>

    <acl_permission id="minimal-deny-op_defaults" kind="deny"
        xpath="/cib/configuration/op_defaults"/>
```

(continues on next page)

(continued from previous page)

```
       <acl_permission id="minimal-deny-rsc_defaults" kind="deny"
           xpath="/cib/configuration/rsc_defaults"/>

       <acl_permission id="minimal-deny-alerts" kind="deny"
           xpath="/cib/configuration/alerts"/>

       <acl_permission id="minimal-deny-acls" kind="deny"
           xpath="/cib/configuration/acls"/>

       <acl_permission id="minimal-cib" kind="read"
           description="allow reading cib element and crm_config/status sections"
           xpath="/cib"/>

   </acl_role>

   <acl_target id="alice">
      <role id="minimal"/>
   </acl_target>

   <acl_target id="bob">
      <role id="read_all"/>
   </acl_target>

   <acl_target id="carol">
      <role id="read_all"/>
      <role id="operator"/>
   </acl_target>

   <acl_target id="dave">
      <role id="administrator"/>
   </acl_target>

</acls>
```

In the above example, the user **alice** has the minimal permissions necessary to run basic Pacemaker CLI tools, including using **crm_mon** to view the cluster status, without being able to modify anything. The user **bob** can view the entire configuration and status of the cluster, but not make any changes. The user **carol** can read everything, and change selected cluster properties as well as resource roles and location constraints. Finally, **dave** has full read and write access to the entire CIB.

Looking at the **minimal** role in more depth, it is designed to allow read access to the **cib** tag itself, while denying access to particular portions of its subtree (which is the entire CIB).

This is because the DC node is indicated in the **cib** tag, so **crm_mon** will not be able to report the DC otherwise. However, this does change the security model to allow by default, since any portions of the CIB not explicitly denied will be readable. The **cib** read access could be removed and replaced with read access to just the **crm_config** and **status** sections, for a safer approach at the cost of not seeing the DC in status output.

For a simpler configuration, the **minimal** role allows read access to the entire **crm_config** section, which contains cluster properties. It would be possible to allow read access to specific properties instead (such as **stonith-enabled**, **dc-uuid**, **have-quorum**, and **cluster-name**) to restrict access further while still allowing status output, but cluster properties are unlikely to be considered sensitive.

### 2.14.6 ACL Limitations

#### Actions performed via IPC rather than the CIB

ACLs apply *only* to the CIB.

That means ACLs apply to command-line tools that operate by reading or writing the CIB, such as `crm_attribute` when managing permanent node attributes, `crm_mon`, and `cibadmin`.

However, command-line tools that communicate directly with Pacemaker daemons via IPC are not affected by ACLs. For example, users in the `haclient` group may still do the following, regardless of ACLs:

- Query transient node attribute values using `crm_attribute` and `attrd_updater`.
- Query basic node information using `crm_node`.
- Erase resource operation history using `crm_resource`.
- Query fencing configuration information, and execute fencing against nodes, using `stonith_admin`.

#### ACLs and Pacemaker Remote

ACLs apply to commands run on Pacemaker Remote nodes using the Pacemaker Remote node's name as the ACL user name.

The idea is that Pacemaker Remote nodes (especially virtual machines and containers) are likely to be purpose-built and have different user accounts from full cluster nodes.

## 2.15 Status

Pacemaker automatically generates a `status` section in the CIB (inside the `cib` element, at the same level as `configuration`). The status is transient, and is not stored to disk with the rest of the CIB.

The section's structure and contents are internal to Pacemaker and subject to change from release to release. Its often obscure element and attribute names are kept for historical reasons, to maintain compatibility with older versions during rolling upgrades.

Users should not modify the section directly, though various command-line tool options affect it indirectly.

### 2.15.1 Node State

The `status` element contains `node_state` elements for each node in the cluster (and potentially nodes that have been removed from the configuration since the cluster started). The `node_state` element has attributes that allow the cluster to determine whether the node is healthy.

**Example minimal node state entry**

```
<node_state id="1" uname="cl-virt-1" in_ccm="1721760952" crmd="1721760952" crm-debug-origin=
↪"controld_update_resource_history" join="member" expected="member">
 <transient_attributes id="1"/>
 <lrm id="1"/>
</node_state>
```

Table 42: **Attributes of a node_state Element**

| Name | Type | Description |
|---|---|---|
| id | *text* | Node ID (identical to `id` of corresponding `node` element in the `configuration` section) |
| uname | *text* | Node name (identical to `uname` of corresponding `node` element in the `configuration` section) |
| in_ccm | *epoch time (since 2.1.7; previously boolean)* | If the node's controller is currently in the cluster layer's membership, this is the epoch time at which it joined (or 1 if the node is in the process of leaving the cluster), otherwise 0 *(since 2.1.7; previously, it was "true" or "false")* |
| crmd | *epoch time (since 2.1.7; previously an enumeration)* | If the node's controller is currently in the cluster layer's controller messaging group, this is the epoch time at which it joined, otherwise 0 *(since 2.1.7; previously, the value was either "online" or "offline")* |
| crm-debug-origin | *text* | Name of the source code function that recorded this `node_state` element (for debugging) |
| join | *enumeration* | Current status of node's controller join sequence (and thus whether it is eligible to run resources). Allowed values:<br>• `down`: Not yet joined<br>• `pending`: In the process of joining or leaving<br>• `member`: Fully joined<br>• `banned`: Rejected by DC |
| expected | *enumeration* | What cluster expects `join` to be in the immediate future. Allowed values are same as for `join`. |

## 2.15.2 Transient Node Attributes

The `transient_attributes` section specifies transient *Node Attributes*. In addition to any values set by the administrator or resource agents using the `attrd_updater` or `crm_attribute` tools, the cluster stores various state information here.

**Example transient node attributes for a node**

```
<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-fail-count-pingd:0.monitor_30000" name="fail-count-pingd:0
↪#monitor_30000" value="1"/>
     <nvpair id="status-cl-virt-1-last-failure-pingd:0" name="last-failure-pingd:0" value=
↪"1239009742"/>
  </instance_attributes>
</transient_attributes>
```

## 2.15.3 Node History

Each `node_state` element contains an `lrm` element with a history of certain resource actions performed on the node. The `lrm` element contains an `lrm_resources` element.

### Resource History

The `lrm_resources` element contains an `lrm_resource` element for each resource that has had an action performed on the node.

An `lrm_resource` entry has attributes allowing the cluster to stop the resource safely even if it is removed from the configuration. Specifically, the resource's `id`, `class`, `type` and `provider` are recorded.

### Action History

Each `lrm_resource` element contains an `lrm_rsc_op` element for each recorded action performed for that resource on that node. (Not all actions are recorded, just enough to determine the resource's state.)

Table 43: **Attributes of an lrm_rsc_op element**

| Name | Type | Description |
|---|---|---|
| id | *text* | Identifier for the history entry constructed from the resource ID, action name or history entry type, and action interval. |
| operation_key | *text* | Identifier for the action that was executed, constructed from the resource ID, action name, and action interval. |
| operation | *text* | The name of the action the history entry is for |
| crm-debug-origin | *text* | Name of the source code function that recorded this entry (for debugging) |
| crm_feature_set | *version* | The Pacemaker feature set used to record this entry. |
| transition-key | *text* | A concatenation of the action's transition graph action number, the transition graph number, the action's expected result, and the UUID of the controller instance that scheduled it. |
| transition-magic | *text* | A concatenation of `op-status`, `rc-code`, and `transition-key`. |
| exit-reason | *text* | An error message (if available) from the resource agent or Pacemaker if the action did not return success. |
| on_node | *text* | The name of the node that executed the action (identical to the `uname` of the enclosing `node_state` element) |
| call-id | *integer* | A node-specific counter used to determine the order in which actions were executed. |
| rc-code | *integer* | The resource agent's exit status for this action. Refer to the *Resource Agents* chapter of *Pacemaker Administration* for how these values are interpreted. |
| op-status | *integer* | The execution status of this action. The meanings of these codes are internal to Pacemaker. |
| interval | *nonnegative integer* | If the action is recurring, its frequency (in milliseconds), otherwise 0. |
| last-rc-change | *epoch time* | Node-local time at which the action first returned the current value of `rc-code`. |
| exec-time | *integer* | Time (in seconds) that action execution took (if known) |
| queue-time | *integer* | Time (in seconds) that action was queued in the local executor (if known) |
| op-digest | *text* | If present, this is a hash of the parameters passed to the action. If a hash of the currently configured parameters does not match this, that means the resource configuration changed since the action was performed, and the resource must be reloaded or restarted. |

Continued on next page

Table 43 – continued from previous page

| Name | Type | Description |
|---|---|---|
| op-restart-digest | *text* | If present, the resource agent supports reloadable parameters, and this is a hash of the non-reloadable parameters passed to the action. This allows the cluster to choose between reload and restart when one is needed. |
| op-secure-digest | *text* | If present, the resource agent marks some parameters as sensitive, and this is a hash of the non-sensitive parameters passed to the action. This allows the value of sensitive parameters to be removed from a saved copy of the CIB while still allowing scheduler simulations to be performed on that copy. |

#### Simple Operation History Example

**A monitor operation (determines current state of the `apcstonith` resource)**

```
<lrm_resource id="apcstonith" type="fence_apc_snmp" class="stonith">
  <lrm_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="2e3da9274d3550dc6526fb24bfcbcba0"
    transition-key="22:2:7:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    transition-magic="0:7;22:2:7:2668bbeb-06d5-40f9-936d-24cb7f87006a"
    last-rc-change="1239008085" exec-time="10" queue-time="0"/>
</lrm_resource>
```

The above example shows the history entry for a probe (non-recurring monitor operation) 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 controller (2668bbeb-06d5-40f9-936d-24cb7f87006a).

The third field of the `transition-key` contains a 7, which indicates that the cluster 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 action recorded for this node, we can conclude that the cluster started the resource elsewhere.

#### Complex Operation History Example

**Resource history of a `pingd` clone with multiple entries**

```
<lrm_resource id="pingd:0" type="pingd" class="ocf" provider="pacemaker">
  <lrm_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-rc-change="1239009741" exec-time="10" queue-time="0"/>
  <lrm_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-rc-change="1239009741" exec-time="10" queue-time="0"/>
  <lrm_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-rc-change="1239009741" exec-time="10" queue-time="0" />
  <lrm_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-rc-change="1239008085" exec-time="20" queue-time="0"/>
  </lrm_resource>
```

When more than one history entry 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 history entry 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 controller 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.

## 2.16 Multi-Site Clusters and Tickets

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)*.

### 2.16.1 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.

### 2.16.2 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.

#### 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`.

#### 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.

#### 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.

#### 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.

### 2.16.3 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 the promoted role.

**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 promotable resource, you might want to configure that only being in the promoted role depends on `ticketA`. With the following configuration, `rsc1` will be demoted if `ticketA` is revoked:

**Constraint that demotes `rsc1` if `ticketA` is revoked**

```
<rsc_ticket id="rsc1-req-ticketA" rsc="rsc1" rsc-role="Promoted" 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 *Resource Sets*), so one can easily list all the resources in one **rsc_ticket** constraint instead.

**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="Promoted">
    <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`.

### 2.16.4 Managing Multi-Site Clusters

#### 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.

### Granting and Revoking Tickets via a Cluster Ticket Registry

We will use Booth here as an example of software that can be used with pacemaker as a Cluster Ticket Registry. Booth implements the Raft 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.

**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.

### 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.

### 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
```

### 2.16.5 For more information

- SUSE's Geo Clustering quick start
- Booth

## 2.17 Sample Configurations

### 2.17.1 Empty

**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>
```

### 2.17.2 Simple

**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.

### 2.17.3 Advanced Configuration

**An advanced configuration with groups, clones and STONITH**

```xml
<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"/>
```

CHAPTER

THREE

# INDEX

- genindex
- search

# INDEX

## Symbols



## A

## G



## H



## I

## J



## K



## L



## M



## N

## S

## T

## U



## V



## W



## X



## Y