

Pacemaker

Pacemaker Configuration

Explained

by Andrew Beekhof

pacemaker@clusterlabs.org

Last updated on Tuesday, August 4, 2009 for Pacemaker 1.0.4

Table of Contents

| | |
|---|-----------|
| Read-Me-First | ix |
| What Is Pacemaker? | ix |
| Pacemaker Architecture | x |
| The Scope of this Document | xi |
| Configuration Basics | 1 |
| Configuration Layout | 1 |
| The Current State of the Cluster | 1 |
| How Should the Configuration be Updated? | 3 |
| Quickly Deleting Part of the Configuration | 3 |
| Updating the Configuration Without Using XML | 4 |
| Making Configuration Changes in a Sandbox | 4 |
| Testing Your Configuration Changes | 5 |
| Do I Need to Update the Configuration on all Cluster Nodes? | 6 |
| Cluster Options | 8 |
| Special Options | 8 |
| Determining Which Configuration to Use | 8 |
| Other Fields | 8 |
| Fields Maintained by the Cluster | 8 |
| Cluster Options | 9 |
| Available Cluster Options | 9 |
| Cluster Configuration Explained | i |

| | |
|---|-----------|
| Querying and Setting Cluster Options | 10 |
| When Options are Listed More Than Once | 10 |
| Cluster Nodes | 11 |
| Defining a Cluster Node | 11 |
| Describing a Cluster Node | 11 |
| Adding a New Cluster Node | 12 |
| OpenAIS | 12 |
| Heartbeat | 12 |
| Removing a Cluster Node | 12 |
| OpenAIS | 12 |
| Heartbeat | 12 |
| Replacing a Cluster Node | 12 |
| OpenAIS | 12 |
| Heartbeat | 12 |
| Cluster Resources | 14 |
| What is a Cluster Resource | 14 |
| Supported Resource Classes | 14 |
| Open Cluster Framework | 14 |
| Linux Standard Base | 14 |
| Legacy Heartbeat | 15 |
| Properties | 15 |
| Options | 16 |
| Setting Global Defaults for Resource Options | 17 |
| Instance Attributes | 17 |
| Resource Operations | 18 |

| | |
|--|-----------|
| Monitoring Resources for Failure | 18 |
| Setting Global Defaults for Operations | 18 |
| When Resources Take a Long Time to Start/Stop | 19 |
| Multiple Monitor Operations | 19 |
| Disabling a Monitor Operation | 20 |
| Resource Constraints | 21 |
| Scores | 21 |
| Infinity Math | 21 |
| Deciding Which Nodes a Resource Can Run On | 21 |
| Options | 21 |
| Asymmetrical “Opt-In” Clusters | 21 |
| Symmetrical “Opt-Out” Clusters | 22 |
| What if Two Nodes Have the Same Score | 22 |
| Specifying the Order Resources Should Start/Stop In | 22 |
| Mandatory Ordering | 23 |
| Advisory Ordering | 23 |
| Placing Resources Relative to other Resources | 23 |
| Options | 23 |
| Mandatory Placement | 24 |
| Advisory Placement | 24 |
| Ordering Sets of Resources | 24 |
| Collocating Sets of Resources | 27 |
| Rules | 30 |
| Node Attribute Expressions | 30 |
| Time/Date Based Expressions | 30 |

| | |
|--|-----------|
| Date Specifications | 31 |
| Durations | 32 |
| <i>Sample Time Based Expressions</i> | 32 |
| Using Rules to Determine Resource Location | 34 |
| <i>Using score-attribute Instead of score</i> | 34 |
| Using Rules to Control Resource Options | 35 |
| Using Rules to Control Cluster Options | 36 |
| Ensuring Time Based Rules Take Effect | 36 |
| Advanced Configuration | 37 |
| Connecting to the Cluster Configuration from a Remote Machine | 37 |
| Specifying When Recurring Actions are Performed | 37 |
| Moving Resources | 38 |
| Manual Intervention | 38 |
| Moving Resources Due to Failure | 39 |
| Moving Resources Due to Connectivity Changes | 40 |
| Resource Migration | 41 |
| <i>Migration Checklist</i> | 42 |
| Reusing Rules, Options and Sets of Operations | 43 |
| Advanced Resource Types | 44 |
| Groups - A Syntactic Shortcut | 44 |
| Properties | 45 |
| Options | 45 |
| Using Groups | 45 |
| <i>Instance Attributes</i> | 45 |
| <i>Contents</i> | 45 |

| | |
|--|-----------|
| <i>Constraints</i> | 45 |
| <i>Stickiness</i> | 45 |
| Clones - Resources That Should be Active on Multiple Hosts | 45 |
| Properties | 46 |
| Options | 46 |
| Using Clones | 47 |
| <i>Instance Attributes</i> | 47 |
| <i>Contents</i> | 47 |
| <i>Constraints</i> | 47 |
| <i>Stickiness</i> | 47 |
| <i>Resource Agent Requirements</i> | 47 |
| <i>Notifications</i> | 48 |
| <i>Proper Interpretation of Notification Environment Variables</i> | 48 |
| Multi-state - Resources That Have Multiple Modes | 49 |
| PropertiesOptions | 49 |
| Using Multi-state Resources | 50 |
| <i>Instance Attributes</i> | 50 |
| <i>Contents</i> | 50 |
| <i>Monitoring Multi-State Resources</i> | 50 |
| <i>Constraints</i> | 50 |
| <i>Stickiness</i> | 51 |
| <i>Which Resource Instance is Promoted</i> | 51 |
| <i>Resource Agent Requirements</i> | 51 |
| <i>Notifications</i> | 52 |
| <i>Proper Interpretation of Notification Environment Variables</i> | 53 |

| | |
|--|-----------|
| Protecting Your Data - STONITH | 56 |
| Why You Need STONITH | 56 |
| What STONITH Device Should You Use | 56 |
| Configuring STONITH | 56 |
| <i>Example</i> | 57 |
| Status - Here be dragons | 58 |
| Transient Node Attributes | 58 |
| Operation History | 58 |
| Appendix: FAQ | 59 |
| Why is the Project Called Pacemaker? | 59 |
| Why was the Pacemaker Project Created? | 59 |
| What Messaging Layers are Supported? | 59 |
| Can I Choose which Messaging Layer to use at Run Time? | 59 |
| Can I Have a Mixed Heartbeat-OpenAIS Cluster? | 59 |
| Where Can I Get Pre-built Packages? | 59 |
| What Versions of Pacemaker Are Supported? | 60 |
| Appendix: More About OCF Resource Agents | 61 |
| Location of Custom Scripts | 61 |
| Actions | 61 |
| How Does the Cluster Interpret the OCF Return Codes? | 62 |
| <i>Exceptions</i> | 63 |
| Appendix: What Changed in 1.0 | 64 |
| New | 64 |
| Changed | 64 |
| Removed | 65 |

| | |
|---|-----------|
| Appendix: Installation | 66 |
| Choosing a Cluster Stack | 66 |
| Enabling Pacemaker | 66 |
| For OpenAIS | 66 |
| For Heartbeat | 67 |
| Appendix: Upgrading Cluster Software | 68 |
| Upgrade Methodologies | 68 |
| Version Compatibility | 68 |
| Complete Cluster Shutdown | 68 |
| <i>Procedure</i> | 68 |
| Rolling (node by node) | 69 |
| <i>Procedure</i> | 69 |
| <i>Version Compatibility</i> | 69 |
| <i>Crossing Compatibility Boundaries</i> | 69 |
| Disconnect & Reattach | 69 |
| <i>Procedure</i> | 69 |
| <i>Notes</i> | 70 |
| Appendix: Upgrading the Configuration from 0.6 | 71 |
| Overview | 71 |
| Preparation | 71 |
| Perform the upgrade | 71 |
| <i>Upgrade the software</i> | 71 |
| <i>Upgrade the Configuration</i> | 71 |
| <i>Manually Upgrading the Configuration</i> | 72 |
| Appendix: Is This init Script LSB Compatible? | 73 |

| | |
|--|-----------|
| Appendix: Sample Configurations | 74 |
| An Empty Configuration | 74 |
| A Simple Configuration | 74 |
| An Advanced Configuration | 75 |
| Appendix: Further Reading | 77 |
| <i>Project Website</i> | 77 |
| <i>Cluster Commands</i> | 77 |
| <i>Heartbeat configuration</i> | 77 |
| <i>OpenAIS Configuration</i> | 77 |

Read-Me-First

What Is Pacemaker?

Pacemaker is a cluster resource manager. It makes use of your cluster infrastructure (either OpenAIS or Heartbeat) to stop, start and monitor the health of the services (aka. resources) you want the cluster to provide.

It can do this for clusters of practically any size and comes with a powerful dependency model that allows the administrator to accurately express the relationships (both ordering and location) between the cluster resources.



Conceptual overview of the cluster stack.

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

- Core cluster infrastructure providing messaging and membership functionality (illustrated in red)
- Non-cluster aware components (illustrated in blue). In a Pacemaker cluster, these pieces include not only the scripts that knows how to start, stop and monitor resources, but also a local daemon that masks the differences between the different standards these scripts implement.
- A brain (illustrated in green) that processes and reacts to events from the cluster (nodes leaving or joining) and resources (eg. monitor failures) as well as configuration changes from the administrator. In response to all of these events, Pacemaker will compute the ideal state of the cluster and plot a path to achieve it. This may include moving resources, stopping nodes and even forcing them offline with remote power switches.

Pacemaker Architecture

Pacemaker itself is composed of four key components (illustrated below in the same color scheme as the previous diagram):

- CIB (aka. Cluster Information Base)
- CRMD (aka. Cluster Resource Management daemon)
- PEngine (aka. PE or Policy Engine)
- STONITHd

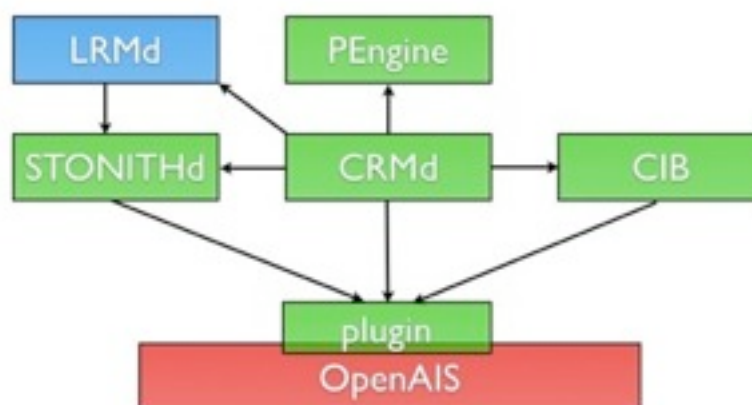
The CIB uses XML to represent both the cluster's configuration and current state of all resources in the cluster. The contents of the CIB are automatically kept in sync across the entire cluster and are used by the PEngine to compute the ideal state of the cluster and how it should be achieved.

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

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

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

In some cases, it may be necessary to power off nodes in order to protect shared data or complete resource recovery. For this Pacemaker comes with STONITHd. STONITH is an acronym for Shoot-The-Other-Node-In-The-Head and is usually implemented with a remote power switch. In Pacemaker, STONITH devices are modeled as resources (and configured in the CIB) to enable them to be easily monitored for failure, however STONITHd takes care of understanding the STONITH topology such that its clients simply request a node be fenced and it does the rest.



Subsystems of a Pacemaker cluster running on OpenAIS

The Scope of this Document

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

For those that are allergic to XML, Pacemaker comes with a cluster shell and a Python based GUI exists, however these tools will not be covered at all in this document¹, precisely because they hide the XML.

Additionally, this document is NOT a step-by-step how-to guide for configuring a specific clustering scenario. Although such guides are likely to be written in the future, the purpose of this document is to provide an understanding of the building blocks that can be used to construct any type of Pacemaker cluster.

¹ It is hoped however, that having understood the concepts explained here, that the functionality of these tools will also be more readily understood.

Configuration Basics

Configuration Layout

The cluster is written using XML notation and divided into two main sections; configuration and status.

The status section contains the history of each resource on each node and based on this data, the cluster can construct the complete current state of the cluster. The authoritative source for the status section is the local resource manager (lrmd) process on each cluster node and the cluster will occasionally repopulate the entire section. For this reason it is never written to disk and admin's are advised against modifying it in any way.

The configuration section contains the more traditional information like cluster options, lists of resources and indications of where they should be placed. The configuration section is the primary focus of this document.

The configuration section itself is divided into four parts:

- Configuration options (called *crm_config*)
- Nodes
- Resources
- Resource relationships (called *constraints*)

```
<cib generated="true" admin_epoch="0" epoch="0" num_updates="0" have-quorum="false">
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
```

An empty configuration

The Current State of the Cluster

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

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

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

```

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

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

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

```

Sample output from crm_mon

```

=====
Last updated: Fri Nov 23 15:26:14 2007
Current DC: sles-3 (2298606a-6a8c-499a-9d25-76242f7006ec)
3 Nodes configured.
5 Resources configured.
=====

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

```

Sample output from crm_mon -n

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

How Should the Configuration be Updated?

There are three basic rules for updating the cluster configuration:

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

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

The most powerful tool for modifying the configuration is the `cibadmin` command which talks to a running cluster. With `cibadmin`, the user can query, add, remove, update or replace any part of the configuration and all changes take effect immediately so there is no need to perform a reload-like operation.

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

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

Some of the better XML editors can make use of a Relax NG schema to help make sure any changes you make are valid. The schema describing the configuration can normally be found in `/usr/lib/heartbeat/pacemaker.rng` on most systems.

If you only wanted to modify the `resources` section, you could instead do

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

to avoid modifying any other part of the configuration.

Quickly Deleting Part of the Configuration

Identify the object you wish to delete. eg.

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

Next identify the resource's tag name and id (in this case `primitive` and `child_DoFencing`). Then simply execute:

```
cibadmin --delete --crm_xml '<primitive id="child_DoFencing"/>'
```

Updating the Configuration Without Using XML

Some common tasks can also be performed with one of the higher level tools that avoid the need to read or edit XML.

To enable stonith for example, one could run:

```
crm_attribute --attr-name stonith-enabled --attr-value true
```

Or to see if `somenode` is allowed to run resources, there is:

```
crm_standby --get-value --node-uname somenode
```

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

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

Making Configuration Changes in a Sandbox

Often it is desirable to preview the effects of a series of changes before updating the configuration atomically. For this purpose we have created `crm_shadow` which creates a "shadow" copy of the configuration and arranges for all the command line tools to use it.

To begin, simply invoke `crm_shadow` and give it the name of a configuration to create² and be sure to follow the simple on-screen instructions. *Failure to do so could result in you updating the cluster's active configuration!*

```
c001n01:~ # crm_shadow --create test
Setting up shadow instance
Type Ctrl-D to exit the crm_shadow shell
shadow[test]:
```

Creating a new sandbox

```
shadow[test] # crm_shadow --which
test
```

Checking which shadow copy is active

From this point on, all cluster commands will automatically use the shadow copy instead of talking to the cluster's active configuration.

```
shadow[test] # crm_failcount -G -r rsc_c001n01
name=fail-count-rsc_c001n01 value=0
shadow[test] # crm_standby -v on -n c001n02
shadow[test] # crm_standby -G -n c001n02
name=c001n02 scope=nodes value=on
shadow[test] # cibadmin --erase --force
shadow[test] # cibadmin --query
<cib cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0" crm_feature_set="3.0" have-
quorum="1" epoch="112" dc-uuid="c001n01" num_updates="1" cib-last-written="Fri Jun 27 12:17:10 2008">
<configuration>
  <crm_config/>
  <nodes/>
  <resources/>
  <constraints/>
</configuration>
```

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

```
<status/>
</cib>
```

Making changes to the shadow configuration

Once you have finished experimenting, you can either commit the changes, or discard them as shown below. Again, be sure to follow the on-screen instructions carefully.

```
shadow[test] # crm_shadow --delete test --force
Now type Ctrl-D to exit the crm_shadow shell
shadow[test] # exit
c001n01:~ # crm_shadow --which
No shadow instance provided
c001n01:~ # cibadmin -Q
<cib cib_feature_revision="1" validate-with="pacemaker-1.0" admin_epoch="0" crm_feature_set="3.0" have-
quorum="1" epoch="110" dc-uuid="c001n01" num_updates="551">
<configuration>
<crm_config>
<cluster_property_set id="cib-bootstrap-options">
<nvpair id="cib-bootstrap-1" name="stonith-enabled" value="1"/>
<nvpair id="cib-bootstrap-2" name="pe-input-series-max" value="30000"/>
```

Discarding changes and verifying the real configuration is intact

For a full list of `crm_shadow` options and command, invoke the command without any arguments.

Testing Your Configuration Changes

We saw previously how to make a series of changes to a "shadow" copy of the configuration. Before loading the changes back into the cluster (eg. `crm_shadow --commit mytest --force`), it is often advisable to simulate the effect of the changes with `ptest`. Eg.

```
ptest --live-check -VVVVV --save-graph tmp.graph --save-dotfile tmp.dot
```

The tool uses the same library as the live cluster to show what it would have done given the supplied input. Its output, in addition to a significant amount of logging, is stored in two files `tmp.graph` and `tmp.dot`, both are representations of the same thing -- the cluster's response to your changes. In the graph file is stored the complete transition, containing a list of all the actions, their parameters and their pre-requisites. Because the transition graph is not terribly easy to read, the tool also generates a Graphviz dot-file representing the same information.



An example transition graph as represented by Graphviz

Interpreting the Graphviz output

- Arrows indicate ordering dependencies
- Dashed-arrows indicate dependencies that are not present in the transition graph
- Actions with a dashed border of any color do not form part of the transition graph
- Actions with a green border form part of the transition graph
- Actions with a red border are ones the cluster would like to execute but are unrunnable
- Actions with a blue border are ones the cluster does not feel need to be executed
- Actions with orange text are pseudo/pretend actions that the cluster uses to simplify the graph
- Actions with black text are sent to the LRM
- Resource actions have text of the form `{rsc}_{action}_{interval} {node}`
- Any action depending on an action with a red border will not be able to execute.
- Loops are **really** bad. Please report them to the development team.

In the above example, it appears that a new node, `node2`, has come online and that the cluster is checking to make sure `rsc1`, `rsc2` and `rsc3` are not already running there (Indicated by the `*_monitor_0` entries). Once it did that, and assuming the resources were not active there, it would have liked to stop `rsc1` and `rsc2` on `node1` and move them to `node2`. However, there appears to be some problem and the cluster cannot or is not permitted to perform the stop actions which implies it also cannot perform the start actions. For some reason the cluster does not want to start `rsc3` anywhere.

For information on the options supported by `ptest`, use `ptest --help`



Another, slightly more complex, transition graph that you're not expected to be able to read

Do I Need to Update the Configuration on all Cluster Nodes?

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

To reduce bandwidth, the cluster only broadcasts the incremental updates that result from your changes and uses md5 sums to ensure that each copy is completely consistent.

Cluster Options

Special Options

The reason for these fields to be placed at the top level instead of with the rest of cluster options 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.

Determining Which Configuration to Use

When a node joins the cluster, the cluster will perform a check to see who has the *best* configuration based on the fields below. It then asks the node with the highest (admin_epoch, epoch, num_updates) tuple to replace the configuration on all the nodes - which makes setting them and setting them correctly very important.

| Field | Description |
|-------------|---|
| admin_epoch | Never modified by the cluster. Use this to make the configurations on any inactive nodes obsolete. Never set this value to zero, in such cases the cluster cannot tell the difference between your configuration and the "empty" one used when nothing is found on disk. |
| epoch | Incremented every time the configuration is updated (usually by the admin) |
| num_updates | Incremented every time the configuration or status is updated (usually by the cluster) |

Other Fields

| Field | Description |
|---------------|--|
| validate-with | Determines the type of validation being done on the configuration. If set to "none", the cluster will not verify that updates conform to the DTD (nor reject ones that don't). This option can be useful when operating a mixed version cluster during an upgrade. |

Fields Maintained by the Cluster

| Field | Description |
|------------------|--|
| crm-debug-origin | Indicates where the last update came from. Informational purposes only. |
| cib-last-written | Indicates when the configuration was last written to disk. Informational purposes only. |
| dc-uuid | Indicates which cluster node is the current leader. Used by the cluster when placing resources and determining the order of some events. |
| have-quorum | Indicates if the cluster has quorum. If false, this may mean that the cluster cannot start resources or fence other nodes. See no-quorum-policy below. |

Note that although these fields can be written to by the admin, in most cases the cluster will overwrite any values specified by the admin with the "correct" ones. To change the admin_epoch, for example, one would use:

```
cibadmin --modify --crm_xml '<cib admin_epoch="42"/>'
```

A complete set of fields will look something like this:

```
<cib have-quorum="true" validate-with="pacemaker-1.0" admin_epoch="1" epoch="12" num_updates="65"
dc-uuid="ea7d39f4-3b94-4cfa-ba7a-952956daabee" >
```

An example of the fields set for a cib object

Cluster Options

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

They are grouped into sets and, in advanced configurations, there may be more than one.³ For now we will describe the simple case where each option is present at most once.

Available Cluster Options

| Option | Default | Description |
|------------------------|---------|--|
| batch-limit | 30 | The number of jobs that the TE is allowed to execute in parallel. The "correct" value will depend on the speed and load of your network and cluster nodes. |
| no-quorum-policy | stop | What to do when the cluster does not have quorum. Allowed values: stop, freeze, ignore, suicide. |
| symmetric-cluster | TRUE | Can all resources run on any node by default? |
| stonith-enabled | FALSE | Should failed nodes and nodes with resources that can't be stopped be shot? If you value your data, set up a STONITH device and enable this. |
| stonith-action | reboot | Action to send to STONITH device. Allowed values: reboot, poweroff. |
| cluster-delay | 60s | Round trip delay over the network (excluding action execution). The "correct" value will depend on the speed and load of your network and cluster nodes. |
| stop-orphan-resources | TRUE | Should deleted resources be stopped |
| stop-orphan-actions | TRUE | Should deleted actions be cancelled |
| start-failure-is-fatal | TRUE | When set to FALSE, the cluster will instead use the resource's failcount and value for resource-failure-stickiness |
| pe-error-series-max | -1 | The number of PE inputs resulting in ERRORS to save. Used when reporting problems. |
| pe-warn-series-max | -1 | The number of PE inputs resulting in WARNINGS to save. Used when reporting problems. |
| pe-input-series-max | -1 | The number of "normal" PE inputs to save. Used when reporting problems. |

You can always obtain an up-to-date list of cluster options, including their default values by running the **pengine metadata** command.

³ This will be described later in the section on [rules](#) where we will show how to have the cluster use different sets of options during working hours (when downtime is usually to be avoided at all costs) than it does during the weekends (when resources can be moved to their preferred hosts without bothering end users)

Querying and Setting Cluster Options

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

```
crm_attribute --attr-name cluster-delay --get-value
```

which is more simply written as

```
crm_attribute --get-value -n cluster-delay
```

If a value is found, the you'll see a result such as this

```
sles-1:~ # crm_attribute --get-value -n cluster-delay
name=cluster-delay value=60s
```

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

```
sles-1:~ # crm_attribute --get-value -n clusta-deway
name=clusta-deway value=(null)
Error performing operation: The object/attribute does not exist
```

To use a different value, eg. 30s, simply run:

```
crm_attribute --attr-name cluster-delay --attr-value 30s
```

To go back to the cluster's default value, you can then delete the value with:

```
crm_attribute --attr-name cluster-delay --delete-attr
```

When Options are Listed More Than Once

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

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

Example of deleting an option that is listed twice

In such cases follow the on-screen instructions to perform the requested action. To determine which value is currently being used by the cluster, please refer to the the section on [rules](#).

Cluster Nodes

Defining a Cluster Node

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

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

An example of a cluster node

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

Describing a Cluster Node

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

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

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

```
crm_attribute --type nodes --node-uname sles-1 --attr-name kernel --attr-value `uname -r`
```

```
<node uname="sles-1" type="normal" id="1186dc9a-324d-425a-966e-d757e693dc86">
  <instance_attributes id="nodes-1186dc9a-324d-425a-966e-d757e693dc86">
    <nvpair id="kernel-1186dc9a-324d-425a-966e-d757e693dc86" name="kernel" value="2.6.16.46-0.4-default"/>
  </instance_attributes>
</node>
```

The result of using `crm_attribute` to specify which kernel `sles-1` is running

A simpler way to determine the current value of an attribute is to use `crm_attribute` command again:

```
crm_attribute --type nodes --node-uname sles-1 --attr-name kernel --get-value
```

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

Adding a New Cluster Node

OpenAIS

Adding a new is as simple as installing OpenAIS and Pacemaker, and copying `/etc/ais/openais.conf` and `/etc/ais/authkey` (if it exists) from an existing node. You may need to modify the `mcastaddr` option to match the new node's IP address.

If a log message containing "Invalid digest" appears from OpenAIS, the keys are not consistent between the machines.

Heartbeat

Provided you specified `autojoin any` in `ha.cf`, adding a new is as simple as installing heartbeat and copying `ha.cf` and `authkeys` from an existing node.

If not, then after setting up `ha.cf` and `authkeys`, you must use the `hb_addnode` command before starting the new node.

Removing a Cluster Node

OpenAIS

TBA

Heartbeat

Because the messaging and membership layers are the authoritative source for cluster nodes, deleting them from the CIB is not a reliable solution. First one must arrange for heartbeat to forget about the node (`sles-1` in the example below). To do this, shut down heartbeat on the node and then, from one of the remaining active cluster nodes, run:

```
hb_delnode sles-1
```

Only then is it safe to delete the node from the CIB with:

```
cibadmin --delete --obj_type nodes --crm_xml '<node uname="sles-1"/>'
cibadmin --delete --obj_type status --crm_xml '<node_status uname="sles-1"/>'
```

Replacing a Cluster Node

OpenAIS

The five-step guide to replacing an existing cluster node:

1. Make sure the old node is completely stopped
2. Give the new machine the same hostname and IP address as the old one
3. Install the cluster software:)
- 4.
5. Copy `/etc/ais/openais.conf` and `/etc/ais/authkey` (if it exists) to the new node
6. Start the new cluster node

If a log message containing "Invalid digest" appears from OpenAIS, the keys are not consistent between the machines.

Heartbeat

The seven-step guide to replacing an existing cluster node:

1. Make sure the old node is completely stopped
2. Give the new machine the same hostname as the old one

3. Go to an active cluster node and look up the UUID for the old node in `/var/lib/heartbeat/hostcache`
4. Install the cluster software
5. Copy `ha.cf` and `authkeys` to the new node
6. On the new node, populate it's UUID using `crm_uuid -w` and the UUID from step 2
7. Start the new cluster node

Cluster Resources

What is a Cluster Resource

The role of a resource agent is to abstract the service it provides and present a consistent view to the cluster, which allows the cluster to be agnostic about the resources it manages. The cluster doesn't need to understand how the resource works because it relies on the resource agent to do the right thing when given a start, stop or monitor command.

For this reason it is crucial that resource agents are well tested.

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

Supported Resource Classes

There are three basic classes of agents supported by Pacemaker. In order of encouraged usage they are:

Open Cluster Framework

The OCF Spec (as it relates to resource agents can be found at: <http://www.opencf.org/cgi-bin/viewcvs.cgi/specs/ra/resource-agent-api.txt?rev=HEAD>⁴ and is basically an extension of the Linux Standard Base conventions for init scripts to

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

OCF specs have strict definitions of what exit codes actions must return⁵. The cluster follows these specifications exactly, and exiting with the wrong exit code will cause the cluster to behave in ways you will likely find puzzling and annoying. In particular, the cluster needs to distinguish a completely stopped resource from one which is in some erroneous and indeterminate state.

Parameters are passed to the script as environment variables, with the special prefix `OCF_RESKEY_`. So, if you need to be given a parameter which the user thinks of as `ip` it will be passed to the script as `OCF_RESKEY_ip`. The number and purpose of the parameters is completely arbitrary, however your script should advertise any that it supports using the `meta-data` command.

For more information, see <http://wiki.linux-ha.org/OCFResourceAgent> and [What Do I need to Know When Writing an OCF Resource Agent](#) in the FAQ section.

Linux Standard Base

LSB resource agents are those found in `/etc/init.d`. Generally they are provided by the OS/distribution and in order to be used with the cluster, must conform to the LSB Spec.

The LSB Spec (as it relates to init scripts) can be found at: http://refspecs.linux-foundation.org/LSB_3.0.0/LSB-Core-generic/LSB-Core-generic/iniscrptact.html

⁴ Note: The Pacemaker implementation has been somewhat extended from the OCF Specs, but none of those changes are incompatible with the original OCF specification

⁵ Included with the cluster is the `ocf-tester` script which can be useful in this regard.

Many distributions claim LSB compliance but ship with broken init scripts. To see if your init script is LSB-compatible, see the FAQ entry [How Can I Tell if an Init Script is LSB Compatible](#). The most common problems are:

- Not implementing the status operation at all
- Not observing the correct exit status codes for start/stop/status actions
- Starting a started resource returns an error (this violates the LSB spec)
- Stopping a stopped resource returns an error (this violates the LSB spec)

Legacy Heartbeat

Version 1 of Heartbeat came with its own style of resource agents and it is highly likely that many people have written their own agents based on its conventions. To enable administrators to continue to use these agents, they are supported by the new cluster manager.

For more information, see: <http://wiki.linux-ha.org/HeartbeatResourceAgent>

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

There is also an additional class, STONITH, which is used exclusively for fencing related resources. This is discussed later in the section on [fencing](#).

Properties

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

| Field | Description |
|----------|--|
| id | Your name for the resource |
| class | Allowed values: heartbeat, lsb, ocf, stonith |
| type | The name of the Resource Agent you wish to use. eg. IPaddr or Filesystem |
| provider | The OCF spec allows multiple vendors to supply the same ResourceAgent. To use the OCF resource agents supplied with Heartbeat, you should specify heartbeat here. |

Properties of a primitive resource

Resource definitions can be queried with the `crm_resource` tool. For example

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

might produce

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

An example LSB resource⁶

or, for an OCF resource:

```
<primitive id="Public-IP" class="ocf" type="IPaddr" provider="heartbeat">
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
  </instance_attributes>
```

⁶ LSB resources do not allow any parameters

```
</primitive>
```

An example OCF resource

or, finally for the equivalent legacy Heartbeat resource:

```
<primitive id="Public-IP-legacy" class="heartbeat" type="IPAddr">
  <instance_attributes id="params-public-ip-legacy">
    <nvpair id="public-ip-addr-legacy" name="1" value="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

An example Heartbeat resource⁷

Options

Options are used by the cluster to decide how your resource should behave and can be easily set using the `--meta` option of the `crm_resource` command.

| Field | Description |
|---------------------|---|
| priority | If not all resources can be active, the cluster will stop lower priority resources in order to keep higher priority ones active. |
| target-role | What state should the cluster attempt to keep this resource in? Allowed values: Stopped, Started |
| is-managed | Is the cluster allowed to start and stop the resource? Allowed values: true , false |
| resource-stickiness | How much does the resource prefer to stay where it is? Defaults to the value of default-resource-stickiness |
| migration-threshold | How many failures should occur for this resource on a node before making the node ineligible to host this resource. Default: none |
| multiple-active | What should the cluster do if it ever finds the resource active on more than one node. Allowed values: block (mark the resource as unmanaged), stop_only, stop_start |
| failure-timeout | How many seconds to wait before acting as if the failure had not occurred (and potentially allowing the resource back to the node on which it failed. Default: never |

Available options for a primitive resource

If you performed the following commands on the above resource

```
crm_resource --meta --resource Email --set-parameter priority --property-value 100
```

```
crm_resource --meta --resource Email --set-parameter multiple-active --property-value block
```

the resulting resource definition would be

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

⁷ Heartbeat resources take only ordered and unnamed parameters. The supplied name therefor indicates the order in which they are passed to the script. Only single digit values are allowed.

```
<nvpair id="email-active" name="multiple-active" value="block"/>
</meta_attributes>
</primitive>
```

An example LSB resource with cluster options

Setting Global Defaults for Resource Options

To set a default value for a resource option, simply add it to the `rsc_defaults` section with `crm_attribute`.

Thus,

```
crm_attribute --type rsc_defaults --attr-name is-managed --attr-value 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 and had *is-managed* set to true).

Instance Attributes

The scripts of some resource classes (LSB not being one of 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 instance

```
crm_resource --resource Public-IP --set-parameter ip --property-value 1.2.3.4
```

would create an entry in the resource like this

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

An example OCF resource with instance attributes

For an OCF resource, the result would be an environment variable called `OCF_RESKEY_ip` with a value of `1.2.3.4`

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

```
export OCF_ROOT=/usr/lib/ocf; $OCF_ROOT/resource.d/heartbeat/IPAddr2 meta-data
```

Displaying the metadata for the IPAddr resource agent

Resource Operations

Monitoring Resources for Failure

By default, the cluster will not ensure your resources are still healthy. To instruct the cluster to do this, you need to add a monitor operation to the resource's definition.

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <operations>
    <op id="public-ip-check" name="monitor" interval="60s"/>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

An OCF resource with a recurring health check

| Field | Description |
|----------|--|
| id | Your name for the action. Must be unique. |
| name | The action to perform. Common values: monitor, start, stop |
| interval | How frequently (in seconds) to perform the operation. Default value: 0 |
| timeout | How long to wait before declaring the action has failed. |
| requires | What conditions need to be satisfied before this action occurs. Allowed values: nothing, quorum, fencing. The default depends on whether fencing is enabled and if the resource's class is <i>stonith</i> |
| on-fail | The action to take if this action ever fails. Allowed values: ignore, block, stop, restart, fence, standby. ignore ::= Pretend the resource did not fail block ::= Don't perform any further operations on the resource stop ::= Stop the resource and do not start it elsewhere restart ::= Stop the resource and start it again (possibly on a different node) fence ::= STONITH the node on which the resource failed standby ::= Move all resources away from the node on which the resource failed |
| enabled | If false, the operation is treated as if it does not exist. Allowed values: true , false |

Valid fields for an operation

Setting Global Defaults for Operations

To set a default value for a operation option, simply add it to the *op_defaults* section with *crm_attribute*.

Thus,

```
crm_attribute --type op_defaults --attr-name timeout --attr-value 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 instead (for that operation only).

When Resources Take a Long Time to Start/Stop

There are a number of implicit operations that the cluster will always perform - start, stop and a non-recurring monitor operation (used at startup to check the resource isn't already active). If one of these is taking too long, then you can create an entry for them and simply specify a new value.

```
<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="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

An OCF resource with custom timeouts for its implicit actions

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. Note that it is up to each resource agent to look for the parameter and decide how to use it.

```
<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="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

An OCF resource with two recurring health checks performing different levels of checks

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 `disabled="true"` to the operation's definition.

```
<primitive id="Public-IP" class="ocf" type="IPAddr" provider="heartbeat">
  <operations>
    <op id="public-ip-check" name="monitor" interval="60s" disabled="true"/>
  </operations>
  <instance_attributes id="params-public-ip">
    <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
  </instance_attributes>
</primitive>
```

Example of an OCF resource with a disabled health check

This can be achieved from the command-line by executing

```
cibadmin -M -X '<op id="public-ip-check" disabled="true"/>'
```

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

```
cibadmin -M -X '<op id="public-ip-check" disabled="false"/>'
```

Resource Constraints

Scores

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

Scores are calculated on a per-resource basis and any node with a negative score for a resource can't run that resource. After calculating the scores for a resource, the cluster then chooses the node with the highest one.

Infinity Math

INFINITY is currently defined as 1,000,000 and addition/subtraction with it follows the following 3 basic rules:

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

Deciding Which Nodes a Resource Can Run On

There are two alternative strategies for specifying which nodes a resources can run on. One way is to say that by default they can run anywhere and then create location constraints for nodes that are not allowed. The other option is to have nodes "opt-in"... to start with nothing able to run anywhere and selectively enable allowed nodes.

Options

| Field | Description |
|-------|--|
| id | A unique name for the constraint |
| rsc | A resource name |
| node | A node's uname |
| score | Positive values indicate the resource can run on this node. Negative values indicate the resource can not run on this node. Values of +/- INFINITY change "can" to "must". |

Asymmetrical "Opt-In" Clusters

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

```
crm_attribute --attr-name symmetric-cluster --attr-value false
```

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

```
<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>
```

Example set of opt-in location constraints

Symmetrical “Opt-Out” Clusters

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

```
crm_attribute --attr-name symmetric-cluster --attr-value true
```

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

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

Example set of opt-out location constraints

Whether you should choose opt-in or opt-out depends both 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.

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 what was intended.

```
<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>
```

Example of two resources that prefer two nodes equally

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 the Order Resources Should Start/Stop In

The way to specify the order in which resources should start is by creating *rsc_order* constraints.

| Field | Description |
|-------------|--|
| id | A unique name for the constraint |
| first | The name of a resource that must be started before the <i>then</i> resource is allowed to. |
| then | The name of a resource. This resource will start after the <i>first</i> resource. |
| score | If greater than zero, the constraint is mandatory. Otherwise it is only a suggestion. Default value: INFINITY |
| symmetrical | If true, which is the default, stop the resources in the reverse order. Default value: true |

Mandatory Ordering

When the *then* resource cannot run without the *first* resource being active, one should use mandatory constraints. To specify a constraint is mandatory, use a scores greater than zero. This will ensure that the *then* resource will react when the *first* resource changes state.

- If the *first* resource was running and is stopped, the *then* resource will also be stopped (if it is running)
- If the *first* resource was not running and cannot be started, the *then* resource will be stopped (if it is running)
- If the *first* resource is (re)started while the *then* resource is running, the *then* resource will be stopped and restarted

Advisory Ordering

On the other-hand, when `score="0"` is specified for a constraint, the constraint is considered optional and only has an effect when both resources are stopping and or starting. Any change in state by the *first* resource will have no effect on the *then* resource.

```
<constraints>
  <rsc_order id="order-1" first="Database" then="Webserver" />
  <rsc_order id="order-2" first="IP" then="Webserver" score="0"/>
</constraints>
```

Example of an optional and mandatory ordering constraint

Some additional information on ordering constraints can be found in the document [Ordering Explained](#)

Placing Resources Relative to other Resources

When the location of one resource depends on the location of another one, we call this colocation.

There is an important side-effect of creating a colocation constraint between two resources, that it affects the order in which resources are assigned to a node. If you think about it, its somewhat obvious. You can't place A relative to B unless you know where B is⁸. So when you are creating colocation constraints, it is important to consider whether you should colocate A with B or B with A.

Another thing to keep in mind is that, assuming A is collocated with B, the cluster will also take into account A's preferences when deciding which node to choose for B. For a detailed look at exactly how this occurs, see the [Colocation Explained](#) document.

Options

| Field | Description |
|----------|--|
| id | A unique name for the constraint |
| rsc | The colocation source. If the constraint cannot be satisfied, the cluster may decide not to allow the resource to run at all. |
| with-rsc | The colocation target. The cluster will decide where to put this resource first and then decide where to put the resource in the <i>rsc</i> field |
| score | Positive values indicate the resource should run on the same node. Negative values indicate the resources should not run on the same node. Values of +/- INFINITY change "should" to "must". |

⁸ 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.

Mandatory Placement

Mandatory placement occurs any time 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 `resource1` to always run on the same machine as `resource2`, you would add the following constraint:

```
<rsc_colocation id="colocate" rsc="resource1" with-rsc="resource2" score="INFINITY"/>
```

An example colocation constraint

Remember, because `INFINITY` was used, if `resource2` can't run on any of the cluster nodes (for whatever reason) then `resource1` will not be allowed to run.

Alternatively, you may want the opposite... that `resource1` cannot run on the same machine as `resource2`. In this case use `score="-INFINITY"`

```
<rsc_colocation id="anti-colocate" rsc="resource1" with-rsc="resource2" score="-INFINITY"/>
```

An example anti-colocation constraint

Again, by specifying `-INFINITY`, the constraint is binding. So if the only place left to run is where `resource2` already is, then `resource1` may not run anywhere.

Advisory Placement

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

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

```
<rsc_colocation id="colocate-maybe" rsc="resource1" with-rsc="resource2" score="500"/>
```

An example advisory-only colocation constraint

Ordering Sets of Resources

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

```
<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>
```

A chain of ordered resources



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

To simplify this situation, there is an alternate format for ordering constraints

```

<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>

```

A chain of ordered resources expressed as a set

NOTE: Resource sets have the **opposite** ordering semantics to groups.

```

<group id="dummy">
  <primitive id="D" .../>
  <primitive id="C" .../>
  <primitive id="B" .../>
  <primitive id="A" .../>
</group>

```

A group resource with the equivalent ordering rules

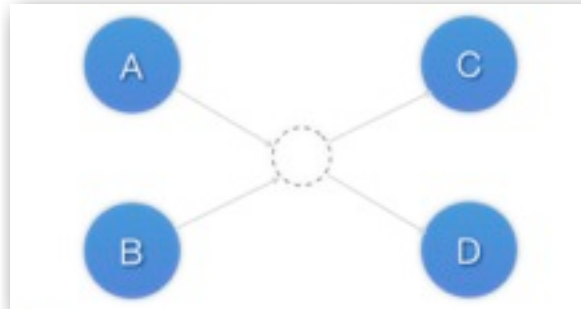
While the set-based format is not less verbose, it is significantly easier to get right and maintain. It can also be expanded to allow ordered sets of (un)ordered resources. In the example below, *rscA* and *rscB* can both start in parallel, as can *rscC* and *rscD*, however *rscC* and *rscD* can only start once **both *rscA* and *rscB*** are active.

```

<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>

```

Ordered sets of unordered resources

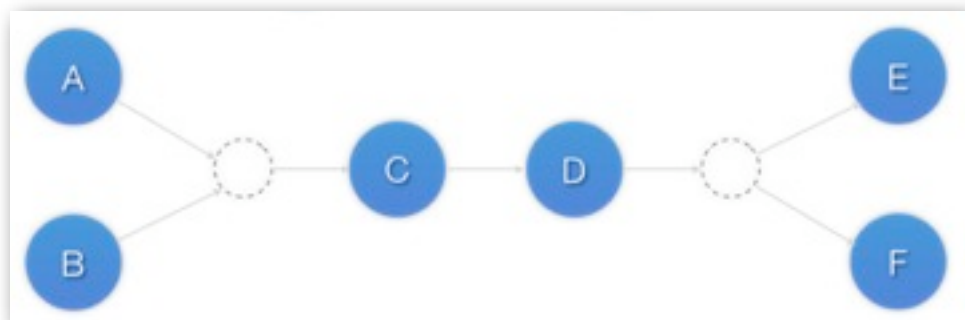


Visual representation of the start order for two ordered sets of unordered resources

Of course either 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.

```
<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>
```

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



Visual representation of the start order for the three sets defined above

Collocating Sets of Resources

Another common situation is for an administrator to create a set of collocated resources. Previously this possible either by defining a resource group (See [Groups - A Syntactic Shortcut](#)) which could not always accurately express the design; or by defining each relationship as an individual constraint, causing a constraint explosion as the number of resources and combinations grew.

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

A chain of collocated resources

To make things easier, we allow an alternate form of collocation constraints using *resource_sets*. Just like the expanded version, a resource that can't be active also prevents any resource that must be collocated with it from being active. For example if B was not able to run, then both C (and by inference D) must also remain stopped.

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

The equivalent collocation chain expressed using resource_sets

NOTE: Resource sets have the **same** collocation semantics as groups.

```
<group id="dummy">
  <primitive id="A" .../>
  <primitive id="B" .../>
  <primitive id="C" .../>
  <primitive id="D" .../>
</group>
```

A group resource with the equivalent collocation rules

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

```
<constraints>
  <rsc_collocation id="coloc-1" score="INFINITY" >
    <resource_set id="collocated-set-1" sequential="false">
      <resource_ref id="A"/>
      <resource_ref id="B"/>
    </resource_set>
  </rsc_collocation>
</constraints>
```

```

    <resource_ref id="C"/>
  </resource_set>
  <resource_set id="collocated-set-2" sequential="true">
    <resource_ref id="D"/>
  </resource_set>
</rsc_colocation>
</constraints>

```

Using colocation sets to specify a common peer.

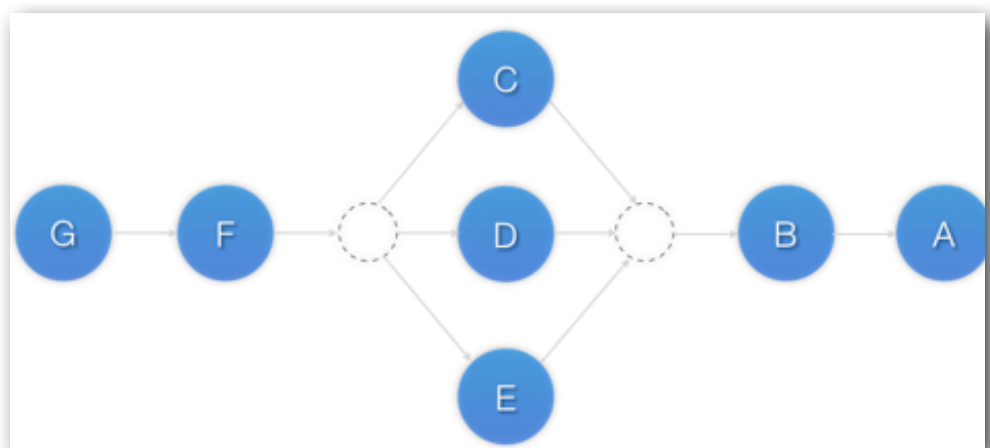
Of course there is no limit to the number and size of the sets used. The only thing that matters is that in order for any member of set N to be active, all the members of set N+1 must also be active (and naturally on the same node), and that if a set has *sequential="true"*, then in order for member M to be active, member M+1 must also be active. You can even specify the role in which the members of a set must be in using the set's *role* attribute.

```

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

```

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



Visual representation of a colocation chain where the members of the middle set have no interdependencies

Rules

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

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

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

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

Properties of a rule object

Node Attribute Expressions

Expression objects are used to control a resource based on the attributes defined by a node or nodes. In addition to any attributes added by the administrator, each node has a built-in node attribute called *#uname* that can also be used.

| Field | Description |
|-----------|--|
| value | User supplied value for comparison |
| attribute | The node attribute to test |
| operation | The comparison to perform. Allowed values: lt, gt, lte, gte, eq, ne, defined, not_defined |
| type | Determines how the value(s) should be tested. Allowed values: integer, string , version |

Properties of expression objects

Time/Date Based Expressions

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

| Field | Description |
|-----------|---|
| start | A date/time conforming to the ISO8601 specification. |
| end | A date/time conforming to the ISO8601 specification. Can be inferred by supplying a value for <i>start</i> and a <i>duration</i> . |
| operation | Compares the current date/time with the <i>start</i> and/or <i>end</i> date, depending on the context. Allowed values: gt, lt, in-range, date-spec. |

Properties of date_expression objects

The *in-range* operation evaluates to true if *start* < current date/time < *end*. If either start or end is omitted, then that part of the comparison is not performed.

The *date-spec* operation performs a cron-like comparison between the contents of date_spec and now. If values for *start* and/or *end* are included, the current date/time must also be within that range.

The *lt* operation evaluates to true if current date/time < *end*.

The *gt* operation evaluates to true if *start* < current date/time.

NOTE: Because the comparisons (except for date_spec) include the time, the eq, neq, gte and lte operators have not been implemented.

Date Specifications

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

For example, *monthdays*="1" matches the first day of every month and *hours*="09-17" matches the hours between 9am and 5pm inclusive). However at this time one cannot specify *weekdays*="1,2" or *weekdays*="1-2,5-6" since they contain multiple ranges. Depending on demand, this may be implemented in a future release.

| Field | Description |
|-----------|---|
| id | A unique name for the date |
| hours | Allowed values: 0-23 |
| monthdays | Allowed values: 0-31 (depending on current month and year) |
| weekdays | Allowed values: 1-7 (1=Monday, 7=Sunday) |
| yeardays | Allowed values: 1-366 (depending on the current year) |
| months | Allowed values: 1-12 |
| weeks | Allowed values: 1-53 (depending on weekyear) |
| years | Year according the Gregorian calendar |
| weekyears | May differ from Gregorian years. Eg. "2005-001 Ordinal" is also "2005-01-01 Gregorian" is also "2004-W53-6 Weekly" |
| moon | Allowed values: 0..7 (0 is new, 4 is full moon). Seriously, you can use this. This was implemented to demonstrate the ease with which new comparisons could be added. |

Properties of date_spec objects

Durations

Durations are used to calculate a value for *end* when one is not supplied to *in_range* operations. They contain the same fields as *date_spec* objects but without the limitations (ie. you can have a duration of 19 days). Like *date_specs*, any field not supplied is ignored.

Sample Time Based Expressions

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

True if now is any time in the year 2005

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

Equivalent expression.

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

9am-5pm, Mon-Friday

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

9am-5pm, Mon-Friday, or all day saturday

```

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

```

9am-5pm or 9pm-12pm, Mon-Friday

```

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

```

Mondays in March 2005

NOTE: Because no time is specified, 00:00:00 is implied.

This means that the range includes all of 2005-03-01 but none of 2005-04-01.

You may wish to write end="2005-03-31T23:59:59" to avoid confusion.

```

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

```

A full moon on Friday the 13th

Using Rules to Determine Resource Location

If the constraint's outer-most rule evaluates to false, the cluster treats the constraint as if it was not there. When the rule evaluates to true, the node's preference for running the resource is updated with the score associated with the rule.

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

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

Prevent myApacheRsc from running on c001n03

This constraint can be more verbosely written as:

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

Prevent myApacheRsc from running on c001n03 - expanded version

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

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

```
<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>
```

A sample nodes section for use with score-attribute

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

```
<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 differently, according to its value for the named node attribute. Thus in the previous example, if a rule used *score-attribute="cpu_mips"*, c001n01 would have its preference to run the resource increased by 1234 whereas c001n02 would have its preference increased by 5678.

Using Rules to Control Resource Options

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

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

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

```
<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>
```

Defining different resource options based on the node name

The order in which *instance_attributes* objects are evaluated is determined by their *score* (highest to lowest). If not supplied, *score* defaults to zero and objects with an equal score are processed in listed order. If the *instance_attributes* object does not have a rule or has a rule that evaluates to true, then for any parameter the resource does not yet have a value for, the resource will use the parameter values defined by the *instance_attributes* object.

Using Rules to Control Cluster Options

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

The difference is that because they are cluster options, one cannot (or should not because they won't work) use attribute based expressions. The following example illustrates how to set a different *resource-stickiness* value during and outside of 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.

```
<rsc_defaults>
  <meta_attributes id="core-hours" score="2">
    <rule id="core-hour-rule">
      <date_expression id="9to5_Mon2Fri" operation="date_spec">
        <date_spec hours="9-17" days="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"/>
  </instance_attributes>
</primitive>
```

Set resource-stickiness=INFINITY Mon-Fri between 9am and 5pm, and resource-stickiness=0 all other times

Ensuring Time Based Rules Take Effect

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

If you rely on time based rules, it is essential that you set the *cluster-recheck-interval* option. This tells the cluster to periodically recalculate the ideal state of the cluster. For example, if you set *cluster-recheck-interval=5m*, then sometime between 9:00 and 9:05 the cluster would notice that it needs to start resource X, and between 17:00 and 17:05 it would realize it needed to be stopped.

Note that the timing of the actual start and stop actions depends on what else needs to be performed first.

Advanced Configuration

Connecting to the Cluster Configuration from a Remote Machine

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

| Environment Variable | Description |
|----------------------|---|
| CIB_user | The user to connect as. Needs to be part of the hacluster group on the target host. Defaults to \$USER |
| CIB_password | The user's password. Read from the command line if unset |
| CIB_server | The host to contact. Defaults to localhost . |
| CIB_port | The port on which to contact the server. Required. |

Variables used to connect to remote instances of the CIB

So if c001n01 is an active cluster node and is listening on 1234 for connections, and someguy is a member of the hacluster group. Then the following would prompt for someguy's password and return the cluster's current configuration:

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

For security reasons, the cluster does not listen remote connections by default. If you wish to allow remote access, you need to set the *remote-tls-port* (encrypted) or *remote-open-port* (unencrypted) top-level options (ie. those kept in the *cib* tag, like *num_updates* and *epoch*).

| Field | Description |
|-------------------|--|
| remote-tls-port | Listen for encrypted remote connections on this port. Default: none |
| remote-clear-port | Listen for plaintext remote connections on this port. Default: none |

Extra top-level CIB options for remote access

Specifying When Recurring Actions are Performed

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

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

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

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 2009 and every monday after that you would add:

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

Moving Resources

Manual Intervention

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

In the case where everything needs to move, since everything eventually comes down to a score, you could create constraints for every resource you have preventing it from running on that node. While the configuration can seem convoluted at times, not even we would require this of administrators.

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

```
crm_standby --get-value
```

A value of *true* indicates that the node is NOT able to host any resources and a value of *false* indicates that it CAN. You can also check the status of other nodes in the cluster by specifying the `--node-uname` option. Eg.

```
crm_standby --get-value --node-uname sles-2
```

To change the current node's standby status, use `--attr-value` instead of `--get-value`. Eg.

```
crm_standby --attr-value
```

Again, you can change another host's value by supplying a host name with `--node-uname`.

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

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

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

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

It is important to note that subsequent invocations of `crm_resource -M` are not cumulative. So if you ran:

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

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

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

To allow the resource to move back again, use:

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

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

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

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

```
crm_resource -M -r Email
```

Which will instead create a negative constraint. Eg.

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

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

In some cases, such as when resource stickiness is set to `INFINITY`, it is possible that you will end up with the problem described in [What if Two Nodes Have the Same Score](#). The tool can detect some of these cases and deals with them by also creating both a positive and negative constraint. Eg.

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

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

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

Moving Resources Due to Failure

New in 1.0 is the concept of a migration threshold⁹. Simply define `migration-threshold=N` for a resource and it will migrate to a new node after N failures. There is no threshold defined by default. To determine the resource's current failure status and limits, use `crm_mon --failcounts`

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

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

There are two exceptions to the migration threshold concept and occur when a resource either fails to start or fails to stop. Start failures cause the failcount to be set to `INFINITY` and thus always cause the resource to move immediately.

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

Note: Please read the section on [Ensuring Time Based Rules Take Effect](#) before enabling this option.

⁹ The naming of this option was unfortunate as it is easily confused with true 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.

Moving Resources Due to Connectivity Changes

Setting up the cluster to move resources when external connectivity is lost, is a two-step process.

Tell Pacemaker to monitor connectivity

To do this, you need to add a `pingd` resource to the cluster. `pingd` is a small daemon that sends special "ping" packets to a list of machines (specified by DNS hostname or IPv4/ IPv6 address) and uses the results to maintain a node attribute also called `pingd`.

NOTE: Older versions of Heartbeat required users to add ping nodes to `ha.cf` - this is no longer required.

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

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

Standard pingd options

```
<clone id="pingd-clone">
  <primitive id="pingd" provider="heartbeat" class="ocf" type="pingd">
    <instance_attributes id="pingd-attribs">
      <nvpair id="pingd-dampen" name="dampen" value="5s"/>
      <nvpair id="pingd-multiplier" name="multiplier" value="1000"/>
      <nvpair id="pingd-hosts" name="host_list" value="my.gateway.com www.bigcorp.com"/>
    </instance_attributes>
  </primitive>
</clone>
```

An example pingd cluster resource

Tell Pacemaker how to interpret the connectivity data

NOTE: Before reading the following, please make sure you have read and understood the [Rules](#) section above.

There are a number of ways to use the connectivity data provided by Heartbeat. The most common setup is for people to have a single ping node and want to prevent the cluster from running a resource on any unconnected node.

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

Don't run on unconnected nodes

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

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

Run only on nodes connected to 3 or more ping nodes (assumes multiplier is set to 1000)

or instead you can tell the cluster only to prefer nodes with the most connectivity.

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

Prefer the node with the most connected ping nodes

It is perhaps easier to think of this in terms of the simple constraints that the cluster translates it into. For example, if **sles-1** is connected to all 5 ping nodes but **sles-2** is only connected to 2, then it would be as if you instead had the following constraints in your configuration:

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

How the cluster translates the pingd constraint

The advantage being that you don't have to manually update them whenever your network connectivity changes.

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

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

A more complex example of choosing a location based on connectivity

Resource Migration

Some resources, such as Xen virtual guests, are able to move to another location without lose of state. We call this resource migration and is different from the normal practice of stopping the resource on the first machine and starting it elsewhere.

Not all resources are able to migrate, see the Migration Checklist below, and those that can wont 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
- 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 new 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 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 to implement migration for your service, as long as it works.

Migration Checklist

1. The resource may not be a clone.
2. The resource must use an OCF style agent.
3. The resource must not be in a failed or degraded state.
4. The resource must not, directly or indirectly, depend on any *primitive* or *group* resources.
5. The resources must support two new actions: *migrate_to* and *migrate_from* and advertise them in its metadata.
6. The resource must have the *allow-migrate* meta-attribute set to *true* (not the default).

If the resource depends on a clone, and at the time the resource needs to be move, the clone has instances that are stopping and instances that are starting, then the resource will be moved in the traditional manner. The Policy Engine is not yet able to model this situation correctly and so takes the safe (yet less optimal) path.

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="pingd-prefer-rule" score-attribute="pingd" >
    <expression id="pingd-prefer" attribute="pingd" operation="defined"/>
  </rule>
</rsc_location>
```

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

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

Illustration of the ability to reference rule from other constraints

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

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

```
<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>
```

Illustration of the ability to reference attributes, options and operations from other resources

Advanced Resource Types

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.

```
<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="1.2.3.4"/>
    </instance_attributes>
  </primitive>
  <primitive id="Email" class="lsb" type="exim"/>
</group>
```

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

The group above is logically equivalent to writing:

```
<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="1.2.3.4"/>
      </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>
```

```
</constraints>
</configuration>
```

How the cluster sees a group resource

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

Properties

| Field | Description |
|-------|-------------------------|
| id | Your name for the group |

Options

| Field | Description |
|-------------|--|
| priority | If not all resources can be active, the cluster will stop lower priority resources in order to keep higher priority ones active. |
| target-role | What state should the cluster attempt to keep this resource in? Allowed values: Stopped, Started |
| is-managed | Is the cluster allowed to start and stop the resource? Allowed values: true , false |

Using Groups

Instance Attributes

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

Contents

Groups may only contain a collection of [primitive](#) cluster resources. To refer to the child of a group resource, just use the child's *id* instead of the group's.

Constraints

Although it is possible to reference the group's children in constraints, it is usually preferable to use the group's name instead.

```
<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>
```

Example constraints involving groups

Stickiness

Stickiness, the measure of how much a resource wants to stay where it is, is additive in groups. Every active member of the group will contribute its stickiness value to the group's total. So if the default *resource-stickiness* is 100 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.

Clones - Resources That Should be Active on Multiple Hosts

Clones were initially conceived as a convenient way to start N instances of an IP resource and have them distributed throughout the cluster for load balancing. They have turned out to quite useful for a number of purposes including integrating with Red Hat's DLM, the fencing subsystem and OCFS2.

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

Three types of cloned resources exist.

- Anonymous
- Globally Unique
- Stateful

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

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

Stateful clones are covered later in the [Advanced Resources - Multi-state](#) section.

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

An example clone

Properties

| Field | Description |
|-------|-------------------------|
| id | Your name for the clone |

Options

| Field | Description |
|-----------------|---|
| priority | If not all resources can be active, the cluster will stop lower priority resources in order to keep higher priority ones active. |
| target-role | What state should the cluster attempt to keep this resource in? Allowed values: Stopped, Started |
| is-managed | Is the cluster allowed to start and stop the resource? Allowed values: true , false |
| clone-max | How many copies of the resource to start. Defaults to the number of nodes in the cluster. |
| clone-node-max | How many copies of the resource can be started on a single node. Defaults to 1. |
| notify | When stopping or starting a copy of the clone, tell all the other copies beforehand and when the action was successful. Allowed values: true, false |
| globally-unique | Does each copy of the clone perform a different function? Allowed values: true , false |
| ordered | Should the copies be started in series (instead of in parallel). Allowed values: true, false |
| interleave | Changes the behavior of ordering constraints (between clones/masters) so that instances can start/stop as soon as their peer instance has (rather than waiting for every instance of the other clone has). Allowed values: true, false |

Clone configuration options

Using Clones

Instance Attributes

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

Contents

Clones must contain exactly one group or one regular resource.

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

Constraints

In most cases, a clone will have a single copy on each active cluster node. However 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 to those for regular resources except that the clone's *id* is used.

Ordering constraints behave slightly differently for clones. In the example below, *apache-stats* will wait until all copies of the 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 the clone.

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

Colocation between clones is also possible. In such cases, the set of allowed locations for the *rsc* clone is limited to nodes on which the *with* clone is (or will be) active. Allocation is then performed as-per-normal.

```
<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>
```

Example constraints involving clones

Stickiness

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

Resource Agent Requirements

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

Globally unique clones do require some additional support in the resource agent. In particular, it must only respond with *{OCF_SUCCESS}* if the node has that exact instance active. All other probes for instances of the clone should result in *{OCF_NOT_RUNNING}*. Unless of course they are failed, in which case they should return one of the other OCF error codes.

Copies of a clone are identified by appending a colon and a numerical offset. Eg. *apache:2*

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

You should not make any assumptions (based on `OCF_RESKEY_CRM_meta_clone`) about which copies are active. In particular, the list of active copies will not always be an unbroken sequence, nor always start at 0.

Notifications

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

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

Environment variables supplied with Clone notify actions

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

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

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

Example notification variables

Proper Interpretation of Notification Environment Variables

Pre-notification (stop)

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

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

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

- Resources that were stopped: `$OCF_RESKEY_CRM_meta_notify_stop_resource`

Post-notification (start)

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

Multi-state - Resources That Have Multiple Modes

Multi-state resources are a specialization of Clones (please ensure you understand the section on clones before continuing) that allow the instances to be in one of two operating modes. These modes are called Master and Slave but can mean whatever you wish them to mean. The only limitation is that when an instance is started, it must come up in the Slave state.

Properties

| Field | Description |
|-------|--|
| id | Your name for the multi-state resource |

Options

| Field | Description |
|-----------------|---|
| priority | If not all resources can be active, the cluster will stop lower priority resources in order to keep higher priority ones active. |
| target-role | What state should the cluster attempt to keep this resource in? Allowed values: Stopped, Started |
| is-managed | Is the cluster allowed to start and stop the resource? Allowed values: true , false |
| clone-max | How many copies of the resource to start. Defaults to the number of nodes in the cluster. |
| clone-node-max | How many copies of the resource can be started on a single node. Defaults to 1. |
| master-max | How many copies of the resource can be promoted to master status. Defaults to 1. |
| master-node-max | How many copies of the resource can be promoted to master status on a single node. Defaults to 1. |
| notify | When stopping or starting a copy of the clone, tell all the other copies beforehand and when the action was successful. Allowed values: true, false |
| globally-unique | Does each copy of the clone perform a different function? Allowed values: true , false |
| ordered | Should the copies be started in series (instead of in parallel). Allowed values: true, false |
| interleave | Changes the behavior of ordering constraints (between clones/masters) so that instances can start/stop as soon as their peer instance has (rather than waiting for every instance of the other clone has). Allowed values: true, false |

Multi-state resource configuration options

Using Multi-state Resources

Instance Attributes

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

Contents

Masters must contain exactly one group or one regular resource.

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

Monitoring Multi-State Resources

The normal type of monitor actions you define are not sufficient to monitor a multi-state resource in the Master state. To detect failures of the master instance, you need to define an additional monitor action with *role="Master"*.

NOTE: It is crucial that every monitor operation has a different interval

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

Monitoring both states of a multi-state resource

Constraints

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

When considering multi-state resources in constraints, for most purposes it is sufficient to treat them as clones. The exception is when the *rsc-role* and/or *with-rsc-role* (for colocation constraints) and *first-action* and/or *then-action* (for ordering constraints) are used.

| Field | Description |
|---------------|--|
| rsc-role | An additional attribute of colocation constraints that specifies the role that <i>rsc</i> must be in. Allowed values: Started , Master, Slave |
| with-rsc-role | An additional attribute of colocation constraints that specifies the role that <i>with-rsc</i> must be in. Allowed values: Started , Master, Slave |
| first-action | An additional attribute of ordering constraints that specifies the action that the <i>first</i> resource must complete before executing the specified action for the <i>then</i> resource. Allowed values: start , stop, promote, demote |
| then-action | An additional attribute of ordering constraints that specifies the action that the <i>then</i> resource can only execute after the <i>first-action</i> on the <i>first</i> resource has completed. Allowed values: start, stop, promote, demote. Defaults to the value of <i>first-action</i> |

Additional constraint options relevant to multi-state resources

In the example below, **myApp** will wait until one of **database** copies has been started and promoted to master before being started itself. Only if no copies can be promoted will **apache-stats** be prevented from being active. Additionally, the **database** will wait for **myApp** to be stopped before it is demoted.

Colocation of a regular (or group) resource with a multi-state resource means that it can run on any machine with an active copy of the clone that is in the specified state (Master or Slave). In the example, the cluster will choose a location based on where **database** is running as a Master, and if there are multiple Master instances it will also factor in **myApp**'s own location preferences when deciding which location to choose.

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

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

Example constraints involving multi-state resources

Stickiness

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

Which Resource Instance is Promoted

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

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

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

Manually specifying which node should be promoted

Resource Agent Requirements

Since multi-state resources are an extension of cloned resources, all the requirements of Clones are also requirements of multi-state resources. Additionally, multi-state resources require two extra actions **demote** and **promote**. These actions are responsible for changing the state of the resource. Like start and stop, they should return OCF_SUCCESS if they completed successfully or a relevant error code if they did not.

The states can mean whatever you wish, but when the resource is started, it must come up in the mode called Slave. From there the cluster will then decide which instances to promote into a Master.

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

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

Role implications of OCF return codes

Notifications

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

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

Environment variables supplied with Master notify actions¹⁰

¹⁰ Variables in bold are specific to Master resources and all behave in the same manner as described for Clone resources.

Proper Interpretation of Notification Environment VariablesPre-notification (demote)

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

Post-notification (demote) / Pre-notification (stop)

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

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

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

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

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

Post-notification (promote)

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

Advanced Configuration

© Andrew Beekhof

- Resources that were promoted: \$OCF_RESKEY_CRM_meta_notify_promote_resource
- Resources that were demoted: \$OCF_RESKEY_CRM_meta_notify_demote_resource
- Resources that were stopped: \$OCF_RESKEY_CRM_meta_notify_stop_resource

Protecting Your Data - STONITH

Why You Need STONITH

STONITH is an acronym for Shoot-The-Other-Node-In-The-Head and it protects your data from being corrupted by rogue nodes or concurrent access.

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

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


What STONITH Device Should You Use

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

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

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

Configuring STONITH

1. Find the correct driver: `stonith -L`
2. Find out the parameters required by the device: `stonith -t {type} -n`
3. To understand what the parameters mean, 
4. Create a file called stonith.xml containing a primitive resource with a *class* of `stonith`, a *type* of `{type}` and a parameter for each of the values returned in step 2
5. Create a clone from the primitive resource if it can shoot more than one node.
6. Upload it into the CIB using cibadmin: `cibadmin -C -o resources --xml-file stonith.xml`

Fill this in

Example

Assuming we have an IBM BladeCenter consisting of four nodes and the management interface is active on 10.0.0.1, then we would chose the `external/ibmrsa` driver in step 2 and obtain the following list of parameters

```
# stonith -t external/ibmrsa -n
hostname ipaddr userid passwd type
```

from which we would create a STONITH resource fragment that might look like this

```
<clone id="Fencing">
  <meta_attributes id="fencing">
    <nvpair id="Fencing-unique" name="globally-unique" value="false"/>
  </meta_attributes>
  <primitive id="rsa" class="stonith" type="external/ibmrsa">
    <operations>
      <op id="rsa-mon-1" name="monitor" interval="120s"/>
    </operations>
    <instance_attributes id="rsa-parameters">
      <nvpair id="rsa-attr-1" name="hostname" value="node1 node2 node3"/>
      <nvpair id="rsa-attr-1" name="ipaddr" value="10.0.0.1"/>
      <nvpair id="rsa-attr-1" name="userid" value="testuser"/>
      <nvpair id="rsa-attr-1" name="passwd" value="abc123"/>
      <nvpair id="rsa-attr-1" name="type" value="ibm"/>
    </instance_attributes>
  </primitive>
</clone>
```

Status - Here be dragons

Transient Node Attributes

Operation History

Appendix: FAQ

Why is the Project Called Pacemaker?

First of all, the reason its not called the CRM is because of the abundance of [terms](#) that are commonly abbreviated to those three letters.

The Pacemaker name came from [Kham](#), a good friend of mine, and was originally used by a Java GUI that I was prototyping in early 2007. Alas other commitments have prevented the GUI from progressing much and, when it came time to choose a name for this project, Lars suggested it was an even better fit for an independent CRM.

The idea stems from the analogy between the role of this software and that of the little device that keeps the human heart pumping. Pacemaker monitors the cluster and intervenes when necessary to ensure the smooth operation of the services it provides.

There were a number of other names (and acronyms) tossed around, but suffice to say "Pacemaker" was the best

Why was the Pacemaker Project Created?

The decision was made to spin-off the CRM into its own project after the 2.1.3 Heartbeat release in order to

- support both the OpenAIS and Heartbeat cluster stacks equally
- decouple the release cycles of two projects at very different stages of their life-cycles
- foster the clearer package boundaries, thus leading to
- better and more stable interfaces

What Messaging Layers are Supported?

- OpenAIS (<http://www.openais.org>)
- Heartbeat (<http://linux-ha.org/>)

Can I Choose which Messaging Layer to use at Run Time?

Yes. The CRM will automatically detect who started it and behave accordingly.

Can I Have a Mixed Heartbeat-OpenAIS Cluster?

No.

Which Messaging Layer Should I Choose?

This is discussed in the [Installation](#) appendix

Where Can I Get Pre-built Packages?

Official packages for most major .rpm and .deb based distributions are available from:

<http://download.opensuse.org/repositories/server:/ha-clustering/>

For more information, we have a [description of the available packages](#).

What Versions of Pacemaker Are Supported?

Please refer to the [Releases](#) page for an up-to-date list of versions supported directly by the project.

When seeking assistance, please try to ensure you have one of these versions.

Appendix: More About OCF Resource Agents

Location of Custom Scripts

OCF Resource Agents are found in `/usr/lib/ocf/resource.d/{provider}`.

When creating your own agents, you are encouraged to create a new directory under `/usr/lib/ocf/resource.d/` so that they are not confused with (or overwritten by) the agents shipped with Heartbeat. So, for example, if you chose the provider name of `bigCorp` and wanted a new resource named `bigApp`, you would create a script called `/usr/lib/ocf/resource.d/bigCorp/bigApp` and define a resource:

```
<primitive id="custom-app" class="ocf" provider="bigCorp" type="bigApp"/>
```

Actions

All OCF Resource Agents are required to implement the following actions

| Action | Description | Instructions |
|--------------|--|---|
| start | Start the resource | Return 0 on success and an appropriate error code otherwise. Must not report success until the resource is fully active. |
| stop | Stop the resource | Return 0 on success and an appropriate error code otherwise. Must not report success until the resource is fully stopped. |
| monitor | Check the resource's state | Exit 0 if the resource is running, 7 if it is stopped and anything else if it is failed. NOTE: The monitor script should test the state of the resource on the local machine only. |
| meta-data | Describe the resource | Provide information about this resource as an XML snippet. Exit with 0. NOTE: This is not performed as root. |
| validate-all | Verify the supplied parameters are correct | Exit with 0 if parameters are valid, 2 if not valid, 6 if resource is not configured. |

Required OCF actions

Additional requirements (not part of the OCF specs) are placed on agents that will be used for advanced concepts like clones and multi-state resources.

| Action | Description | Instructions |
|---------|--|---------------------|
| promote | Promote the local instance of a multi-state resource to the master/primary state | Return 0 on success |
| demote | Demote the local instance of a multi-state resource to the slave/secondary state | Return 0 on success |

| Action | Description | Instructions |
|--------|--|----------------------------|
| notify | Used by the cluster to send the agent pre and post notification events telling the resource what is or did just take place | Must not fail. Must exit 0 |

Optional extra actions

Some actions specified in the OCF specs are not currently used by the cluster

- reload - reload the configuration of the resource instance without disrupting the service
- recover - a variant of the start action, this should try to recover a resource locally.

Remember to use **ocf-tester** to verify that your new agent complies with the OCF standard properly.

How Does the Cluster Interpret the OCF Return Codes?

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

There are three types of failure recovery:

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

Types of recovery performed by the cluster

Assuming an action is considered to have failed, the following table outlines the different OCF return codes and the type of recovery the cluster will initiate when it is received.

| OCF Return Code | OCF Alias | Description | Recovery Type |
|-----------------|-----------------------|---|---------------|
| 0 | OCF_SUCCESS | Success. The command complete successfully. This is the expected result for all start, stop, promote and demote commands. | soft |
| 1 | OCF_ERR_GENERIC | Generic "there was a problem" error code. | soft |
| 2 | OCF_ERR_ARGS | The resource's configuration is not valid on this machine. Eg. Refers to a location/tool not found on the node. | hard |
| 3 | OCF_ERR_UNIMPLEMENTED | The requested action is not implemented. | hard |
| 4 | OCF_ERR_PERM | The resource agent does not have sufficient privileges to complete the task. | hard |
| 5 | OCF_ERR_INSTALLED | The tools required by the resource are not installed on this machine. | hard |
| 6 | OCF_ERR_CONFIGURED | The resource's configuration is invalid. Eg. A required parameters are missing. | fatal |

| OCF Return Code | OCF Alias | Description | Recovery Type |
|-----------------|--------------------|--|---------------|
| 7 | OCF_NOT_RUNNING | The resource is safely stopped. The cluster will not attempt to stop a resource that returns this for any action. | N/A |
| 8 | OCF_RUNNING_MASTER | The resource is running in Master mode. | soft |
| 9 | OCF_FAILED_MASTER | The resource is in Master mode but has failed. The resource will be demoted, stopped and then started (and possibly promoted) again. | soft |
| other | NA | Custom error code. | soft |

OCF Return Codes and How They are Handled

Although counter intuitive, even actions that return 0 (aka. OCF_SUCCESS) can be considered to have failed. This can happen when a resource that is expected to be in the Master state is found running as a Slave, or when a resource is found active on multiple machines..

Exceptions

- Non-recurring monitor actions (probes) that find a resource active (or in Master mode) will not result in recovery action unless it is also found active elsewhere
- The recovery action taken when a resource is found active more than once is determined by the multiple-active property of the resource
- Recurring actions that return OCF_ERR_UNIMPLEMENTED do not cause any type of recovery

Appendix: What Changed in 1.0

New

- Failure timeouts. See [Moving Resources Due to Failure](#)
- New section for resource and operation defaults. See Setting Global Defaults for Resource Options and Setting Global Defaults for Operations
- Tool for making offline configuration changes. See [Making Configuration Changes in a Sandbox](#)
- *Rules*, *instance_attributes*, *meta_attributes* and sets of *operations* can be defined once and referenced in multiple places. See [Reusing Rules, Options and Sets of Operations](#)
- The CIB now accepts XPath-based create/modify/delete operations. See the `cibadmin` help text.
- Multi-dimensional colocation and ordering constraints. See [Ordering](#) and [Collocating Sets of Resources](#)
- The ability to connect to the CIB from non-cluster machines. See [Connecting to the Cluster Configuration from Remote Machines](#)
- Allow recurring actions to be triggered at known times. See [Specifying When Recurring Actions are Performed](#)

Changed

- Syntax
 - All resource and cluster options now use dashes (-) instead of underscores (_)
 - *master_slave* was renamed to *master*
 - The *attributes* container tag was removed
 - The operation field *pre-req* has been renamed *requires*
 - All operations must have an interval, start/stop must have it set to zero
- The attributes of colocation and ordering constraints were renamed for clarity. See [Specifying the Order Resources Should Start/Stop In](#) and [Placing Resources Relative to other Resources](#)
- *resource-failure-stickness* has been replaced by *migration-threshold*. See [Moving Resources Due to Failure](#)
- The arguments for command-line tools has been made consistent
- Switched to RelaxNG schema validation and libxml2 parser.
 - *id* fields are now XML IDs which have the following limitations
 - *id*'s cannot contain colons (:)
 - *id*'s cannot begin with a number
 - *id*'s must be globally unique (not just unique for that tag)
 - Some fields (such as those in constraints that refer to resources) are *IDREFs*. This means that they must reference existing resources or objects in order for the configuration to be valid. Removing an object which is referenced elsewhere will therefor fail.
 - The CIB representation from which the md5 digest used to verify CIBs has changed. This means that every CIB update will require a full refresh on any upgraded nodes until the cluster is fully upgraded to 1.0. This will result in significant performance degradation and it is therefor highly inadvisable to run a mixed 1.0/0.6 cluster for any longer than absolutely necessary.
- Ping node information no longer needs to be added to ha.cf Simply include the lists of hosts in your pingd resource(s).

Removed

- Syntax
 - It is no longer possible to set resource meta options as top-level attributes. Use meta attributes instead.
 - Resource and operation defaults are no longer read from `crm_config`. See [Setting Global Defaults for Resource Options](#) and [Setting Global Defaults for Operations](#) instead.

Appendix: Installation

Choosing a Cluster Stack

Ultimately the choice of cluster stack is a personal decision that must be made in the context of you or your company's needs and strategic direction. Pacemaker currently functions equally well with both stacks.

Here are some factors that may influence the decision

- SUSE/Novell, Red Hat and Oracle are all putting their collective weight behind the OpenAIS cluster stack.
- OpenAIS is an OSI Certified implementation of an industry standard (the Service Availability Forum Application Interface Specification).
- Using OpenAIS gives your applications access to the following additional cluster services
 - checkpoint service
 - distributed locking service
 - extended virtual synchrony service
 - cluster closed process group service
- It is likely that Pacemaker, at some point in the future, will make use of some of these additional services not provided by Heartbeat
- To date, Pacemaker has received less real-world testing on OpenAIS than it has on Heartbeat.

Enabling Pacemaker

For OpenAIS

The OpenAIS configuration is normally located in `/etc/ais/openais.conf` and an example for a machine with an address of 1.2.3.4 in a cluster communicating on port 1234 (without peer authentication and message encryption) is shown below.

```
totem {
  version: 2
  secauth: off
  threads: 0
  interface {
    ringnumber: 0
    bindnetaddr: 1.2.3.4
    mcastaddr: 226.94.1.1
    mcastport: 1234
  }
}
logging {
  fileline: off
  to_syslog: yes
  syslog_facility: daemon
}
```

```
amf {
  mode: disabled
}
```

An example OpenAIS configuration file

The *logging* should be mostly obvious and the *amf* section refers to the Availability Management Framework and is not covered in this document.

The interesting part of the configuration is the *totem* section. This is where we define the how the node can communicate with the rest of the cluster and what protocol version and options (including encryption¹¹) it should use. Beginners are encouraged to use the values shown and modify the *interface* section based on their network.

It is also possible to configure OpenAIS for an IPv6 based environment. Simply configure *bindnetaddr* and *mcastaddr* with their IPv6 equivalents. Eg

```
bindnetaddr: fec0::1:a800:4ff:fe00:20
mcastaddr: ff05::1
```

Example options for an IPv6 environment

To tell OpenAIS to use the Pacemaker cluster manager, add the following fragment to a functional OpenAIS configuration and restart the cluster.

```
aisexec {
  user: root
  group: root
}
service {
  name: pacemaker
  ver: 0
}
```

Configuration fragment for enabling Pacemaker under OpenAIS

The cluster needs to be run as root so that its child processes (the *lrmd* in particular) have sufficient privileges to perform the actions requested of it. After-all, a cluster manager that can't add an IP address or start apache is of little use.

The second directive is the one that actually instructs the cluster to run Pacemaker.

For Heartbeat

Add the following to a functional *ha.cf* configuration file and restart Heartbeat

```
crm respawn
```

Configuration fragment for enabling Pacemaker under Heartbeat

¹¹ Please consult the OpenAIS website and documentation for details on enabling encryption and peer authentication for the cluster.

Appendix: Upgrading Cluster Software

Upgrade Methodologies

Version Compatibility

When releasing newer versions we take care to make sure we are backwardly compatible with older versions. While you will always be able to upgrade from version x to x+1, in order to continue to produce high quality software it may occasionally be necessary to drop compatibility with older versions.

There will always be an upgrade path from any series-2 release to any other series-2 release.

There are three approaches to upgrading your cluster software

1. Complete Cluster Shutdown
2. Rolling (node by node)
3. Disconnect & Reattach

Each method has advantages and disadvantages, some of which are listed in the table below, and you should chose the one most appropriate to your needs.

| Type | Available between all software versions | Service Outage During Upgrade | Service Recovery During Upgrade | Exercises Failover Logic/Configuration | Allows change of cluster stack |
|----------|---|-------------------------------|---------------------------------|--|--------------------------------|
| Shutdown | yes | always | N/A | no | yes |
| Rolling | no | always | yes | yes | no |
| Reattach | yes | only due to failure | no | no | yes |

Complete Cluster Shutdown

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

Procedure

1. On each node:
 - a. Shutdown the cluster stack (Heartbeat or OpenAIS)
 - b. Upgrade the Pacemaker software. This may also include upgrading the cluster stack and/or the underlying operating system..
2. Check the configuration manually or with the `crm_verify` tool if available.
3. On each node:

- a. Start the cluster stack. This can be either OpenAIS or Heartbeat and does not need to be the same as the previous cluster stack.

Rolling (node by node)

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

Procedure

On each node:

1. Shutdown the cluster stack (Heartbeat or OpenAIS)
2. Upgrade the Pacemaker software. This may also include upgrading the cluster stack and/or the underlying operating system.
 - a. On the first node, check the configuration manually or with the `crm_verify` tool if available.
3. Start the cluster stack. This must be the same cluster stack that the rest of the cluster is using.

Repeat for each node in the cluster

Version Compatibility

| Version being Installed | Oldest Compatible Version |
|---------------------------|--|
| Pacemaker 1.0 | Pacemaker 0.6 or Heartbeat 2.1.3 |
| Pacemaker 0.7 | Pacemaker 0.6 or Heartbeat 2.1.3 |
| Pacemaker 0.6 | Heartbeat 2.0.8 |
| Heartbeat 2.1.3 (or less) | Heartbeat 2.0.4 |
| Heartbeat 2.0.4 (or less) | Heartbeat 2.0.0 |
| Heartbeat 2.0.0 | None. Use an alternate upgrade strategy. |

Crossing Compatibility Boundaries

Rolling upgrades that cross compatibility boundaries must be preformed in multiple steps. For example, to perform a rolling update from Heartbeat 2.0.1 to Pacemaker 1.0.0 one must:

1. Perform a rolling upgrade from Heartbeat 2.0.1 to Heartbeat 2.0.4
2. Perform a rolling upgrade from Heartbeat 2.0.4 to Heartbeat 2.1.3
3. Perform a rolling upgrade from Heartbeat 2.1.3 to Pacemaker 1.0.0

Disconnect & Reattach

A variant of a complete cluster shutdown, but the resources are left active and re-detected when the cluster is restarted.

Procedure

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

```
crm_attribute -t crm_config -n is-managed-default -v false
```

2. For any resource that has a value for *is-managed*, make sure it is set to false (so that the cluster will not stop it)

```
crm_resource -t primitive -r <rsc_id> -p is-managed -v false
```

3. On each node:

- a. Shutdown the cluster stack (Heartbeat or OpenAIS)
 - b. Upgrade the cluster stack program - This may also include upgrading the underlying operating system.
4. Check the configuration manually or with the `crm_verify` tool if available.
 5. On each node:
 - a. Start the cluster stack. This can be either OpenAIS or Heartbeat and does not need to be the same as the previous cluster stack.
 6. Verify the cluster re-detected all resources correctly
 7. Allow the cluster to resume managing resources again
`crm_attribute -t crm_config -n is-managed-default -v true`
 8. For any resource that has a value for is-managed reset it to true (so the cluster can recover the service if it fails) if desired
`crm_resource -t primitive -r <rsc_id> -p is-managed -v false`

Notes

- Always check your existing configuration is still compatible with the version you are installing before starting the cluster.
- The oldest version of the CRM to support this upgrade type was in Heartbeat 2.0.4

Appendix: Upgrading the Configuration from 0.6

Overview

Preparation

Download the latest DTD from <http://hg.clusterlabs.org/pacemaker/dev/file-raw/tip/xml/crm.dtd> and ensure your configuration validates.

Perform the upgrade

Upgrade the software

Refer to the appendix: [Upgrading Cluster Software](#)

Upgrade the Configuration

As XML is not the friendliest of languages, it is common for cluster administrators to have scripted some of their activities. In such cases, it is likely that those scripts will not work with the new 1.0 syntax.

In order to support such environments, it is actually possible to continue using the old 0.6 syntax.

The downside however, is that not all the new features will be available and there is a performance impact since the cluster must do a non-persistent configuration upgrade before each transition. So while using the old syntax is possible, it is not advisable to continue using it indefinitely.

Even if you wish to continue using the old syntax, it is advisable to follow the upgrade procedure to ensure that the cluster is able to use your existing configuration (since it will perform much the same task internally).

1. Create a shadow copy to work with

```
crm_shadow --create upgrade06
```

2. Verify the configuration is valid

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

3. Fix any errors or warnings

4. Perform the upgrade

```
cibadmin --upgrade
```

If this step fails, there are three main possibilities

- The configuration was not valid to start with - go back to step 2
- The transformation failed - report a bug or email the project at pacemaker@clusterlabs.org

- The transformation was successful but produced an invalid result¹²

If the result of the transformation is invalid, you may see a number of errors from the validation library. If these are not helpful, visit http://clusterlabs.org/mw/Validation_FAQ and/or try the following procedure described below under "Manually Upgrading the Configuration".

5. Check the changes

```
crm_shadow --diff
```

If at this point there is anything about the upgrade that you wish to fine-tune (for example, to change some of the automatic IDs) now is the time to do so. Since the shadow configuration is not in use by the cluster, it is safe to edit the file manually:

```
crm_shadow --edit
```

Will open the configuration in your favorite editor (or whichever one is specified by the standard EDITOR environment variable).

6. Preview how the cluster will react

Test what the cluster will do when you upload the new configuration

```
ptest -VVVVV --live-check --save-dotfile upgrade06.dot
graphviz upgrade06.dot
```

Verify that either no resource actions will occur or that you are happy with any that are scheduled. If the output contains actions you do not expect (possibly due to changes to the score calculations), you may need to make further manual changes. See [Testing Your Configuration Changes](#) for further details on how to interpret the output of `ptest`.

7. Upload the changes

```
crm_shadow --commit upgrade06 --force
```

If this step fails, something really strange has occurred. You should report a bug.

Manually Upgrading the Configuration

It is also possible to perform the configuration upgrade steps manually. To do this

1. Locate the upgrade06.xsl conversion script or download the latest version from <http://hg.clusterlabs.org/pacemaker/dev/file-raw/tp/xml/upgrade06.xsl>
2. `xsitproc /path/tp/upgrade06.xsl config06.xml > config10.xml`
3. Locate the pacemaker.rng script.
4. `xmllint --relaxng /path/tp/pacemaker.rng config10.xml`

The advantage of this method is that it can be performed without the cluster running and any validation errors should be more informative (despite being generated by the same library!) since they include line numbers.

¹² The most common reason is ID values being repeated or invalid. Pacemaker 1.0 is much stricter regarding this type of validation

Appendix: Is This init Script LSB Compatible?

Assuming **some_service** is configured correctly and currently not active, the following sequence will help you determine if it is LSB compatible:

1. Start (stopped):
`/etc/init.d/some_service start ; echo "result: $?"`
 - a. Did the service start?
 - b. Did the command print result: 0 (in addition to the regular output)?
2. Status (running):
`/etc/init.d/some_service status ; echo "result: $?"`
 - a. Did the script accept the command?
 - b. Did the script indicate the service was running?
 - c. Did the command print result: 0 (in addition to the regular output)?
3. Start (running):
`/etc/init.d/some_service start ; echo "result: $?"`
 - a. Is the service still running?
 - b. Did the command print result: 0 (in addition to the regular output)?
4. Stop (running):
`/etc/init.d/some_service stop ; echo "result: $?"`
 - a. Was the service stopped?
 - b. Did the command print result: 0 (in addition to the regular output)?
5. Status (stopped):
`/etc/init.d/some_service status ; echo "result: $?"`
 - a. Did the script accept the command?
 - b. Did the script indicate the service was not running?
 - c. Did the command print result: 3 (in addition to the regular output)?
6. Stop (stopped):
`/etc/init.d/some_service stop ; echo "result: $?"`
 - a. Is the service still stopped?
 - b. Did the command print result: 0 (in addition to the regular output)?
7. Status (failed):
This step is not readily testable and relies on manual inspection of the script.
The script can use one of the error codes (other than 3) listed in the LSB spec to indicate that it is active but failed.
This tells the cluster that before moving the resource to another node, it needs to stop it on the existing one first.

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

Appendix: Sample Configurations

An Empty Configuration

```
<cib admin_epoch="0" epoch="0" num_updates="0" have-quorum="false">
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
```

An empty configuration

A Simple Configuration

```
<cib admin_epoch="0" epoch="1" num_updates="0" have-quorum="false" validate-with="pacemaker-1.0">
  <configuration>
    <crm_config>
      <nvpair id="option-1" name="symmetric-cluster" value="true"/>
      <nvpair id="option-2" name="no-quorum-policy" value="stop"/>
    </crm_config>
    <op_defaults>
      <nvpair id="op-default-1" name="timeout" value="30s"/>
    </op_defaults>
    <rsc_defaults>
      <nvpair id="rsc-default-1" name="resource-stickiness" value="100"/>
      <nvpair id="rsc-default-2" name="migration-threshold" value="10"/>
    </rsc_defaults>
    <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>
          <nvpair name="ip" value="10.0.200.30"/>
        </instance_attributes>
      </primitive>
    </resources>
  </configuration>
  <status/>
</cib>
```

```

</instance_attributes>
</primitive>
</resources>
<constraints>
  <rsc_location id="myAddr-preferred-host" rsc="myAddr" node="c001n01" score="INFINITY"/>
</constraints>
</configuration>
<status/>
</cib>

```

2 nodes, some cluster options and a resource

In this 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.

An Advanced Configuration

```

<cib admin_epoch="0" epoch="1" num_updates="0" have-quorum="false" validate-with="pacemaker-1.0">
  <configuration>
    <crm_config>
      <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"/>
    </crm_config>
    <op_defaults>
      <nvpair id="op-default-1" name="timeout" value="30s"/>
    </op_defaults>
    <rsc_defaults>
      <nvpair id="rsc-default-1" name="resource-stickiness" value="100"/>
      <nvpair id="rsc-default-2" name="migration-threshold" value="10"/>
    </rsc_defaults>
    <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>
          <nvpair name="ip" value="10.0.200.30"/>
        </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-atrs">
            <nvpair id="stonith-attr-1" name="hostlist" value="c001n01,c001n02"/>
        </instance_attributes>
    </primitive>
</clone>
</resources>
<constraints>
    <rsc_location id="myAddr-preferred-host" rsc="myAddr" node="c001n01" score="INFINITY"/>
    <rsc_colocation id="group-with-ip" rsc="myGroup" with-rsc="myAddr" score="INFINITY"/>
</constraints>
</configuration>
<status/>
</cib>

```

groups and clones with stonith

Appendix: Further Reading

Project Website

<http://www.clusterlabs.org>

Cluster Commands

A comprehensive guide to cluster commands has been written by Novell and can be found at:

<http://www.novell.com/documentation/sles10/heartbeat/index.html?page=/documentation/sles10/heartbeat/data/heartbeat.html>

Heartbeat configuration

<http://www.linux-ha.org>

OpenAIS Configuration

<http://www.openais.org>