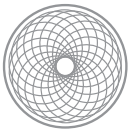


Pacemaker

Pacemaker Configuration

Explained

by Andrew Beekhof

pacemaker@clusterlabs.org

Table of Contents

Installation

Choosing a Cluster Stack

Configuration Basics

Configuration Layout

The cluster is 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.

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 us 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 --cib_query > tmp.xml
vi tmp.xml
cibadmin --cib_replace --xml-file tmp.xml
```

Some of the more better XML editors can make use of a DTD to help make sure any changes you make are valid. The DTD describing the configuration can normally be found in **/usr/lib/heartbeat/crm.dtd** on most systems.

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

```
cibadmin --cib_query --obj_type resources > tmp.xml
vi tmp.xml
cibadmin --cib_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 --cib_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
```

What About the GUI?

The GUI implements a fundamentally broken design that has been known to destroy configurations it doesn't understand (which is plenty of them). As far as I am concerned it does not exist and it will not be mentioned again in this document.

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.

Testing Your Configuration

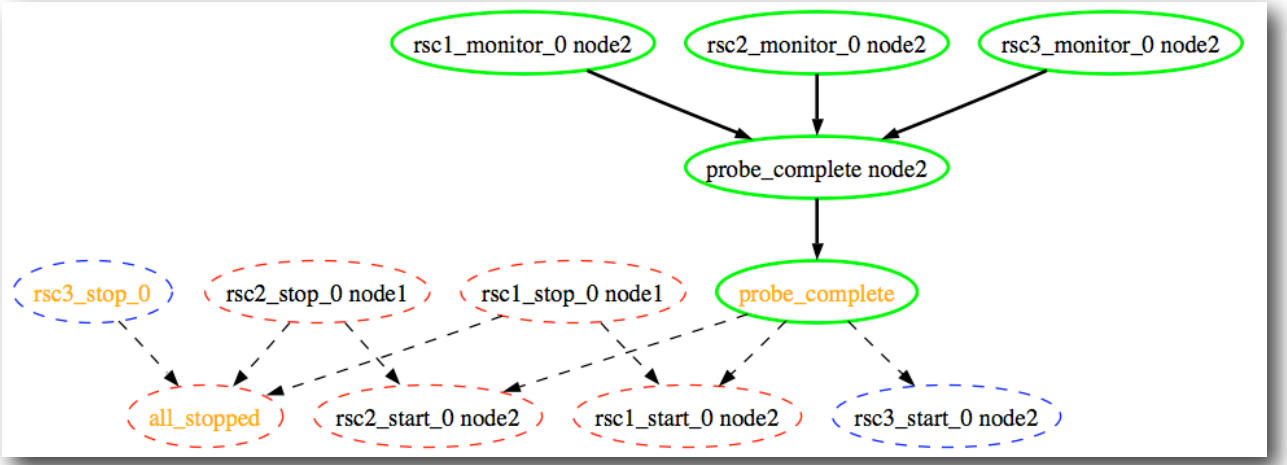
It is not necessary to modify a real cluster in order to test the effect of most configuration changes. Earlier, we saw how to download and modify a local copy of the cluster configuration.

```
cibadmin --cib_query > tmp.xml
vi tmp.xml
```

Now instead of loading the changes back into the cluster, simulate the effect of the changes with **ptest**. Eg.

```
ptest -VVVVV --xml-file tmp.xml --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. It's 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.



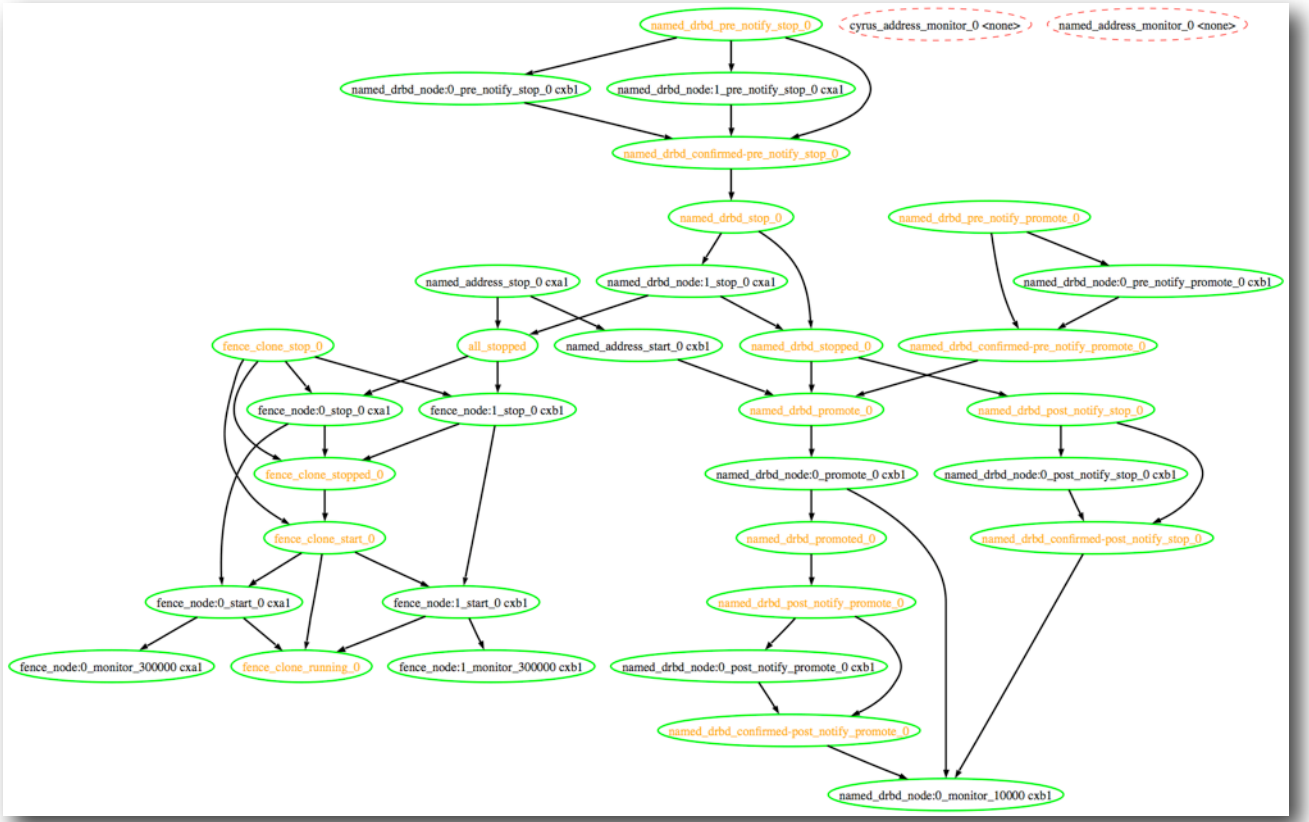
An example transition graph as represented by Graphviz

Interpreting the Graphviz output

- Arrows indicate dependancies
- Dashed-arrows indicate dependancies that are not present in 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 a dashed border of any color do not form part of the transition graph
- Actions with orange text are pseudo/pretend actions
- 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

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
ignore_dtd	If set to FALSE, the cluster will not verify that updates conform the the DTD (and reject ones that don’t). This option can be useful when operating a mixed version cluster during an upgrade.
remote_access_port	

Fields Maintained by the Cluster

Field	Description
generated	If true, indicates that a default configuration was used. Informational purposes only.
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.
num_peers	Automatically set at runtime based on the number of active peers. Informational purposes only.
ccm_transition	Indicates the most recent membership data the configuration database has received. This is used by the cluster to ensure that its view of the world is consistent.
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 --cib_modify --crm_xml ‘<cib admin_epoch=”42”/>’
```

A complete set of fields will look something like this:

```
<cib cib_feature_revision="1" have_quorum="true" ignore_dtd="false" admin_epoch="1" num_peers="4"
    epoch="12" ccm_transition="3" generated="true" dc_uuid="ea7d39f4-3b94-4cfa-ba7a-952956daabee"
    num_updates="65">
```

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.[1] 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 parallell. 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.
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.
default-resource-stickiness	0	How much do resources prefer to stay where they are? Used when no value for resource-stickiness is supplied by the resource.
default-resource-failure-stickiness	0	How much do resources prefer to move nodes when they fail? Used when no value for resource-stickiness is supplied by the resource.
is-managed-default	TRUE	Should the cluster start/stop resources as required
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.
default-action-timeout	20s	How long to wait for actions to complete by default
stop-orphan-resources	TRUE	Should deleted resources be stopped
stop-orphan-actions	TRUE	Should deleted actions be cancelled

[1] 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 the their preferred hosts without bothering end users)

Option	Default	Description
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.

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 -G -n cluster-delay
```

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

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

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

```
sles-1:~ # crm_attribute -G -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 suppy 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">
    <attributes>
      <nvpair id="kernel-1186dc9a-324d-425a-966e-d757e693dc86" name="kernel" value="2.6.16.46-0.4-default"/>
    </attributes>
  </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

TBA

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

Replacing a Cluster Node

OpenAIS

TBA

Heartbeat

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

1. Make sure the old node is completely stopped
2. Go to an active cluster node and look up the UUID for the old node in **/var/lib/heartbeat/hostcache**
3. Install the cluster software
4. Copy ha.cf and authkeys to the new node
5. On the new node, populate it's UUID using **crm_uuid -w** and the UUID from step 2
6. 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 [2] 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 return[3]. 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://www.linuxbase.org/spec/refspecs/LSB_3.0.0/LSB-Core-generic/LSB-Core-generic/iniscrptact.html

---

[2] Note: Linux-ha implementation have been somewhat extended from the OCF Specs, but none of those changes are incompatible with the original OCF specification

[3] 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 as it is both 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 resource

Options

Options are used by the cluster to decide how your resource should behave and can be easily set using the **crm_resource --meta** 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**
resource_failure_stickiness	How much does the resource prefer to move nodes when it fails? Defaults to the value of **default-resource-failure-stickiness**
multiple_active	What should the cluster do if it ever finds the resource active on more than one node. Allowed values: block, stop_only, **stop_start**

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">
    <attributes>
      <nvpair id="email-priority" name="priority" value="100"/>
      <nvpair id="email-active" name="multiple_active" value="block"/>
    </attributes>
  </meta_attributes>
</primitive>
```

An example LSB resource with cluster options

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">
    <attributes>
      <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
    </attributes>
  </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">
    <attributes>
      <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
    </attributes>
  </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	
timeout	How long to wait before declaring the action has failed.
role	
prereq	
on_fail	The action to take if this action ever fails. Allowed values: ignore, block, stop, restart, fence
disabled	If true, the operation is treated as if it does not exist. Allowed values: true, **false**

Valid fields for an operation

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">
      <attributes>
        <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
      </attributes>
    </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">
          <attributes>
            <nvpair id="public-ip-depth-60" name="OCF_CHECK_LEVEL" value="10"/>
          </attributes>
        </instance_attributes>
      </op>
      <op id="public-ip-health-60" name="monitor" interval="300">
        <instance_attributes id="params:public-ip-depth-300">
          <attributes>
            <nvpair id="public-ip-depth-300" name="OCF_CHECK_LEVEL" value="20"/>
          </attributes>
        </instance_attributes>
      </op>
    </operations>
    <instance_attributes id="params:public-ip">
      <attributes>
        <nvpair id="public-ip-level" name="ip" value="1.2.3.4"/>
      </attributes>
    </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">
    <attributes>
      <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
    </attributes>
  </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 migrating 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 resourc 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="100"/>
   <rsc_location id="loc-3" rsc="Database" node="sles-2" score="200"/>
   <rsc_location id="loc-4" rsc="Database" node="sles-3" score="100"/>
</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_ordering*** constraints.

Field	Description
id	A unique name for the constraint
from	The name of a resource. This resource will start after the ***to*** resource.
to	The name of a resource that must be started before the ***from*** resource is allowed to.
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 *to* resource cannot run without the *from* 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 *from* resource will react when the *to* resource changes state.

- If the *to* resource was running and is stopped, the *from* resource will also be stopped (if it is running)
- If the *to* resource was not running and cannot be started, the *from* resource will be stopped (if it is running)
- If the *to* resource is (re)started while the *from* resource is running, the *from* 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 *to* resource will have no effect on the *from* resource.

```
<constraints>
   <rsc_order id="order-1" from="Webserver" to="Database"/>
   <rsc_order id="order-2" from="Webserver" to="IP" 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 is[4]. 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
from	The colocation source. If the constraint cannot be satisfied, the cluster may decide not to allow the resource to run at all.
to	The colocation target. The cluster will decide where to put this resource first and then decide where to put the resource in the *from* 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".

[4] 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 ***from*** resource is not permitted to run. For ***score=INFINITY***, this includes cases where the ***to*** 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" from="resource1" to="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" from="resource1" to="resource2" score="-INFINITY"/>
```

An example anti-colocation constraint

Again, by specifying -INFINTY, 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" from="resource1" to="resource2" score="500"/>
```

An example advisory-only colocation constraint

Advanced Configuration

Rules

Rules can be used to make your configuration more dynamic. One common example is to set one value for *default-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.

* role

  limits this rule to applying to Multi State resources with the named role.

  Roles include Started, Stopped, Slave, Master though only the last two are considered useful.

  NOTE: A rule with role="Master" can not determin the initial location of a clone instance.

  It will only affect which of the active instances will be promoted.

* score_attribute

  an alternative to the score attribute that provides extra flexibility.

  Each node matched by the rule has its score adjusted differently, according to its value for the named node attribute.

  Thus in the example below, if score_attribute="installed_ram" and nodeA would have its preference to run "the resource" increased by 1024 whereas nodeB would have its preference increased only by half as much.

```
<nodes>
  <node id="uuid1" uname="nodeA" type="normal">
    <instance_attributes id="uuid1:custom_attrs">
      <attributes>
        <nvpair id="uuid1:installed_ram" name="installed_ram" value="1024"/>
        <nvpair id="uuid1:my_other_attr" name="my_other_attr" value="bob"/>
      </attributes>
    </instance_attributes>
  </node>
  <node id="uuid2" uname="nodeB" type="normal">
    <instance_attributes id="uuid2:custom_attrs">
      <attributes>
        <nvpair id="uuid2:installed_ram" name="installed_ram" value="512"/>
      </attributes>
    </instance_attributes>
  </node>
</nodes>
```

Returns TRUE or FALSE depending on the properties of the object being tested.

* type determines how the values being tested.
  * integer Values are converted to floats before being compared.
  * version The "version" type is intended to solve the problem of comparing 1.2 and 1.10
  * string Uses strcmp

Built-in attribute node uname: #uname

```
<!ATTLIST expression
        id         CDATA                    #REQUIRED
        attribute  CDATA                    #REQUIRED
        operation  (lt|gt|lte|gte|eq|ne|defined|not_defined) #REQUIRED
        value      CDATA                    #IMPLIED
        type       (number|string|version) 'string'>
```

```
<!--
```

* start : A date-time conforming to the ISO8601 specification.
* end : A date-time conforming to the ISO8601 specification.

  A value for end may, for any usage, be omitted and instead inferred using start and duration.

* operation
  * gt : Compares the current date-time with start date.

    Checks now > start.
  * lt : Compares the current date-time with end date.

    Checks end > now
  * in_range : Compares the current date-time with start and end.

    Checks now > start and end > now.

    If either start or end is omitted, then that part of the comparision is not performed.
  * date_spec : Performs a cron-like comparision between the contents of date_spec and now.

    If values for start and/or end are included, now must also be within that range.

    Or in other words, the date_spec operation can also be made to perform an extra in_range check.

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

-->

<!ELEMENT date_expression (date_spec?,duration?)>

<!ATTLIST date_expression

id CDATA #REQUIRED

operation (in_range|date_spec|gt|lt) 'in_range'

start CDATA #IMPLIED

end CDATA #IMPLIED>

<!--

date_spec is used for (surprisingly) date_spec operations.

Fields that are not supplied are ignored.

Fields can contain a single number or a single range.

Eg.

monthdays="1" (Matches the first day of every month) and hours="09-17" (Matches hours between 9am and 5pm inclusive) are both valid values.

weekdays="1,2" and weekdays="1-2,5-6" are NOT valid ranges.

This may change in a future release.

* seconds : Value range 0-59

* minutes : Value range 0-59

* hours : Value range 0-23

* monthdays : Value range 0-31 (depending on current month and year)

* weekdays : Value range 1-7 (1=Monday, 7=Sunday)

* yeardays : Value range 1-366 (depending on current year)

* months : Value range 1-12

* weeks : Value range 1-53 (depending on weekyear)

* weekyears : Value range 0...

```
  (NOTE: weekyears may differ from Gregorian years.

  Eg. 2005-001 Ordinal == 2005-01-01 Gregorian == 2004-W53-6 Weekly )

 * years : Value range 0...

 * moon : Value range 0..7 - 0 is new, 4 is full moon.

   Because we can(tm)

-->


<!ELEMENT date_spec EMPTY>

<!ATTLIST date_spec

        id          CDATA  #REQUIRED

        hours       CDATA  #IMPLIED

        monthdays   CDATA  #IMPLIED

        weekdays    CDATA  #IMPLIED

        yeardays    CDATA  #IMPLIED

        months      CDATA  #IMPLIED

        weeks       CDATA  #IMPLIED

        weekyears   CDATA  #IMPLIED

        years       CDATA  #IMPLIED

        moon        CDATA  #IMPLIED>


<!--

duration is optionally used for calculating a value for end.

Any field not supplied is assumed to be zero and ignored.

Negative values might work.

Eg. months=11 should be equivalent to writing years=1, months=-1 but is not encouraged.

-->

<!ELEMENT duration EMPTY>

<!ATTLIST duration

        id          CDATA  #REQUIRED

        hours       CDATA  #IMPLIED

        monthdays   CDATA  #IMPLIED
```

```
  weekdays   CDATA  #IMPLIED
  yeardays   CDATA  #IMPLIED
  months     CDATA  #IMPLIED
  weeks      CDATA  #IMPLIED
  years      CDATA  #IMPLIED>
```

```
<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" operation="date_spec" start="2005-03-01" end="2005-04-01">
    <date_spec weekdays="1"/>
  </date_expression>
</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>
```

Friday the 13th if it is a full moon

Field	Description
id	A unique name for the rule
score	

Field	Description
score_attribute	
score	
role	Limit the rule to resources in the specified role. Allowed values: **Started**, Slave, Master
boolean_op	How the results of multiple expressions are combined. Allowed values: **and**, or

Properties of a rule

Node Attribute Expressions

Field	Description
id	A unique name for the expression
attribute	The name of a node attribute. Use the built-in value #uname for the node's uname.
operation	Allowed values: lt, gt, lte, gte, eq, ne, defined, not_defined
value	
type	How to perform the attribute/value comparison. Allowed values: number, **string**, version

Properties of an expression

Time Based Expressions

Field	Description
id	A unique name for the expression
operation	
start	
end	

Properties of a time-based expression

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)

Field	Description
months	Allowed values: 1-12
weeks	Allowed values: 0-53 (depending on weekyear)
years	Year according the Gregorian calendar
weekyears	May differ from Gregorian years
moon	Allowed values: 0..7 (0 is new, 4 is full moon)

Date Spec

Duration

Ensuring Time Based Rules Take Effect

Example Rules

Time Based Cluster Options

Location Based Resource Options

Using Attributes to Determine Resource Location

Resource Migration

Manual Migration

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***, migrate 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.

Migrating Due to Failure

WARNING: The contents of this section are likely to change in upcoming cluster versions as the concept of failure stickiness is widely considered to be far too complex.

Everything is mapped to a score, failure is no different. Each time a resource fails on a node, its ***failcount*** is incremented on that node. This is then multiplied by the resource's failure stickiness and subtracted from the node's overall score.

So if *default-failure-stickiness=100* and *Email* had failed 5 times on ***sles-1***, then that is the equivalent of having the following additional constraint:

```
<rsc_location id="loc-1" rsc="Webserver" node="sles-1" score="-500"/>
```

Eventually the resource will fail enough times that the node ends up with a negative score and the resource will be moved to another machine. There are some formulas for determining what value of ***default-failure-stickiness*** one should use to achieve failover after N failures, however as the complexity of the configuration grows, so do the formulas.

For now, the simplest way to pick a value for failure stickiness is to

1. Start the cluster
2. Wait until everything is running and the cluster is idle
3. run ./ptest -VVVVVV -L 2>&1 | grep "native_assign_node: Color"

You now have the final scores for every node and resource in the cluster[5]. Now, for each resource you care about:

4. Find the highest value
5. Divide this value by N, where N is the number of times you'd like the resource to fail. Call this value *sticky*.
6. Set the resource's failure stickiness

   crm_resource --resource *resource-name* --meta --set-parameter resource_failure_stickiness --property-value *sticky*

Migration Due to Connectivity Changes - Heartbeat only

WARNING: The contents of this section are likely to change in upcoming cluster versions when a stack-independent version of pingd becomes available.

Setting up the cluster to migrate resources when external connectivity is a three-step process.

1. **Tell Heartbeat how to detect connectivity**

Add one or more ping nodes to /etc/ha.d/ha.cf and restart Heartbeat on all cluster nodes.

```
ping 192.168.9.1
```

Example ping directive in ha.cf

2. **Tell Pacemaker to request connectivity data from Heartbeat**

To do this, you need to add a **pingd** resource to the cluster. **pingd** is a small daemon that receives node status information from Heartbeat and uses it to maintain a node attribute also called ***pingd***.

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 connectively 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.

Standard pingd options

[5] If you see one node with a score of 1000000 and all other nodes with a score of -1000000, then the resource is collocated with another resource. In such cases, you should use the scores of the other resource when following steps 4-6.

```
<clone id="pingd-clone">
  <primitive id="pingd" provider="heartbeat" class="ocf" type="pingd">
    <instance_attributes id="pingd-attrs">
      <attributes>
        <nvpair id="pingd-dampen" name="dampen" value="5s"/>
        <nvpair id="pingd-multiplier" name="multiplier" value="1000"/>
      </attributes>
    </instance_attributes>
  </primitive>
</clone>
```

An example pingd cluster resource

3. **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 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 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">
      <attributes>
        <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
      </attributes>
    </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 started 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">
     <attributes>
      <nvpair id="public-ip-addr" name="ip" value="1.2.3.4"/>
     </attributes>
    </instance_attributes>
   </primitive>
   <primitive id="Email" class="lsb" type="exim"/>
  </resources>
  <constraints>
    <rsc_colocation id=”xxx” from=”Email” to=”Public-IP” score=”INFINITY”/>
    <rsc_order id=”yyy” from=”Email” to=”Public-IP”/>
  </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” from=”Webserver” to=”shortcut”/>
```

```
    <rsc_order id="start-group-then-webserver" from="Webserver" to="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 ***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.

Likewise failure stickiness, the measure of how much a resource wants to move after a failure, is also cumulative. If three children of a group fail, it is as if the group as a whole had failed three times.

Advanced Resources - Clones

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">
      <attributes>
        <nvpair id="apache-unique" name="globally_unique" value="false"/>
      </attributes>
    </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	Allowed values: true, **false**

Clone configuration options

Using Clones

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.

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 to 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 *from* resource's own location preferences.

Colocation between clones is also possible. In such cases, the set of allowed locations for the *from* clone is limited to nodes on which the *to* 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" from="apache-stats" to="apache-clone"/>
   <rsc_order id="start-clone-then-stats" from="apache-stats" to="apache-clone"/>
</constraints>
```

Example constraints involving clones

Stickiness

To achieve a stable allocation pattern, clones are slightly sticky by default. If no value for *default-resource-stickiness* or *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.

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

Environment variables supplied with 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_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

Advanced Resources - Multi-state

Active instances of these resources are divided into two states active and passive (aka. primary/secondary, master/slave, choose whichever term suits you). Stateful clones can be either anonymous or globally unique.

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.

Field	Description
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	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 group's children.

Contents

Constraints

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

```
<constraints>
   <rsc_location id="db-prefers-node1" rsc="database" node="node1" score="500"/>
   <rsc_colocation id="backup-with-db" from="backup" to="database" to_role="Slave"/>
   <rsc_order id="start-db-then-backup" from="backup" to="database"/>
</constraints>
```

Example constraints involving multi-state resources

- Constraints for Multi-state Resources
- Colocation Based on Role
- Waiting for the Master to Start
- Choosing Where Masters Should be Placed

Stickiness

Protecting Your Data - STONITH

Why You Need STONITH

Configuring STONITH

Status

Transient Node Attributes

Operation History

Appendix: FAQ

How Can I Tell if an Init Script is 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: $?"
   - Did the service start?
   - Did the command print result: 0 (in addition to the regular output)?
2. Status (running): /etc/init.d/some_service status ; echo "result: $?"
   - Did the script accept the command?
   - Did the script indicate the service was running?
   - Did the command print result: 0 (in addition to the regular output)?
3. Start (running): /etc/init.d/some_service start ; echo "result: $?"
   - Is the service still running?
   - Did the command print result: 0 (in addition to the regular output)?
4. Stop (running): /etc/init.d/some_service stop ; echo "result: $?"
   - Was the service stopped?
   - Did the command print result: 0 (in addition to the regular output)?
5. Status (stopped): /etc/init.d/some_service status ; echo "result: $?"
   - Did the script accept the command?
   - Did the script indicate the service was not running?
   - Did the command print result: 3 (in addition to the regular output)?
6. Stop (stopped): /etc/init.d/some_service stop ; echo "result: $?"
   - Is the service still stopped?
   - 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.

What Do I Need to Know When Writing an OCF Resource Agent?

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/** and use provider={your subdirectory name} in the configuration. So, for example, if you want to name your provider **dubrouski**, and create a resource named **serge**, you would make a directory called **/usr/lib/ocf/resource.d/dubrouski** and name your resource script **/usr/lib/ocf/resource.d/dubrouski/serge**.

All OCF Resource Agents are required to implement the following actions

Action	Description	Instructions
start	Start the resource	Exit 0 when the resource is correctly running (i-e providing the service) and anything else except 7 if it failed
stop	Stop the resource	Exit 0 when the resource is correctly stopped and anything else except 7 if it failed
monitor	Check the resource's health/state	Exit 0 if the resource is running, 7 if it is stopped and anything else if it is failed. Note that 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.
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 resource to the master/primary state	Must not fail. Must exit 0
demote	Demote the local instance of a resource to the slave/secondary state	Must not fail. Must exit 0
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

What Do the Resource Agent Return Codes Mean?

There are three types of failure recovery:

- soft = stop and retry
- hard = stop and retry - current node is excluded
- fatal = stop - all nodes are excluded

OCF Return Code	Description	Recovery Type
0	Success. The command complete successfully. This is the expected result for all start, stop, promote and demote commands.	N/A
1	Generic “there was a problem” error code.	soft
2	Incorrect arguments were passed to the resource agent.	fatal
3	The requested action is not implemented.	hard
4	The resource agent does not have sufficient privileges to complete the task.	hard
5	The requested agent or tool required by the agent is not installed.	hard
6	The resource’s configuration is invalid.	fatal
7	The resource is safely stopped. The cluster will not attempt to stop a resource that returns this for any action.	N/A
8	The resource is running in Master mode.	N/A
9	The resource is in Master mode but has failed. The resource will be demoted, stopped and then started (and possibly promoted) again.	soft
other	Custom error code.	soft

OCF Return Codes and How They are Handled

Appendix: Sample Configurations

An Empty Configuration

```
<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

A Simple Configuration

2 nodes, some cluster options and a resource

A Standard Configuration

3 nodes, some options, a number of resources with constraints

An Advanced Configuration

groups and clones with stonith

Appendix: Further Reading

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