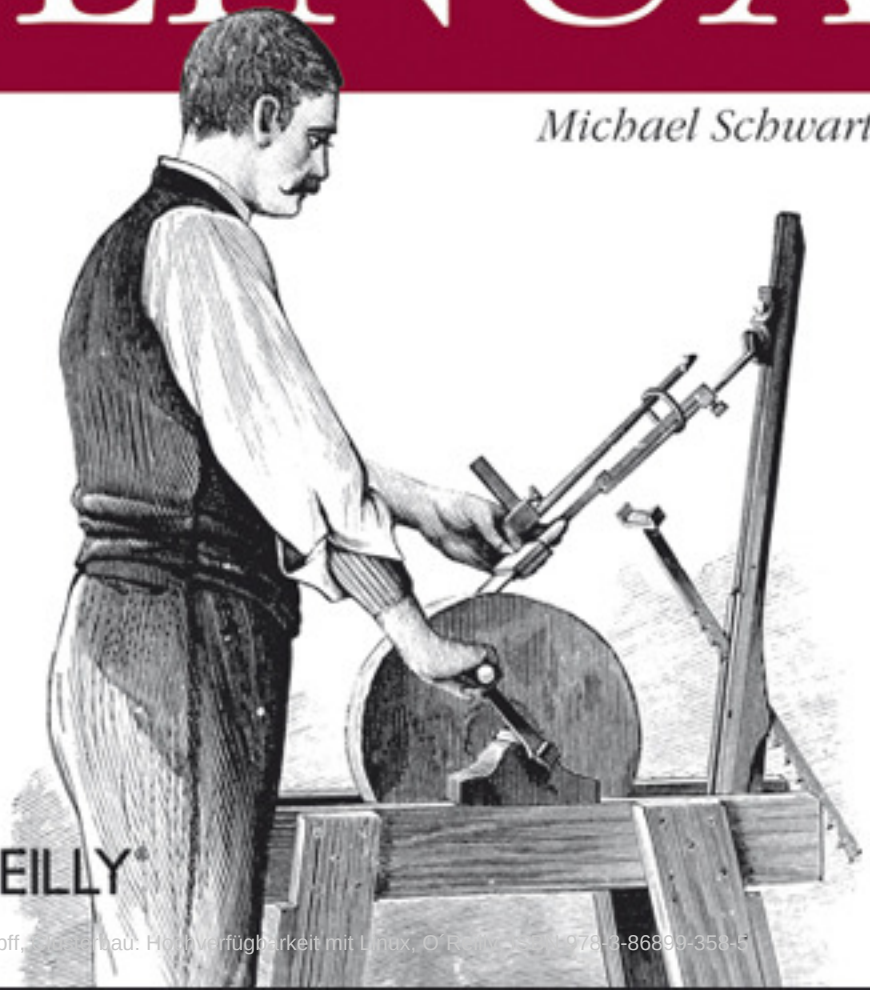
Hochverfügbare Linux-Server

3. Auflage

CLUSTERBAU: Hochverfügbarkeit mit LINUX

Michael Schwartzkopff



O'REILLY®

Inhalt

Michael Schwartzkopff, Clusterbau: Hochverfügbarkeit mit Linux, O´Reilly, ISBN 978-3-86899-358-5

KAPITEL 4

Ressourcen einrichten

In diesem Kapitel werden Sie die Konfiguration von Ressourcen und die Bedingungen, nach denen die Ressourcen auf die Knoten verteilt werden, kennenlernen. Dazu müssen wir uns allerdings erst einmal näher mit der Struktur der *Cluster Information Base* (CIB), der zentralen Verwaltung der Ressourcen, beschäftigen.

Der erste Teil dieses Kapitels soll erst einmal grundlegende Informationen zur Konfiguration eines Clusters bieten. Deshalb werden zunächst die einzelnen Elemente wie Ressourcen und Bedingungen eingeführt. Zu den einzelnen Elementen werden stets Beispiele gezeigt, um die Erklärungen verständlicher zu machen. Diese Beispiele sind immer nur Ausschnitte einer Gesamtkonfiguration und können deshalb in dieser Form nicht ausprobiert werden. Erst im nächsten Kapitel werden die Werkzeuge (Subshell, GUI und Kommandozeile) vorgestellt, um den Cluster tatsächlich zu konfigurieren. Dazu ist aber das Verständnis der Bausteine einer Clusterkonfiguration notwendig, sodass ich letztlich diesen Aufbau der Kapitel gewählt habe. Die einzelnen Beispiele des ersten Teils können dann wie Bausteine verwendet werden, um bei Bedarf in abgewandelter Form im eigenen Projekt eingebaut zu werden. Auch in Kapitel 9, *Beispielszenarien*, kommen sie nochmals zum Einsatz, um die Beispiele dort zu konfigurieren.

XML – die Sprache der CIB

Wie schon in Kapitel 2 im Abschnitt »Komponenten der Software« auf Seite 34 gesagt wurde, beschreibt die CIB den Zustand des Clusters. Dabei ist dort sowohl die Konfiguration hinterlegt, die der Administrator dem Cluster vorgegeben hat, als auch die Informationen über den aktuellen Status.

Die Sprache, in der die CIB die Informationen beschreibt, ist die *Extended Markup Language* (XML). Die Entwickler haben XML gewählt, da sie eine elegante Methode bietet, Informationen strukturiert darzustellen. Natürlich kann man eine Struktur in einfachen Konfigurationsdateien auf Textbasis ebenfalls darstellen, aber die Beziehungen der Elemente untereinander wären viel schwerer nachzuvollziehen als mit der vorgegebenen Struktur innerhalb von XML.

Im Laufe der Entwicklung hat sich dann aber gezeigt, dass diese Wahl die Konfiguration von Clustern doch unnötig erschwerte. Ab der pacemaker-Version 1.0.1 versteckte eine neue Kommandosprache, besser eine Subshell, die XML-Bausteine (Snipplets). Der Administrator kann dem Cluster seine Vorgaben also in einer relativ verständlichen und vor allem kompakten Konfigurationssprache mitteilen.

Dieses Buch stellt die meisten Beispiele in beiden Notationen vor. In der CRM-Notation kann man die Beispiele intuitiv verstehen und später schnell umsetzen. Das Verständnis der kompletten XML-Syntax ist meines Erachtens aber weiterhin notwendig, um tiefer liegende Probleme an der Wurzel zu packen und die Arbeitsweise des Clusters von Grund auf zu verstehen.

Innerhalb der CIB, also in XML-Kodierung, benötigen alle Elemente (Informationen) eine eindeutige »ID«. Die Werte der ID aller Elemente desselben Typs müssen einmalig sein, damit die CIB sie nicht verwechseln kann. Natürlich ist es gut, die IDs sogar global eindeutig zu vergeben, damit nie ein Problem auftritt. Die Subshell erzeugt diese IDs automatisch.

An vielen Stellen der CIB werden Variablen Werte zugeordnet. Hierfür benutzt die CIB die Name-Value-Zuweisung von XML:

```
<nvpair id="einmalige_ID" name="Name" value="Wert"/>
```

In einer Programmiersprache würde das etwa so ausgedrückt werden:

```
Name = "Wert"
```

Die CIB selbst ist in einen Abschnitt configuration und einen Abschnitt status unterteilt. Die configuration, also die Informationen über die Knoten, die Ressourcen und Bedingungen für die Ressourcen, werden vom Administrator vorgegeben. Die Informationen über den Status der Knoten und der Ressourcen auf den Knoten werden vom Cluster selbst über den *Local Resource Manager* (LRM) ermittelt und in der CIB gespeichert.

Nachdem die Software zum ersten Mal installiert und gestartet wurde, sieht die jungfräuliche CIB ungefähr so aus:

```
<cib ...>
  <configuration>
    <crm_config/>
    <nodes/>
    <resources/>
    <constraints/>
  </configuration>
  <status/>
</cib>
```

In den einzelnen Abschnitten sind keine Werte vorgegeben. Somit läuft der Cluster mit den eingestellten Vorgaben. Die Kunst der Clusterkonfiguration besteht jetzt darin, die einzelnen Abschnitte der CIB mit sinnvollen Werten zu füllen. Dabei bezieht sich der Abschnitt crm_config auf das Verhalten des gesamten Clusters. Im

Abschnitt nodes werden die einzelnen Knoten erfasst und im Abschnitt resources die konfigurierten Ressourcen. Unter constraints werden die Bedingungen vorgegeben, mit deren Hilfe der Cluster-Manager die Ressourcen den Wünschen des Administrators entsprechend auf den Knoten platzieren kann. Der Abschnitt status wird im Betrieb vom Cluster selbst aufgefüllt.

Der Cluster speichert die CIB auf jeden Knoten in der Datei */var/lib/heartbeat/crm/cib.xml*. Allerdings sollte der Administrator nie (in Worten: NIE!) diese Datei direkt editieren, sondern immer die Werkzeuge nutzen, die das Softwarepaket bereitstellt. Die einzelnen Kommandos werden in Kapitel 5 im Abschnitt »Die Befehle« auf Seite 128 beschrieben. Die GUI, die ebenfalls zur Verwaltung des Clusters genutzt werden kann, wird in Kapitel 5 im Abschnitt »Die GUI« auf Seite 112, beschrieben. Der Cluster speichert den Statusabschnitt der CIB auch nicht in der Datei ab, sondern hält diesen Teil im RAM, um Administratoren schon vom Versuch der Manipulation abzuhalten.

Die globalen Einstellungen der CIB

Im Abschnitt crm_config werden Einstellungen vorgegeben, die das Verhalten des gesamten Clusters betreffen. Alle Parameter sind im Abschnitt cluster_property_set zusammengefasst. Eine leere Konfiguration des Clusters sieht demnach so aus:

```
<crm_config>
  <cluster_property_set id="cib-bootstrap-options">
    (...)
  </cluster_property_set>
</crm_config>
```

Optionen werden als nvpair-Tags eingefügt.

Die Parameter, die an dieser Stelle eingegeben werden können, beeinflussen das Verhalten des *Cluster Resource Manager* und der *Policy Engine*. Die Optionen können deshalb auch in diese zwei Gruppen eingeteilt werden. Wie wir später sehen werden, gibt es diese Unterscheidung auch in der grafischen Benutzeroberfläche zur Konfiguration des Clusters. In der Subshell ist die Aufteilung nicht so klar, und alle Optionen sind im Abschnitt property der Konfiguration (configure) zusammengefasst.

Das Verhalten der *Policy Engine* bestimmen folgende Parameter:

no-quorum-policy
: Was passiert, wenn ein Knoten eines Teilclusters kein Quorum mehr hat? Dieser Wert bestimmt das Verhalten beim Verlust des Quorums.

  stop
  : Alle Ressourcen, die im Teilcluster ohne Quorum laufen, werden angehalten. Das Fencing ist deaktiviert. Diese Option ist der Vorgabewert.

`ignore` Der Verlust des Quorums wird ignoriert, und die Knoten des Clusters machen so weiter, als hätten sie das Quorum.

`freeze` Es werden keine Ressourcen neu gestartet, die sich bei Verlust des Quorums nicht im Teilcluster befunden haben. Ressourcen des Teilclusters können auf einen anderen Knoten im Teilcluster verschoben werden. Das Fencing ist deaktiviert.

`suicide` Diese Option sorgt dafür, dass ein Knoten ohne Quorum sich selbst aus dem Netz verabschiedet.

Während `heartbeat` weiß, dass für Cluster aus zwei Knoten die Definition eines Quorums sinnlos ist, und es deshalb nicht beachtet, hält sich `corosync` streng an die Vorschriften. Beim Ausfall eines Knotens hat der verbleibende kein Quorum mehr und hält auch alle Ressourcen an. Um in diesem Fall die Übernahme von Ressourcen zu ermöglichen, muss der Administrator den Parameter auf `ignore` setzen. Dasselbe gilt auch für Cluster, bei denen der Administrator erst einmal nur einen Knoten aufsetzt und sich dann wundert, dass auf diesem keine Ressourcen starten.

Am einfachsten lässt sich der Parameter mit dem `property`-Befehl aus dem Konfigurationsmenü der Subshell ändern. Auf der Kommandozeile kann man folgenden Befehl eingeben:

```
# crm configure property no-quorum-policy="ignore"
```

`symmetric-cluster`
: Vorgegeben ist hier `true`, also dass der Cluster symmetrisch ist. Bei dieser Einstellung dürfen die Ressourcen auf jedem Knoten laufen. Andernfalls muss zu jeder Ressource angegeben werden, auf welchem Knoten sie laufen soll. Üblicherweise bleibt dieser Wert auf `true`.

`default-resource-stickiness`
: Soll eine Ressource auf dem Knoten bleiben, auf dem sie aktuell läuft, oder soll der CRM einen Knoten suchen, der vielleicht »besser« für die Ressource ist? Der Hintergrund ist die Fragestellung, was besser ist: Ressourcen nach bestimmten Kriterien optimal auf den Knoten zu verteilen und eventuell den Umzug einer Ressource auf einen anderen Knoten zu riskieren oder alle Ressourcen möglichst auf dem Knoten laufen zu lassen, auf dem sie aktuell laufen, um keinen Verbindungsabbruch zu provozieren? Da die Entwickler diese Frage nicht für jeden Fall im Voraus beantworten können, gibt es diese globale Variable, die aber von den entsprechenden Parametern der einzelnen Ressourcen überschrieben werden kann. Folgende Werte können vorgegeben werden:

  `0` Die Ressource wird »optimal« im Cluster platziert. Dies heißt, dass Ressourcen neu im Cluster verteilt werden können, wenn ein »besserer« Knoten verfügbar wird oder ein Knoten, auf dem weniger Ressourcen laufen. Dieser Wert ist die Vorgabe, wenn nichts anderes eingegeben wird.

`> 0` Die Ressourcen haben die Tendenz, auf dem aktuellen Knoten zu verbleiben, können aber durchaus wechseln, wenn ein »besserer« Knoten verfügbar wird. Höhere Werte verstärken die Tendenz, auf dem aktuellen Knoten zu verbleiben.

`< 0` Ressourcen haben die Tendenz, vom aktuellen Knoten abzuwandern. Niedrigere Werte verstärken die Tendenz, den aktuellen Knoten zu verlassen. Diese Option ergibt nicht viel Sinn, ist aber aufgrund der Symmetrie doch vorhanden.

`INFINITY` Die Ressourcen bleiben auf dem aktuellen Knoten, bis sie gezwungen sind, diesen zu verlassen. Das geschieht immer dann, wenn auf dem Knoten keine Ressourcen mehr laufen können, weil entweder der Knoten herunterfährt, auf Stand-by geschaltet wird oder sich die Konfiguration ändert.

`-INFINITY` Die Ressourcen werden immer den aktuellen Knoten verlassen. Diese Einstellung ist nicht unbedingt sinnvoll.

Das genaue Konzept, wie der Cluster-Manager die Ressourcen auf die Knoten verteilt, finden Sie in Abschnitt »Bedingungen« auf Seite 80

Die Option `default-resource-stickiness` an dieser Stelle ist veraltet und sollte nicht mehr verwendet werden. Dieselbe Funktion erfüllt jetzt eine entsprechende Option an anderer Stelle. Mehr dazu finden Sie in Abschnitt »Globale Vorgaben« auf Seite 76.

`is-managed-default`

`true` Der CRM kümmert sich um das Starten und Stoppen von Ressourcen. Dieser Wert ist die Vorgabe.

`false` Der CRM kümmert sich nicht mehr um das Starten und Stoppen von Ressourcen. Falls eine Ressource lief, läuft sie weiter, wird aber bei einem Ausfall nicht auf einem anderen Knoten gestartet.

Wieder haben die Vorgaben für einzelne Ressourcen Priorität vor dieser globalen Vorgabe. Diese Option, speziell bei einer einzelnen Ressource angewandt, ist ideal, um diese Ressource vorübergehend aus dem Management des Clusters herauszulösen, beispielsweise zum Einspielen eines Patches.

Auch für diese Option gilt, dass die Konfiguration an dieser Stelle veraltet ist und nicht mehr verwendet werden sollte.

`maintenance-mode`

Diese Option ist eine verschärfte Variante von `is-managed-default`. Ein `true` an dieser Stelle, und der Cluster kümmert sich generell um gar nichts mehr. Neben

Start und Stopp werden auch alle laufenden Operationen, also zum Beispiel die Überwachung, für die Ressourcen eingestellt. Die Vorgabe ist hier natürlich `false`.

`start-failure-is-fatal`
: Soll ein Fehler beim Start einer Ressource als fataler Fehler bewertet werden (`true`), oder sollen die Fehlerzähler der einzelnen Ressourcen Vorrang haben? Wenn der Start einer Ressource fehlschlägt, ist es wahrscheinlich wenig sinnvoll, noch einen Start zu versuchen. Die Ressource erhält auf diesem Knoten `INFINITY` Fehler und kann hier nicht mehr ausgeführt werden. Sie wandert also auf einen anderen Knoten ab. Achtung: Diese Option bestimmt nur das Verhalten beim *Start* von Ressourcen. Ursprünglich war die Vorgabe `false`, inzwischen gilt aber `true`.

`stonith-enabled`
: Wenn diese Option auf `true` gesetzt wird, versucht der Cluster, fehlerhafte Knoten vom Netz und/oder der Stromversorgung zu trennen. Diese Option bedingt eine funktionierende Konfiguration einer STONITH-Ressource.

  Für diesen Parameter gilt die Vorgabe `true`. Falls man noch kein STONITH-Gerät konfiguriert hat oder zuerst den Cluster ohne diese Möglichkeit ausprobieren will, muss man die Konfiguration des Cluster-Managers auf `false` setzen. Andernfalls meckert der Cluster beständig das Fehlen dieser Geräte an, obwohl man ja offensichtlich die Technik einsetzen will. Die Option ändert man mit folgendem Befehl auf der Kommandozeile:

  ```
  # crm configure stonith-enabled="false"
  ```

Hat man im Laufe der Konfiguration des Clusters dann STONITH-Geräte eingebaut, darf man nicht vergessen, die Option wieder auf `true` zu setzen. Ansonsten funktioniert das Fencing nicht!

`stonith-action`
: Dieser Parameter gibt vor, was passiert, wenn einem Rechner der Strom abgedreht werden soll:

  `reboot` — Es wird versucht, einen fehlerhaften Knoten neu zu starten. Dieser Wert ist der Vorgabewert.

  `poweroff` — Der fehlerhafte Knoten wird abgeschaltet.

`stonith-timeout`
: Wie lange soll der Cluster auf die Rückmeldung einer STONITH-Aktion warten? Die Vorgabe ist hier `60` Sekunden.

`startup-fencing`

  `true` — Knoten, die der Cluster nicht kennt, werden ausgegrenzt (engl.: *fenced*). Dies ist der Vorgabewert.

`false` Nur Knoten, die Mitglied des Clusters waren, also mit dem DC kommuniziert hatten und dann aus dem Cluster ausschieden, werden ausgegrenzt.

`cluster-delay`
: Die Signallaufzeit im Netz, die Sie am einfachsten mit `ping` ermitteln. Der »richtige« Wert für jeden einzelnen Cluster hängt von der Geschwindigkeit im Netzwerk und der Last auf den Knoten ab. Die Vorgabe ist `60` Sekunden. Der Wert kann an die aktuellen Gegebenheiten angepasst werden.

`batch-limit`
: Gibt die Anzahl der Jobs an, die die *Transition Engine* (ein Teil des CRM) maximal gleichzeitig ausführen darf, also zum Beispiel Ressourcen starten oder anhalten. Diese Option dient der Begrenzung der Last auf den Knoten. Die Vorgabe ist 30.

`default-action-timeout`
: Dieser Wert gibt einen globalen Timeout für jede Aktion an. Hat sich eine Aktion innerhalb dieser Zeit nicht erfolgreich zurückgemeldet, gilt die Aktion als gescheitert. Andere Werte, die bei speziellen Aktionen eingestellt sind, gehen natürlich vor. Der vorgegebene Wert ist `20` Sekunden.

`stop-all-resources`
: Ein `true` bei dieser Option stoppt sofort alle Ressourcen bis auf die, die fürs Fencing benötigt werden. Die Vorgabe ist natürlich `false`.

`stop-orphan-resources`
: Dieser Parameter bestimmt das Verhalten des CRM, wenn Ressourcen vorgefunden werden, zu denen aber keine Definition mehr existiert. Dies kann vorkommen, wenn eine der Ressourcen gelöscht wird, ohne dass sie vorher angehalten wurde. Gerade bei der Nutzung der GUI zum Experimentieren kann dieser Fehler schnell passieren.

  `true` Die Ressource wird angehalten. Dieses Verhalten entspricht der Vorgabe.

  `false` Die Ressource wird ignoriert.

`stop-orphan-actions`
: Dieser Parameter bestimmt das Verhalten des CRMs, wenn eine Aktion vorgefunden wird, zu der keine Definition existiert. Dies kann bei der Änderung des Intervalls für eine wiederkehrende Aktion vorkommen.

  `true` Aktion anhalten. Vorgabewert.

  `false` Aktion ignorieren.

`remove-after-stop`
: Sollen die Ressourcen, die angehalten werden, aus dem *Local Resource Manager* (LRM), also dem Teil, der die Ressourcen lokal auf den einzelnen Knoten verwaltet, gelöscht werden? Vorgabe und der einzig sinnvolle Wert ist `false`.

Somit erinnert sich der LRM daran, dass die Ressource lokal angehalten ist. Der Wert `true` ist bestenfalls wenig getestet und potenziell gefährlich.

`pe-error-series-max`
: Die maximale Anzahl von Fehlern, die beim Fehlersuchen gespeichert werden. Diese Option begrenzt den Zähler für Fehler bei der Verarbeitung von Daten durch die *Policy Engine* (PE). Ein Wert von `0` schaltet den Zähler ab, und die Vorgabe `-1` erlaubt dem Zähler, unendlich viele Fehler zu protokollieren.

`pe-warn-series-max`
: Analog zum Fehlerzähler, allerdings für Warnungen bei der Verarbeitung von Eingaben durch die PE.

`pe-input-series-max`
: Die Größe des Speichers für »normale« Vorgänge, die durch die PE verarbeitet wurden. Auch dieser Zähler kommt bei Fehlerberichten zum Einsatz. Vorgabe ist hier ebenfalls wieder `-1`.

`node-health-strategy`
: Mit dieser und den folgenden Optionen erhält der Administrator die Möglichkeit, zu bestimmen, was der Cluster machen soll, wenn sich ein Problem abzeichnet oder schon eingetreten ist. Die Vorgabe `none` bestimmt, dass diese Möglichkeit erst einmal nicht genutzt wird. Alle Optionen dieses Parameters und die Anwendung werden weiter unten im Abschnitt »Systemgesundheit« auf Seite 109 besprochen.

`node-health-green`, `node-health-yellow` *und* `node-health-red`
: Mit diesen Einstellungen kann der Administrator die Reaktion des Clusters auf Probleme sehr fein abstimmen. Mehr dazu finden Sie im Abschnitt »Systemgesundheit« auf Seite 109.

`placement-strategy`
: Diese Option gibt die Strategie für die Platzierung von Ressourcen im Cluster vor. Bei der Vorgabe (`default`) entscheidet der `CRM` aufgrund der Bedingungen, wo was laufen soll. Daneben kann der Administrator die Auslastung von Knoten (`utilization`) in die Entscheidung mit einbeziehen. Details dazu gibt es auch in Kapitel 5 im Abschnitt »Utilization« auf Seite 171. Die erlaubten Optionen sind:

  `utilization` — Der Cluster entscheidet, welche Ressourcen auf welchem Knoten laufen, und bezieht in diese Entscheidung auch den Ressourcenverbrauch von Ressourcen mit ein. Falls genügend Knoten zur Verfügung stehen, die die notwendigen Ressourcen bieten, werden die Clusterressourcen entsprechend ihrer Anzahl gleichmäßig auf die Knoten verteilt.

  `minimal` — Der Cluster versucht, die Ressourcen auf möglichst wenig Knoten zu verteilen, sodass aber trotzdem kein Knoten überlastet wird.

`balanced` Der Cluster verteilt die Clusterressourcen gleichmäßig auf alle Knoten. In die Entscheidung geht aber auch der Ressourcenverbrauch von Clusterressourcen ein, sodass die Last gleichmäßig auf alle Knoten verteilt wird.

Folgende Optionen finden sich ebenfalls in dem Abschnitt `crm_config`. Sie bestimmen das Verhalten des *Cluster Resource Manager*-Diensts `crmd`. Änderungen an dieser Stelle sind aber nur dem fortgeschrittenen Benutzer oder den Entwicklern empfohlen. Also Vorsicht bei Änderungen.

`dc-version`
: Version von `pacemaker` inklusive des Hashwerts, der die exakte Version des Sourcecodes identifiziert. Diese Option wird nur zur Diagnose benutzt.

`cluster-infrastructure`
: Die Information, ob `heartbeat` oder `OpenAIS` bzw. `corosync` für die Kommunikation benutzt wird, dient ebenfalls nur der Information und Diagnose.

`dc-deadtime`
: Wie lange soll ein Knoten beim Systemstart auf eine Rückmeldung von möglicherweise schon existierenden Knoten im Cluster warten? Der »richtige« Wert hängt von der Geschwindigkeit und den Lastbedingungen im Netzwerk ab. Die Vorgabe ist `60` Sekunden.

`cluster-recheck-interval`
: Üblicherweise werden Änderungen im Cluster ereignisgesteuert verbreitet. Einige zeit-basierende Änderungen werden durch diesen Mechanismus eventuell nicht erfasst. Dieses Intervall gibt an, wie oft die Knoten ein virtuelles Ereignis auslösen. Die Vorgabe ist hier `15` Minuten.

`shutdown-escalation`
: Wenn der Cluster herunterfährt, kann es passieren, dass eine Ressource sich nicht ordnungsgemäß verabschiedet. Dieser Parameter gibt an, wie lange der Cluster warten soll, bis der Dienst hart heruntergefahren wird. Die Vorgabe ist hier `20` Minuten.

`expected-quorum-votes`
: Dieser Wert wird zur Berechnung des Quorums genutzt, wenn `corosync` für die Kommunikation im Cluster sorgt. Der Wert ist die Anzahl der erwarteten Stimmen. Die Vorgabe ist `2`, wird aber automatisch an die Anzahl der erkannten Knoten angepasst.

Die restlichen Optionen `election-timeout`, `crmd-integration-timeout` und `crmd-finalization-timeout` sind nicht dokumentiert und eigentlich nur zur Nutzung durch die Entwickler gedacht.

Knoten in der CIB

Die Informationen über die Knoten werden vom CRM automatisch in den Abschnitt über die Konfiguration der Knoten der CIB eingetragen. Der Administrator braucht sich um nichts zu kümmern. Ein Beispiel für diesen Abschnitt der CIB sieht wie folgt aus:

```
<nodes>
  <node uname="node1" type="normal" id="node-node1"/>
  <node uname="node2" type="normal" id="node-node2"/>
</nodes>
```

In der Notation der Subshell kann man sich die Information über die Knoten wie folgt anzeigen lassen:

```
# crm node show
node1: normal
node2: normal
```

Beim Start des Clusters werden die Informationen über die Knoten automatisch erzeugt. Meldet sich ein neuer Knoten beim *Designated Coordinator* (DC) an, werden diese neuen Informationen automatisch an dieser Stelle der CIB hinterlegt.

Falls ein Administrator ein System mit `heartbeat` aufsetzt, zu dem eventuell später neue Knoten hinzugefügt werden sollen, ist es ratsam, von Beginn an nicht die Knoten fest vorzugeben, sondern die dynamische Erweiterungsmöglichkeit mit der Option `node autojoin` innerhalb von *ha.cf* zu nutzen.

Jeder der Knoten besitzt seine eigenen Attribute (siehe `instance_attributes` oben), die wiederum mit beliebigen Attributen (`nvpair`) beschrieben werden können. Da Werte mithilfe der Werkzeuge von `pacemaker` in die CIB eingefügt werden können, ist es also denkbar, Entscheidungen innerhalb des Clusters aufgrund solcher Attribute zu fällen. Denkbar sind dabei Angaben wie »Angebunden ans SAN« oder »Prozessorlast«. Wie das genau funktioniert, wird im Abschnitt »Bedingungen für Fortgeschrittene« auf Seite 95 gezeigt. Allerdings wird dort der dynamische Abschnitt `status` der CIB genutzt und nicht die relativ starre Konfiguration der Knoten, die ja erst vom CRM angelegt wird und anschließend vom Administrator ergänzt werden kann.

Einfache Ressourcen

Im nächsten Abschnitt der CIB werden die Ressourcen eingetragen. Leider erledigt das der `crmd` nicht automatisch, sondern der Administrator ist gefragt, hier seine Vorstellungen einzugeben.

Ein pacemaker-Cluster kann viele verschiedene Arten von Ressourcen verwalten. Als Ressource wird hierbei alles definiert, was von der Clustersoftware kontrolliert, also gestartet, überwacht und angehalten wird. Die Definition von »Ressource« beschränkt sich hierbei nicht auf einen klassischen Dienst wie einen Webserver, sondern auf alles, was vom CRM kontrolliert wird. Das können (virtuelle) IP-Adressen des Clusters, Dienste, Dateisysteme, Überwachungsfunktionen oder kurz alles sein, was sich über ein Skript programmieren lässt.

Diese Skripte, die sogenannten *Resource Agents*, sind die Verbindungen zwischen dem CRM und der eigentlichen Ressource, zum Beispiel einem Webserver. Der Cluster übernimmt hierbei die Funktion des `init`-Prozesses, der normalerweise die Dienste startet. Mehr zu den verschiedenen Klassen der Skripte und zu den Skripten selbst folgt in Kapitel 8, *Agenten*.

Einfache Ressourcen werden im Cluster als »primitive« Ressourcen bezeichnet, um den Unterschied zu komplexeren Ressourcen wie zum Beispiel Gruppen zu verdeutlichen. Somit sieht die Definition einer solchen einfachen Ressource wie folgt aus:

```
primitive ID type:provider:class meta … params … op …
```

bzw. in XML-Notation:

```
<primitive id="ID" class="class" provider="provider" type="type">
  <meta_attributes" id="ID-meta_attributes"/>
  <instance_attributes id="ID-instance_attributes"/>
  <operations id="ID-operations"/>
</primitive>
```

Wie beim Programmieren üblich, sollte man für die IDs einfache, verständliche Namen wählen, anhand deren die Ressourcen schnell zu identifizieren sind. Auch sollte man innerhalb einer Installation eines Clusters konsequent ein einheitliches Schema nutzen. In diesem Buch werden IDs für Ressourcen mit `resService` bezeichnet. Im Weiteren wird dieses Schema entsprechend erweitert.

Die Klasse des Agenten (`class`) kann `ocf`, `lsb`, `heartbeat` oder `stonith` sein. `ocf` verweist auf einen Agenten, der kompatibel zu den Vorgaben des *Open Clustering Framework* ist, `lsb` dementsprechend auf einen Agenten, der den Ansprüchen der *Linux Standards Base* genügt, `heartbeat` auf einen Agenten entsprechend der Version 1 von Linux-HA und `stonith` auf eine Schnittstelle zu einem ebensolchen Gerät. Näheres zu den einzelnen Klassen der Agenten finden Sie in Kapitel 8, *Agenten*.

Der Typ der Ressource (`type`) entspricht dem Namen des Skripts des Agenten, der zum Einsatz kommt. Will der Administrator einen MySQL-Server starten, lautet der Typ einfach `mysql`. Bei `init`-Skripten entspricht der Typ zum Beispiel dem Namen des Skripts in */etc/init.d*.

Der Provider einer Ressource (nur bei OCF-Ressourcen!) bezeichnet unterschiedliche Hersteller von Agenten. Moderne Versionen des Pakets `resource-agents` ken-

nen zum Beispiel die Hersteller (provider) heartbeat und rgmanger. Das Paket pacemaker stellt einige Agenten unter dem Provider pacemaker zur Verfügung.

Ein vollständiges Beispiel für eine Ressource wird weiter unten gezeigt, wenn alle Teile einer Ressource besprochen werden.

Attribute

Ebenso wie die Attribute für den gesamten Cluster im Abschnitt crm_config gesetzt werden, kann man im Abschnitt meta_attributes (CRM-Notation: meta) Attribute für einzelne Ressourcen setzen. Sie bestimmen das Verhalten von Ressourcen im Cluster. Die Attribute der Ressourcen haben Vorrang vor den vererbten Werten des Clusters. Solche vorrangigen Attribute sind zum Beispiel is-managed oder resource-stickiness. Sie haben innerhalb der einzelnen Ressource die Bedeutung, die die Vorgaben im Gesamtcluster haben. Zusätzlich zu den beiden kann man für die einzelne Ressource noch die folgenden Attribute setzen:

priority
: Mithilfe dieser Ganzzahl entscheidet der Cluster, welche Ressourcen wichtiger sind und deshalb in einem Cluster starten dürfen, in dem es nicht genug Knoten für alle Ressourcen gibt. Eigentlich sollte man den Cluster so dimensionieren, dass dieses Attribut nie zum Einsatz kommt.

  Man kann sich aber durchaus Szenarien für den Einsatz der Priorität in Clustern vorstellen, in denen ein Test- und ein Produktivsystem konfiguriert sind. Beide Instanzen der Software würden sich auf einem Knoten nicht vertragen. Verlässt ein Knoten den Cluster, soll die produktive Instanz weiterlaufen, und das Testsystem muss notfalls ausgeschaltet werden. Das erreicht der Administrator, wenn er die Ressource für die produktive Instanz mit einer höheren Priorität konfiguriert.

target-role
: Dieser Parameter bestimmt, ob und in welchem Modus diese Ressource läuft. Wenn nichts gesetzt ist, gilt die Vorgabe Started. Mehr zu diesem Attribut finden Sie bei komplexen Ressourcen (Abschnitt »Ressourcen für Fortgeschrittene« auf Seite 88).

migration-threshold
: Mit dieser Ganzzahl und einem Fehlerzähler entschiedet der Cluster, ob eine Ressource noch auf dem aktuellen Knoten laufen darf oder auf einen anderen verschoben wird. Näheres dazu finden Sie im Abschnitt »Bedingungen für Fortgeschrittene« auf Seite 95.

multiple-active
: Mit diesem Attribut kann der Administrator steuern, was der Cluster unternehmen soll, wenn er entdeckt, dass eine Ressource mehr als einmal im Cluster läuft. Mögliche Werte sind:

`block` Nichts unternehmen. Der Administrator regelt alles.

`stop_only` Alle Instanzen der Ressource anhalten.

`stop_start` Die Vorgabe ist am sinnvollsten: alle Instanzen der Ressource anhalten und anschließend nur auf einem Knoten unter Kontrolle des Clusters wieder starten.

Die Situation, dass der Cluster-Manager mehrere Instanzen einer Ressource im Cluster entdeckt, kann zum Beispiel entstehen, wenn der Administrator die Ressource sowohl im Cluster also auch über das normale `init`-System startet. Oder anders ausgedrückt: Wenn `init` den Webserver startet, der auf einem anderen Knoten schon unter Kontrolle des Clusters läuft, merkt der Cluster das normalerweise und reagiert entsprechend der Option.

Deshalb ist es wichtig, die Ressourcen, die im Cluster laufen sollen, nicht durch `init` kontrollieren zu lassen. Ein

```
# chkconfig --del Ressource
```

reicht dafür in den meisten Fällen.

`failure-timeout`

Dieser Parameter bestimmt, nach welcher Zeit der Cluster Fehler wieder vergisst. Mit dieser Option bekommt der Cluster sozusagen die Fähigkeit, »selbstheilend« zu arbeiten. Sie sollte mit Vorsicht eingesetzt werden – die Vorgabe ist deshalb auch `never`. Nützlich ist ihr Einsatz beispielsweise in automatisierten Clustern.

Zur Definition einer Ressource sind von den Attributen nur die ID, die Klasse und der Typ notwendig. Alle anderen sind entweder nicht notwendig oder werden aus den globalen Einstellungen vererbt.

Die Attribute, die die globalen Einstellungen des Clusters überschreiben oder das Verhalten einer einzelnen Ressource bestimmen, erscheinen im Abschnitt `meta_attributes` dieser Ressource. Daneben können noch beliebige weitere `instance_attributes` (in CRM-Notation: `params`) in einem weiteren Abschnitt der Ressource definiert werden. Notwendig ist dies bei Ressourcen nach OCF-Standard. Dort werden diese Attribute der einzelnen Instanz als Parameter an den entsprechenden *Resource Agent* weitergegeben. Zum Beispiel benötigt der OCF-Agent für eine IP-Adresse eben diese Adresse als Parameter, und die muss der Administrator als Variable der Ressource frei vergeben können. Somit kann die Konfiguration von OCF-Ressourcen über die Attribute eingegeben werden und wird nicht, wie bei LSB-Ressourcen, in Konfigurationsdateien hinterlegt. Der Cluster hat deshalb auch die volle Kontrolle über die Parameter der Ressourcen. Andernfalls muss der Administrator dafür sorgen, dass zum Beispiel diese Dateien auf allen Knoten identisch sind. Die OCF-Agenten bringen also hier eine wesentliche Erleichterung für den Administrator, da der Cluster die CIB ja automatisch im gesamten Cluster repliziert.

Als Beispiel sehen Sie hier die Definition einer IP-Adresse 192.168.188.95 als primitive OCF-Ressource, zuerst in Notation für die crm-Subshell:

```
primitive resIP ocf:heartbeat:IPaddr2 meta target-role="stopped" \
  params ip="192.168.188.95"
```

oder in der ausführlichen XML-Notation:

```
<primitive id="resIP" class="ocf" provider="heartbeat" type="IPaddr2">
  <meta_attributes id="resIP-meta_attributes">
    <nvpair id="resIP-meta_attributes-target-role" name="target-role"
      value="stopped"/>
  </meta_attributes>
  <instance_attributes id="resIP-instance_attributes">
    <nvpair id="resIP-ip" name="ip" value="192.168.188.95"/>
  </instance_attributes>
</primitive>
```

Neben dem Instance-Attribut für die Adresse selbst gibt es noch ein Meta-Attribut target-role, das angibt, ob die Ressource gerade läuft oder, wie hier, angehalten ist.

Näheres zu den Parametern und zur Möglichkeit, diese in Agenten zu nutzen, finden Sie in Kapitel 8, *Agenten*. Die obige Konfiguration kann erweitert werden, wenn zusätzlich noch ein Webserver eingegeben wird. Dazu wird eine zweite primitive Ressource eingerichtet. Zum besseren Verständnis wird der komplette Ressourcenabschnitt der CIB mit beiden Ressourcen dargestellt:

```
<primitive id="resIP" class="ocf" provider="heartbeat" type="IPaddr2">
  <meta_attributes id="resIP-meta_attributes">
    <nvpair id="resIP-meta_attributes-target-role" name="target-role"
      value="started"/>
  </meta_attributes>
  <instance_attributes id="resIP-instance_attributes">
    <nvpair id="resIP-ip" name="ip" value="192.168.188.95"/>
  </instance_attributes>
</primitive>
<primitive id="resApache" class="ocf" provider="heartbeat" type="apache">
  <instance_attributes id="resApache_instance_attrs">
    <nvpair id="resApache-conf" name="configfile"
      value="/etc/apache2/httpd.conf"/>
    <nvpair id="resApache-httpd" name="httpd" value="/usr/sbin/httpd"/>
  </instance_attributes>
</primitive>
```

oder in CRM-Notation:

```
primitve resIP ocf:heartbeat:IPaddr2 params ip="192.168.188.95"
primitive resApache ocf:heartbeat:apache \
  params configfile="/etc/apache2/httpd.conf" httpd="/usr/sbin/httpd"
```

Der Apache-Ressource werden zwei Parameter (die Konfigurationsdatei und die ausführbare Datei des Webservers) übergeben. Die Ressource IP-Adresse läuft bei dieser Konfiguration und ist nicht wie zuvor angehalten. Da nichts Gegenteiliges angegeben ist, läuft der Webserver ebenfalls.

Die Ressourcen in der CRM-Notation kann der Administrator im Konfigurationsteil der Subshell eingeben. Diesen ruft er mit

```
# crm configure
```

auf. Der neue Prompt

```
crm(live) configure#
```

weist auf die neue Umgebung hin. An dieser Stelle kann der Administrator die gewünschten Ressourcen eingeben. Eine Ressource muss auf einer einzigen Zeile eingegeben werden. Wenn der Administrator der Übersichtlichkeit halber doch auf eine neue Zeile wechseln will, kann er mit dem Zeichen »\« der Subshell mitteilen, dass nach dem CR-LF noch mehr kommt, das zu der Zeile gehört.

Nach der Eingabe der Ressource übernimmt der Cluster die Änderungen nicht sofort. Der Administrator muss sie erst mit einem `commit` bestätigen. Das gibt einerseits dem Cluster die Chance, alle Änderungen im Zusammenhang auf korrekte Syntax zu prüfen, und andererseits dem Administrator die Gelegenheit, allzu unbedachte Änderungen mit einem einfachen `exit` nicht wirksam werden zu lassen. Der Cluster fragt dann nach, ob man die Änderungen wirklich verwerfen will:

```
# crm configure
crm(live) configure# primitive resDummy ocf:pacemaker:Dummy
crm(live) configure# exit
There are changes pendig. Do you want commit them? n
bye
```

Ist eine Ressource erst einmal angelegt, startet der Cluster diese auch sofort, sofern kein Meta-Attribut `target-role="Stopped"` etwas anderes verlangt. Die Subshell können Sie übrigens jederzeit mit dem Befehl `exit` verlassen.

Die Subshell bietet mit dem Menüpunkt `resource` die Möglichkeit, den Status von Ressourcen direkt zu beeinflussen. Mit den Befehlen

```
# crm
crm(live)# resource
crm(live) resource# stop resDummy
crm(live) resource# cd
crm(live)#
```

hält der Administrator eine Ressource mit der ID `resDummy` an und wechselt in das Hauptmenü der Subshell zurück. Nachdem die Ressource angehalten ist, lässt die Subshell es auch zu, die Ressource wieder zu löschen. Dazu wechselt der Administrator erneut zur Konfiguration:

```
crm(live)# configure
crm(live) configure# delete resDummy
crm(live) configure# commit
crm(live) configure# exit
```

Anhand dieses kurzen Beispiels haben Sie gelernt, die Ressouren von oben (IP-Addresse und Webserver) selbst einzurichten und notfalls auch wieder zu löschen.

Globale Vorgaben

Wie oben schon erwähnt, haben die Entwickler die Konfiguration einiger Attribute aus dem Abschnitt `crm_config` herausgelöst und einen eigenen Abschnitt hierfür angelegt. Das hat keine technischen Gründe, die Konfiguration strukturiert sich allerdings besser und wird deshalb logischer.

Alle Vorgaben für Attribute, die das Verhalten von Ressourcen im Cluster bestimmen, also für die Meta-Attribute, liegen jetzt im Abschnitt `rsc_defaults`, einem direkten Unterabschnitt der `configuration`. Die Vorgaben für das Verhalten von Ressourcen im Cluster sollte der Administrator also in diesem Abschnitt eingegeben haben und nicht mehr in der Gesamtkonfiguration des Clusters.

Neben der besseren Strukturierung der CIB liegt ein weiterer Vorteil darin, dass die Attribute im Abschnitt `rsc_defaults` den gleichen Namen haben wie das entsprechende Attribut einer Ressource. Die Vorgabe für die Stickiness heißt jetzt eben `resource-stickiness` im Abschnitt `rsc_defaults` und nicht mehr `default-resource-stickiness` im Abschnitt `crm_config`.

Die Vorgaben für die Operationen (siehe nächster Abschnitt) liegen entsprechend im Abschnitt `op_defaults`.

Operationen

Für Ressourcen können (und sollten!) auch sogenannte Operationen definiert werden. Mögliche Konfigurationen sind alle Aktionen, die ein Agent zulässt. Möglich sind deshalb unter anderem `start`, `stop` und `monitor`.

Ein `monitor`, das für eine LSB-Ressource definiert ist, ist kein Beinbruch, da `pacemaker` weiß, dass der *Resource Agent* den Aufruf nicht verstehen wird und ihn automatisch in eine `status`-Abfrage umwandelt.

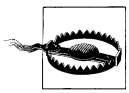
Diese Operationen werden zum Beispiel eingesetzt, um den Zustand einer Ressource zu kontrollieren. Abhängig davon, wie gut der `status`- oder `monitor`-Befehl im Agenten implementiert ist, kann festgestellt werden, ob der Webserver tatsächlich noch die richtige Seite ausliefert oder die Datenbank noch sinnvolle Ergebnisse liefert. Je nach Rückgabewert der Operation kann der Cluster-Manager bei Bedarf die Ressource auf einem anderen Knoten starten.

Diese konfigurierten Aktionen werden, kontrolliert von der Clustersoftware, in regelmäßigen Zeitabschnitten (Attribut `interval`) durchgeführt, in denen das Agentenskript mit diesem Parameter aufgerufen wird. Die Zeitabstände kann der Administrator natürlich frei wählen. Auch kann der Administrator angeben, unter welchen Voraussetzungen (Attribut `requires`) die Operation überhaupt durchgeführt wird:

`nothing`
: Es gibt keine speziellen Voraussetzungen.

`quorum`
: Nur dann, wenn der Teilcluster `quorum` hat.

`fencing`
: Nur dann, wenn der Teilcluster `quorum` hat und alle `fencing`-Aktionen vollständig ausgeführt wurden.



Es können keine zwei Operationen einer Ressource im selben Zeitintervall konfiguriert werden! Zwei Operationen derselben Ressource müssen immer unterschiedliche Intervalle haben.

Weiterhin kann der Administrator konfigurieren, in welchem Zustand sich die Ressource befinden soll, damit der Test durchgeführt wird. Das Attribut `role` kann dabei die Werte `Master`, `Slave`, `Started` oder `Stopped` annehmen. Die Vorgabe ist hierbei `Started`, das heißt, die Operation wird nur durchgeführt, wenn die Ressource gestartet ist. `Master` und `Slave` beziehen sich auf Zustände, die für Ressourcen gelten, die in diesen Zwischenstufen existieren können. Mehr dazu folgt bei der Beschreibung dieser Ressourcen selbst (Abschnitt »Ressourcen für Fortgeschrittene« auf Seite 88).

Neben dem Attribut für die Wiederholfrequenz (`interval`) der Operation sind auch die Zeit (Attribut `timeout`), nach der der CRM annimmt, dass die Operation fehlgeschlagen ist, und ein Attribut für die Verzögerung nach dem Start der Ressource, bevor zum ersten Mal die Operation durchgeführt wird (`start-delay`), anzugeben. Über diesen Wert kann der Administrator beeinflussen, wie viel Zeit der Ressource bleibt, um interne Angelegenheiten zu regeln, bevor sie auf den Zustand hin überprüft wird. Ein `start-delay` ist in den meisten Fällen falsch und tatsächlich nur dann notwendig, wenn eine Überwachung einer Ressource augenblicklich nach dem Start der Ressource fehlschlagen würde. Das ist zum Beispiel der Fall, wenn der *Resource Agent* den Start der Ressource nur als Hintergrundprozess anstößt und sich sofort mit einer Erfolgsmeldung zurückmeldet. Überprüft der Cluster nun den Status der Ressource, ohne dass der Hintergrundprozess seine Arbeit beendet hat, führt das zu einem Fehler. Die Überwachung muss in solch einem Fall die Zeitspanne `start-delay` warten, bevor sie das erste Mal ausgeführt werden kann.

Natürlich ist es auch wichtig, die Aktion anzugeben, die durchgeführt wird, wenn die Operation fehlschlägt (Attribut `on-fail`). Hier stehen folgende Werte zur Auswahl:

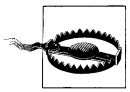
`ignore`
: Der Cluster soll annehmen, dass die Operation nicht fehlgeschlagen ist.

`block`
: Nichts unternehmen. Der Administrator muss sich um das Problem kümmern.

`restart`
: Die Ressource wird angehalten, und es wird versucht, die Ressource neu zu starten. Zusätzlich wird ein interner Fehlerzähler hochgesetzt. Diese Option ist auch die Vorgabe.

`stop`
: Die Ressource wird angehalten und nicht auf einem anderen Knoten wieder gestartet.

`fence`
: Der komplette Knoten, auf dem die Ressource lief, wird ausgegrenzt. Alle anderen Ressourcen, die auf dem Knoten laufen, werden ebenfalls gestoppt und auf anderen Knoten wieder gestartet.



Ohne eine kontinuierliche Überprüfung der Funktion der Ressource kann eine hohe Verfügbarkeit nicht garantiert werden. Der Cluster kann nur auf den Ausfall einer Ressource reagieren, wenn er Kenntnis vom Ausfall erhält. Ohne eine `monitor`-Operation kann eine Ressource also nicht hochverfügbar realisiert werden.

Für eine Überwachung des Webservers müsste das obige Beispiel um den Abschnitt `operations` erweitert werden:

```
<primitive id="resApache" class="ocf" provider="heartbeat" type="apache">
  <instance_attributes id="resApache_instance_attrs">
    <nvpair id="resApache-conf" name="configfile"
      value="/etc/apache2/httpd.conf"/>
    <nvpair id="resApache-httpd" name="httpd" value="/usr/sbin/httpd"/>
  </instance_attributes>
  <operations>
    <op id="opApache" name="monitor" interval="60s" timeout="20s"
      on-fail="restart"/>
  </operations>
</primitive>
```

oder wiederum in CRM-Notation:

```
primitive resApache ocf:heartbeat:apache \
  params configfile="/etc/apache2/httpd.conf" httpd="/usr/sbin/httpd" \
  op monitor interval="60s"
```

Falls eine Ressource länger zum Starten braucht als die Vorgabe des `timeout` von 20 Sekunden, muss der Administrator dieses in der Konfiguration berücksichtigen. Eine größere MySQL-Datenbank lässt sich zum Beispiel so konfigurieren:

```
primitve resMySQL ocf:heartbeat:mysql \
  op monitor interval="20s" timeout="30s" \
  op start timeout="120s" \
  op stop timeout="120s"
```

Natürlich wissen die Agenten selbst am besten darüber Bescheid, wie lange die Operationen üblicherweise dauern und welche Werte empfohlen werden. Diese Ratschläge finden sich am Ende der Beschreibung des Agenten. Ausgeben kann der Administrator diese mit dem Kommando:

```
# crm ra info class:provider:type
```

Ein weiteres Beispiel für eine Ressource mit der Definition einer `monitor`-Operation ist die Ressource `SysInfo`:

```
primitive resSysinfo ocf:pacmaker:SysInfo op monitor interval="60s"
```

oder in XML:

```
<primitive id="resSysinfo" class="ocf" provider="pacemaker" type="SysInfo">
  <operations>
    <op id="opSysinfo" name="monitor" interval="60s" on_fail="restart"/>
  </operations>
</primitive>
```

Bei einem Aufruf mit der Aktion `monitor` werden bestimmte Systemparameter (freier Plattenplatz auf der *root*-Partition oder die Last der CPU) in den Statusabschnitt der CIB geschrieben. Mit der obigen Definition werden diese Informationen also jede Minute erneuert. Dies geschieht für den Knoten, auf dem diese Ressource aktuell läuft. Der entsprechende Abschnitt würde dann zum Beispiel so aussehen:

```
<node_state uname="xen3" ha="active" in_ccm="true" crmd="online" join="member"
  id="xen3" shutdown="0" expected="member" crm-debug-origin="do_update_resource">
  (…)
  <transient_attributes id="transient-cbb68cf2">
    <instance_attributes id="inst_attr_status">
      <nvpair id="complete" name="probe_complete" value="true"/>
      <nvpair id="arch" name="arch" value="i686"/>
      <nvpair id="os" name="os" value="Linux-2.6.28-1-686"/>
      <nvpair id="free_swap" name="free_swap" value="1000"/>
      <nvpair id="cpu_info" name="cpu_info" value="AMD Sempron(tm)
        Processor 3200+"/>
      <nvpair id="cpu_speed" name="cpu_speed" value="3616.52"/>
      <nvpair id="cpu_cores" name="cpu_cores" value="1"/>
      <nvpair id="cpu_load" name="cpu_load" value="0.04"/>
      <nvpair id="ram_total" name="ram_total" value="950"/>
      <nvpair id="ram_free" name="ram_free" value="300"/>
      <nvpair id="root_free" name="root_free" value="14"/>
    </instance_attributes>
  </transient_attributes>
</node_state>
```

Man erkennt, dass sich der Knoten gerade langweilt (`cpu_load="0.04"`) und dass auch noch genügend Platz auf der *root*-Partition ist (`root_free="14"` [GByte]). Die Ressource `Sysinfo` und die Interpretation der Ergebnisse werden in Abschnitt »Bedingungen, die sich auf Knoten-Attribute beziehen« auf Seite 99 genauer diskutiert.

Bedingungen

Damit der Cluster ordentlich funktioniert, muss der Administrator noch Bedingungen für die Platzierung der Ressourcen auf den Knoten und für die Beziehungen der Ressourcen untereinander angeben können. Diese Angaben sind im Abschnitt `constraints` innerhalb der Konfiguration der CIB zu finden. Es gibt drei verschiedene Arten von Bedingungen:

- Eine *Anordnung* (`order`) gibt die Beziehung zweier Ressourcen untereinander an. So ist es zum Beispiel sinnvoll, den Webserver erst nach der Ressource `IP-Adresse` für den Cluster zu starten.
- Eine Bedingung vom Typ *Co-Lokation* (`collocation`) stellt sicher, dass zwei Ressourcen auf demselben Knoten laufen. Hier ergibt es beispielsweise Sinn, dass der Webserver auf demselben Knoten läuft wie die IP-Adresse des Clusters.
- Eine *Platzierung* (`location`) von Ressourcen gibt an, auf welchen Knoten welche Ressourcen bevorzugt laufen.

Diese Bedingungen und ihre Verwendung werden im Folgenden genauer beschrieben.

Der CRM nutzt für die endgültige Auswahl, auf welchem Knoten eine Ressource laufen soll, ein Punktesystem: Der Administrator kann für zutreffende Regeln Punkte vergeben. Diese Regeln werden dann für die Ressourcen im *Kontext der einzelnen Knoten* ausgewertet. Falls die Regel zutrifft, erhält diese Ressource auf diesem Knoten die angegebenen Punkte. Nach Auswertung aller Regeln für alle Knoten wird die Ressource auf dem Knoten gestartet, auf dem sie die meisten Punkte erhalten hat. Falls mehrere Knoten die gleiche Punktzahl erreichen, verteilt der CRM die Ressourcen »optimal«, wie es in der Dokumentation heißt. Er versucht, die Ressourcen bestmöglich auf dem gesamten Cluster zu verteilen. Hierbei wird die Anzahl der Ressourcen pro Knoten bewertet. Die Last der Knoten wird bei der Verteilung nicht berücksichtigt. Jeder Punktwert von weniger als »0« führt dazu, dass diese Ressource auf diesem Knoten nicht laufen kann.

Neben den normalen Ganzzahlen für die Punkte gibt es folgende spezielle Werte:

`INFINITY`
: Größer als jede ganze Zahl. Die Ressource wird also in jedem Fall dort gestartet, wo eine Regel mit diesem Wert zutrifft.

`-INFINITY`
: Kleiner als jede ganze Zahl. Die Ressource wird also in keinem Fall dort gestartet, wo eine Regel mit diesem Wert zutrifft.

Für die Verknüpfung dieser speziellen Werte gelten folgende Regeln:

`INFINITY` +/- Zahl	ergibt	`INFINITY`

`-INFINITY` +/- Zahl	ergibt	`-INFINITY`
`INFINITY` +/- (`-INFINITY`)	ergibt	`-INFINITY`

Mit der Angabe der Punktzahl für einzelne Bedingungen kann der Administrator die Platzierung von Ressourcen innerhalb des Clusters gut beeinflussen. Vor der Eingabe komplexer Beziehungen sollte er sich aber mögliche Nebenwirkungen gut überlegen.

Anordnung (rsc_order)

Wie für den Systemstart, den der `init`-Prozess kontrolliert, ist es im Cluster ebenfalls wichtig, die Ressourcen in einer bestimmten Reihenfolge zu starten und anzuhalten. Dazu bietet der Cluster die Möglichkeit, solche Anordnungen explizit anzugeben.

Normalerweise werden die Ressourcen im Cluster willkürlich abgearbeitet. Wenn das nicht gewünscht wird, kann der Administrator die Reihenfolge auch über das Element `rsc_order` des Abschnitts `constraints` der CIB bestimmen. Die Attribute, neben der obligatorischen ID, sind:

`first`
: ID der Ressource, mit der zuerst eine Aktion (siehe `first-action`) durchgeführt wird.

`then`
: ID der Ressource, die abhängig von der ersten Ressource behandelt wird.

`score`
: Punktwert der Regel. Sinnvoll ist hier der Wert `INFINITY`, daher auch die Vorgabe. Manchmal kann aber auch ein Wert von »0« sinnvoll sein, wie wir später (siehe Kapitel 9, *Beispielszenarien*) noch sehen werden.

`symmetrical`
: Soll die Anordnung ebenfalls beim Anhalten der Ressourcen gelten, soll also zuerst die `then`-Ressource gestoppt werden und erst die `frist`-Ressource? Die Vorgabe ist `true`.

`first-action`
: An dieser Stelle kann der Administrator bestimmen, bei welcher Aktion der `first`-Ressource die Regel gelten soll. Die Vorgabe ist `start`. Möglich sind aber auch `stop`, `promote` und `demote`. Dann wird die Aktion der `then`-Ressource nur bei entsprechender Aktion ausgeführt. `promote` und `demote` sind Aktionen im Zusammenhang mit komplexen Ressourcen und werden weiter unten erklärt (siehe Abschnitt »Bedingungen im Zusammenhang mit Multi-State-Ressourcen« auf Seite 98).

then-action
: Falls die Ressource first die first-action ausgeführt hat, wird die Ressource then mit dieser Aktion angerufen. Die Vorgabe ist die Aktion, die die first-Ressource durchgeführt hat.

Eine Beispiel für eine Anordnung ist in der XML-Notation dargestellt:

```
<rsc_order id="ord_First_Then" first="resFirst" then="resThen"/>
```

oder auch in der CRM-Notation:

```
order ord_First_Then inf: resFirst resThen
```

Solch eine Beziehung wird zum Beispiel eingesetzt, wenn der Webserver erst dann gestartet werden soll, wenn die IP-Adresse für den Cluster auch schon gestartet wurde. In diesem Fall ist die first-Ressource die IP-Adresse und die then-Ressource der Webserver. Entsprechend der Vorgabe (symmetrical="true") wird der Webserver auch zuerst angehalten, und erst danach wird die IP-Adresse vom Interface gelöscht. Konkret lautet das in der XML-Notation:

```
<rsc_order id="order_IP_Apache" first="resIP" then="resApache"/>
```

und wiederum in CRM-Notation:

```
ord ord_IP_Apache inf: resIP resApache
```

Die Notation, die im Folgenden für Anordnungen verwendet wird, ist wie oben schon angedeutet ord_*First_Then*. Die Notation für andere Bedingungen wird analog aufgebaut.

Gibt es eine spezielle Richtlinie dazu, in welcher Reihenfolge der Cluster Ressourcen starten soll? Im Allgemeinen ist das relativ egal, und eine Anordnung ist nur in wenigen Fällen interessant:

In der Konfiguration der Applikation taucht die IP-Adresse auf. Das kann zum Beispiel der Fall sein, wenn der Administrator den Webserver nur an der virtuellen Adresse des Clusters lauschen lassen will. In diesem Fall muss zuerst die Ressource starten, die die Adresse konfiguriert, und erst anschließend die Applikation. Andernfalls wird die Applikation mit einer entsprechenden Fehlermeldung ihren Dienst quittieren. In den meisten Fällen ist es aber egal, weil die Applikation auf alle vorhandenen Adresse reagiert.

Die Ausnahme bildet ein NFS-Server als Ressource im Cluster. In diesem Fall ist es wichtig, dass die Applikation vor der IP-Adresse startet. Andernfalls würde ein Client, der in dem kurzen Zeitabschnitt den Dienst nutzen will, in dem die Adresse schon existiert, aber die Applikation noch nicht bereit ist, eine Fehlermeldung, genauer gesagt eine ICMP port unreachable-Meldung, erhalten. Das würde ihn ziemlich beleidigen, und er würde es nicht noch einmal versuchen. Die Verbindung risse dann ab. Falls die Applikation schon läuft, aber die IP-Adresse noch nicht im Netz bekannt ist, versucht es der Client einfach noch einmal, und die Verbindung kommt zustande.

Co-Lokation (rsc_colocation)

Mit dieser Bedingung kann der Administrator bestimmen, dass eine Ressource auf demselben Knoten laufen soll wie eine andere Ressource. Wie oben dargestellt, ist es meistens sinnvoll, den Webserver auf demselben Knoten zu starten wie die virtuelle IP-Adresse des Clusters, unter der der Webserver erreichbar sein soll. Die Attribute, neben der obligatorischen ID, für die `rsc_colocation`-Bedingung sind:

`rsc`
: ID der Ressource, die auf demselben Knoten laufen soll wie die Ressource `with-rsc`. Die Angabe ist notwendig.

`rsc-role`
: Die Rolle der Ressource, wenn diese Regel greifen soll. Möglich sind `Started` (Vorgabewert), `Stopped`, `Master` oder `Slave`.

`with-rsc`
: Die ID der Ressource, auf die sich diese Bedingung bezieht. Die Angabe ist notwendig.

`with-rsc-role`
: Die Rolle der Ressource, wenn diese Regel greifen soll. Möglich sind `Started` (Vorgabewert), `Stopped`, `Master` oder `Slave`.

`node-attribute`
: Das Attribut eines Knotens, wenn diese Bedingung gelten soll.

`score`
: Die Punktanzahl für diese Bedingung. Sinnvoll ist hier `INFINITY`. Die Angabe ist notwendig.

  Meistens reicht es aus, für diese Bedingung die notwendigen Attribute anzugeben.

Als Beispiel für die Co-Lokation soll die `rsc`-Ressource (`rsc="resRsc"`) unbedingt (`score="INFINITY"`) auf demselben Knoten laufen wie die `with`-Ressource (`with-rsc="resWith"`):

```
<rsc_colocation id="col_Rsc_With" rsc="resRsc" with-rsc="resWith"
                                                score="INFINITY"/>
```

In CRM-Notation:

```
colocation col_Rsc_With inf: resRsc resWith
```

Natürlich ist es auch im obigen Beispiel sinnvoll, den Webserver auf demselben Knoten zu starten wie die IP-Adresse, deshalb lautet die Bedingung dazu:

```
<rsc_colocation id="col_Apache_IP" rsc="resApache" with-rsc="resIP"
                                                    score="INFINITY">
```

bzw. in CRM-Notation:

```
colocation col_Apache_IP inf: resApache resIP
```

Mit dem score-Attribut kann man steuern, wie wichtig die Co-Lokation ist. Dieser Wert geht in die Berechnung für die Platzierung von Ressourcen ein. Wenn der Wert nicht INFINITY ist, kann es auch passieren, dass die beiden Ressourcen nicht auf demselben Knoten gestartet werden. Ein Wert von -INFINITY verhindert, dass die rsc-Ressource auf einem Knoten gestartet wird, wenn dort schon die Ressource resWith läuft.

Platzierung (rsc_location)

Im Gegensatz zu den Bedingungen oben, die nur die Beziehungen zwischen Ressourcen betrafen, kann der Administrator mit der letzten Bedingung ganz individuell bestimmen, auf welchem Knoten eine Ressource laufen oder nicht laufen soll. Es sind folgende Angaben denkbar:

- Der Apache-Webserver soll auf dem Knoten mit dem Namen node1 laufen, oder
- DRBD soll nicht auf einem Knoten laufen, auf dem die Kernelversion 2.6.35 ist.

Dies geschieht über das sogenannte rsc_location-Element der CIB. Das Element selbst hat, neben der obligatorischen ID, die folgenden Attribute:

rsc
: Die ID der Ressource.

node
: Der Name des Knotens, auf den sich diese Bedingung bezieht.

score
: Der Punktwert für diese Bedingung.

Mit folgender Bedingung kann der Administrator bestimmen, dass die Datenbank möglichst auf dem Knoten node1 und der Webserver auf dem Knoten node2 laufen soll:

```
<rsc_location id="loc_Database" rsc="resDatabase" node="node1" score="100"/>
<rsc_location id="loc_Apache" rsc="resApache" node="node2" score="100"/>
```

oder in der Kurzform, nur für den Webserver:

```
location loc_Apache resApache 100: node2
```

Falls eine Ressource unbedingt auf einem Knoten laufen soll, muss der Administrator INFINITY Punkte vergeben oder -INFINITY für alle anderen Knoten. Natürlich kann der CRM diese Bedingung nur dann erfüllen, wenn der Knoten auch verfügbar ist. Die genaue Syntax der Subshell wird später erklärt.

Mit der obigen Syntax können Ressourcen auf bestimmten Knoten platziert werden. Erweitert man die Platzierungen allerdings um *Regeln*, kann man noch wesentlich interessantere Konfigurationen anlegen.

Regeln

Neben den Bedingungen bieten Regeln eine weitere Flexibilität. Regeln können Platzierungen von Ressourcen von weiteren Faktoren abhängig machen. Regeln bestehen aus einem Ausdruck und einem `score`-Attribut. Die Punkte werden bei der Punktzählung von Ressourcen berücksichtigt, wenn der Ausdruck »wahr« ist. Ein Ausdruck (`expression`) *vergleicht* ein *Attribut* mit einem *Wert*.

Attribute können beliebige Attribute sein, die den Knoten bei ihrer Definition zugeordnet wurden: Attribute eines Knotens, die bei der Konfiguration der Knoten eingegeben wurden, dynamische Attribute, die im Abschnitt `status` der CIB gespeichert werden, oder das Meta-Attribut `#uname`, das den Namen des Knotens bezeichnet. Die Attribute können vom Typ Integer, String oder Version sein.

Vergleiche werden über Operatoren ausgewertet. Diese Operatoren sind:

`lt`, `gt`, `lte`, `gte`, `eq`, `ne`
: für die üblichen arithmetischen Operationen sowie

`defined` *und* `not_defined`
: um Attribute, die im Kontext eines Knotens definiert (oder eben nicht definiert) sind, zu bewerten.

Der *Wert*, mit dem das Attribut verglichen wird, muss natürlich vom selben Typ wie das Attribut sein. »Zahlen« werden numerisch verglichen, »Versionen« ermöglichen Vergleiche zwischen den Versionsständen, zum Beispiel »1.2« und »1.10«, wobei letzterer Wert größer ist, und »Zeichenketten« werden mit `strcmp()` verglichen.

Folgender Ausdruck ist also korrekt und zulässig:

```
attribute="#uname" operation="eq" value="node2"
```

Ausdrücke liefern wahr oder falsch, und dementsprechend werden die vorgegebenen Punkte zum Punktekonto der Ressource für diesen Knoten dazugezählt. Alle Ausdrücke werden immer im Kontext aller Knoten ausgewertet. Deshalb kann ein Ausdruck auch für mehrere Knoten gelten und auch für mehrere Knoten Punkte abwerfen.

Die Platzierung des Webservers auf Knoten `node2` von oben könnte also auch ausgeschrieben so lauten:

```
<rsc_location id="loc_Apache" rsc="resApache">
  <rule id="ruleApache" score="100">
    <expression id="expressionApache" attribute="#uname" operation="eq"
      value="node2">
  </rule>
</rsc_location>
```

oder in Kurzform

```
location loc_Apache resApache rule 100: #uname eq node2
```

Im folgenden Beispiel werden alle Elemente (einfache Ressourcen und Bedingungen) noch einmal zusammengefasst, um eine IP-Adresse und einen Webserver zu konfigurieren, die auf demselben Knoten richtig angeordnet laufen sollen:

```
<resources>
  <primitive id="resIP" class="ocf" provider="heartbeat" type="IPaddr2">
    <meta_attributes id="resIP_meta_attrs">
      <nvpair id="id_stickiness" name="resource-stickiness" value="100"/>
    </meta_attributes>
    <instance_attributes id="resIP_instance_attrs">
      <nvpair name="resIP-ip" id="ip_id" value="192.168.188.105"/>
    </instance_attributes>
  </primitive>
  <primitive id="resApache" class="ocf" provider="heartbeat" type="apache">
    <meta_attributes id="resApache_meta_attrs">
      <nvpair id="resApache-target_role" name="target-role" value="stopped"/>
    </meta_attributes>
    <instance_attributes id="resApache_instance_attrs">
      <nvpair id="resApache-conf" name="configfile"
        value="/etc/apache2/httpd.conf"/>
      <nvpair id="resApache-httpd" name="httpd" value="/usr/sbin/httpd"/>
    </instance_attributes>
  </primitive>
</resources>
<constraints>
  <rsc_order id="order_IP_Apache" first="resIP" then="resApache"/>
  <rsc_colocation id="col_Apache_IP" res="resApache" with-res="resIP"
    score="INFINITY"/>
</constraints>
```

Auch dieses Beispiel sei der Vollständigkeit halber noch einmal in CRM-Notation aufgeführt:

```
primitive resIP ocf:heartbeat:IPaddr2 \
  params ip="192.168.188.105" \
  meta resource-stickiness="100"
primitive resApache ocf:heartbeat:apache \
  params configfile="/etc/apache2/httpd.conf" httpd="/usr/sbin/httpd" \
  meta target-role="stopped"
order ord_IP_Apache inf: resIP resApache
colocation col_Apache_IP inf: resApache resIP
```

An diesem Beispiel kann man gut die `meta_attributes`, die das Verhalten der Ressource im Cluster bestimmen, und die `instance_attributes`, die der Konfiguration der OCF-Ressource dienen, unterscheiden. Das Meta-Attribut `target_role` des Webservers gibt zum Beispiel an, dass diese Ressource gerade angehalten ist. Die Ressource `resIP` versucht, sich mit 100 Punkten (Stickiness!) am aktuellen Knoten festzuklammern.

Den Status des Clusters kann sich der Administrator mit dem Befehl `crm_mon` anzeigen lassen. Er funktioniert ähnlich wie `top`, zeigt also durchgehend der Status an.

Falls der Administrator nur einen einmaligen Status haben will, gibt es die Option -1 für den Befehl.

```
# crm_mon -1
============
Last updated: Sun Jan 29 20:17:59 2012
Last change: Sun Jan 29 20:05:23 2012 via crmd on node1
Stack: openais
Current DC: node1 - partition WITH quorum
Version: 1.1.6-4.fc16-89678d4947c5bd466e2f31acd58ea4e1edb854d5
2 Nodes configured, 2 expected votes
2 Resources configured.
============

Online: [ node1 node2 ]

 resIP        (ocf::heartbeat:IPaddr2):    Started node1
 resApache    (ocf::heartbeat:apache):      Started node1
```

Alle Ressourcen laufen friedlich auf dem Knoten node1.

Das Punktesystem

Intern rechnet der Cluster bei jeder Änderung die Punkte für alle Ressourcen auf allen Knoten unter Berücksichtigung aller Bedingungen aus. Für unseren Beispielcluster ergibt sich folgende Tabelle:

	node1	node2
resIP	100	0
resApache	0	-INFINITY

Die Ressource resIP erhält auf Knoten node1 100 Punkte, weil sie auf diesem Knoten läuft. Die resource-stickiness von 100 Punkten gibt es also nur für diesen Knoten. Der Webserver sollte aufgrund der Co-Lokationsbedingung INFINITY Punkte auf Knoten node1 erhalten. Der Cluster verhält sich allerdings anders: Er vergibt -INFINITY Punkte für alle anderen Knoten, in unserem Fall node2.

Die Punkte, die der Cluster aktuell vergibt, kann man sich mit dem Kommando

```
# ptest -s -L
```

anzeigen lassen. Mehr dazu in Kapitel 5 im Abschnitt »ptest« auf Seite 147.

Ressourcen sollen dort laufen, wo sie am meisten Punkte einsammeln können. Sollte eine Ressource nicht auf diesem Knoten laufen, wird sie auf dem aktuellen Knoten angehalten und auf dem besseren gestartet.

Ressourcen für Fortgeschrittene

Neben den einfachen »primitiven« Ressourcen gibt es komplexere Typen von Ressourcen, die sich aus den einfachen zusammensetzen. Die Entwickler haben die Möglichkeit vorgesehen, einfache Ressourcen zu Gruppen zusammenzufassen, Ressourcen anzulegen, die mehr als einen internen Zustand haben (Multi-State), und sogenannte Klone zu erzeugen, bei denen einfache Ressourcen mehrmals im Cluster laufen dürfen. Welche Vorteile und Vereinfachungen sich dadurch für den Administrator ergeben, wird im Folgenden gezeigt.

Gruppen

Wenn mehrere einfache Ressourcen logisch zusammengehören und auf demselben Knoten in einer bestimmten Reihenfolge ausgeführt werden sollen, kann der Administrator natürlich alle Ressourcen einzeln anlegen und die entsprechenden Bedingungen manuell konfigurieren. Sollen zum Beispiel eine IP-Adresse, eine MySQL-Datenbank und ein Apache-Webserver zusammen auf einem Knoten ausgeführt in der genannten Reihenfolge gestartet werden, sind drei einfache Ressourcen (IP, MySQL und Apache) notwendig und zusätzlich je zwei Bedingungen für Co-Lokation und Anordnungen.

Werden einfache Ressourcen zu einer Gruppe zusammengefasst, entfallen die zusätzlichen vier Bedingungen, da Co-Lokation und Anordnung implizite Eigenschaften der Gruppe sind. Zusätzlich lassen sich diese Vorgaben einer Gruppe durch Attribute beeinflussen. Die Attribute, die schon von der einfachen Ressource her bekannt sind, lauten:

`id`
: Die eindeutige ID der Gruppe. Notwendig.

`description`
: Ein aussagekräftiger Name, der zur Beschreibung der Gruppe dient.

`priority`
: Die Priorität der Gruppe. Sie hilft dem Cluster, zu entscheiden, was gestartet werden soll, wenn nicht genügend Knoten zur Verfügung stehen, um alle Ressourcen zu starten.

`is-managed`
: Die Ressourcen der Gruppe werden durch den Cluster kontrolliert. Entspricht dem Attribut für den gesamten Cluster.

`target-role`
: Der Zustand der Gruppe. Dieses Meta-Attribut der Gruppe wird an die einzelnen Mitglieder der Gruppe vererbt und bestimmt den Zustand der einzelnen primitiven Ressourcen, falls dort nichts angegeben ist.

Einfache Ressourcen innerhalb von Gruppen werden in der angegebenen Reihenfolge gestartet und in der umgekehrten Reihenfolge wieder angehalten. Das Verhalten von `pacemaker` entspricht an dieser Stelle genau dem `init`-Prozess.

Nur primitive Ressourcen, keine der weiter unten beschriebenen Klone, können Mitglieder einer Gruppe sein.

Das Beispiel von oben (IP-Adresse, Datenbank und Webserver) lautet als Gruppe in XML-Notation:

```
<group id="groupWebserver">
  <primitive id="resIP" class="ocf" provider="heartbeat" type="IPaddr2">
    <instance_attributes id="resIP_instance_attrs">
      <nvpair id="resIP-ip" name="ip" value="192.168.188.95"/>
    </instance_attributes>
  </primitive>
  <primitive id="resMySQL" class="ocf" provider="heartbeat" type="mysql">
    <instance_attributes id="resource_MySQL_instance_attrs">
      (...)
    </instance_attributes>
  </primitive>
  <primitive id="resApache" class="ocf" provider="heartbeat" type="apache">
    <instance_attributes id="resApache_instance_attrs">
      <nvpair id="resApache-config" name="configfile"
        value="/etc/apache/httpd.conf"/>
      <nvpair id="resApache-httpd" name="httpd" value="/usr/sbin/httpd"/>
    </instance_attributes>
  </primitive>
</group>
```

In der CRM-Notation werden die Ressourcen zuerst separat angelegt und anschließend zu einer Gruppe zusammengefasst:

```
primitive resIP ocf:heartbeat:IPaddr2 \
  params ip="192.168.188.95"
primitive resMySQL ocf:heartbeat:mysql \
  params ...
primitive resApache ocf:heartbeat:apache \
  params configfile="/etc/apache2/httpd.conf" httpd="/usr/sbin/httpd"
group groupWebserver resIP resMySQL resApache
```

Gruppen können in Bedingungen genauso wie einfache Ressourcen verwendet werden. Wenn die oben definierte Ressource zum Beispiel bevorzugt auf Knoten `node2` laufen soll, kann man folgende Platzierung angeben:

```
<rsc_location id="locWebserver" rsc="groupWebserver" node="node2" score="100"/>
```

oder in Kurzform:

```
location locWebserver groupWebserver 100: node2
```

Ebenso kann die ID der Gruppen in Bedingungen für Co-Lokationen oder Anordnungen verwendet werden.

Klone

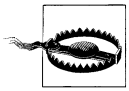
Neben den Gruppen gibt es noch eine weitere Vereinfachung: Wenn identische Ressourcen auf mehreren oder allen Knoten ausgeführt werden sollen, kann der Administrator sogenannte Klone verwenden.

Der Architekt eines Clusters kann im Grunde alle Ressourcen klonen, vorausgesetzt, der *Resource Agent* unterstützt das. Die meisten der OCF-Agenten haben nichts dagegen, geklont zu werden. Im Gegenteil, manchmal können die Agenten auch unterschiedlich konfiguriert werden, je nachdem, welcher der Klone gerade läuft. `pacemaker` lässt deshalb drei Typen von Klonen zu:

Anonyme Klone (anonymous clones)
: Dieser Typ ist der einfachste Klon. Alle Ressourcen verhalten sich identisch, egal wo sie laufen. Bei diesem Typ muss der Administrator aufpassen, ob es zu Konflikten kommt, wenn zwei Ressourcen auf einem Knoten laufen, und dies notfalls verhindern.

Global eindeutige Klone (globally unique clones)
: Alle Instanzen der Ressource sind unterschiedlich. Sie unterscheiden sich sowohl dann, wenn sie auf unterschiedlichen Knoten laufen, als auch, wenn zwei Ressourcen desselben Klons zusammen auf einem Knoten laufen. Möglich ist diese Art von Klonen, wenn zum Beispiel ihr eindeutiges Merkmal, das der CRM verteilt, in die Konfiguration der Ressource eingeht. *Resource Agents* können dann die Umgebungsvariable `OCF_RESEY_CRM_meta_clone` abfragen und nutzen. Sie ist für jeden Klon unterschiedlich. Ein Beispiel ist die Ressource `IPaddr2`.

Klone mit Status (stateful clones)
: Zusätzlich gibt es noch Ressourcen, bei denen sich aktive Ressourcen in unterschiedlichen Zuständen befinden können. Die Zustände werden als primär/sekundär (Master/Slave) bezeichnet. Diese Ressourcen können wiederum global eindeutig (oder eben nicht) sein. Diese Ressourcen werden im nächsten Abschnitt genauer beschrieben.



Auch LSB-Ressourcen können geklont werden. Allerdings dürfen sie nur in anonymen Klonen verwendet werden.

Natürlich werden für den Klon neben den Attributen der einfachen Ressourcen (`id`, `target-role`, `is-managed`, `priority`) weitere benötigt, um das Verhalten zu definieren. Diese Optionen sind:

`ordered`
: Die einzelnen Instanzen der Klone werden geordnet, also nacheinander gestartet und wieder angehalten. Die Vorgabe (`false`) ist, dass alle Klone gleichzeitig gestartet werden können.

`interleave`
: Diese Option bezieht sich auf die Anordnung der einzelnen Ressourcen bei der Konfiguration von mehreren Klonen. Weiter unten folgt eine genauere Erläuterung. Die Vorgabe ist `false`.

`notify`
: Gibt an, ob die Ressource die Klone auf den anderen Knoten vor und nach jeder Aktion über die eigene Tätigkeit unterrichten soll. Dazu muss der Agent natürlich so implementiert sein, dass er etwas mit dieser Benachrichtigung anfangen kann. Eine genauere Erklärung kommt weiter unten. Die Vorgabe ist `false`.

`clone-max`
: Diese wichtige Option gibt an, wie viele Instanzen der Ressource maximal im Cluster laufen können. Je nach Art der Ressource ist es sinnvoll, diese Anzahl gleich der Anzahl der Knoten im Cluster oder auf `2` zu setzen. Die Vorgabe ist hier die Anzahl der Knoten im Cluster.

`clone-node-max`
: Dieser Wert gibt an, wie viele Instanzen einer Ressource gleichzeitig auf einem Knoten laufen können. Meistens entspricht der Wert hier der Vorgabe `1`. Dieser Wert verhindert auch zum Beispiel Kollisionen bei ununterscheidbaren Ressourcen.

`globally-unique`
: Mit dieser Option kann man angeben, ob die Instanzen im Cluster global eindeutig oder ununterscheidbar sind. Die Vorgabe hier ist `true`. Bei LSB-Ressourcen muss dieses Attribut auf `false` gesetzt werden.

Zweckmäßig ist das Klonen bei Ressourcen, die mehrmals im Cluster ausgeführt werden sollen. Eine typische Anwendung sind STONITH-Ressourcen, die als identische Klone auf allen Knoten des Clusters gleichzeitig ausgeführt werden können. Die Ressource weiß ja, auf welchen Knoten sie aktuell läuft und welchem Knoten der Strom abgedreht werden soll. Die STONITH-Ressource auf dem aktuellen DC wird also richtig reagieren können. Ein Beispiel für einen solchen Klon-Agenten ist:

```
<clone id="cloneFencing">
  <meta_attributes id="cloneFencing-meta_attributes">
    <nvpair id="cloneFencing_clone-max" name="clone-max" value="2"/>
    <nvpair id="cloneFencing_clone-node-max" name="clone-node-max" value="1"/>
  </meta_attributes>
  <primitive id="fencing" class="stonith" type="ibmhc">
    <operations id="fencing-operations">
      <op id="fencing-op-monitor-15" interval="15" name="monitor" timeout="15"/>
    </operations>
    <instance_attributes id="fencing-instance_attributes">
      <nvpair id="resFencing_ip" name="ipaddr" value="192.168.224.99"/>
    </instance_attributes>
  </primitive>
</clone>
```

Auch hier wieder die Kurzform der Konfiguration:

```
primitive resHMC stonith:ibmhmc \
  params ipaddr="192.168.224.99" \
  op monitor interval="15s" timeout="15s"
clone cloneFencing resHMC \
  meta clone-max="2" clone-node-max="1"
```

Hier wird die einfache STONITH-Ressource vom Typ `ibmhmc` angelegt. Zur Konfiguration muss als Attribut der Ressource nur die IP-Adresse des entsprechenden Geräts eingegeben werden. Da diese Ressource auf allen Knoten des Clusters ausgeführt werden soll, muss `clone-max` entsprechend der Anzahl der Knoten im Cluster gesetzt werden, im Beispiel auf `2`. Somit wird auf beiden Knoten des Clusters die Ressource gestartet. Im Fehlerfall startet der überlebende DC den anderen Knoten noch einmal oder fährt ihn herunter.

Klone können nicht Mitglied einer Gruppe sein. Gruppen können aber sehr wohl geklont werden. Inwieweit das sinnvoll ist, muss der Administrator für seinen Aufbau entscheiden.

Die ID von Klonen kann ganz normal in Bedingungen benutzt werden.

Die Punkteverteilung bei Klonen unterscheidet sich in bisschen von einfachen Ressourcen. Damit ein Klon stabil auf einem Knoten bleibt, rechnet der CRM immer mit einer Vorgabe der `resource-stickiness` bei Klonen von `1`, nicht `0`.

Unterschiedliche Instanzen eines Klons kennzeichnet der Cluster durch eine angehängte Zahl, angefangen bei `0`. Die erste Instanz einer geklonten `Sysinfo`-Ressource heißt folglich `resSysinfo:0`. Klone mit unterschiedlichen Funktionen (`globally-unique="true"`) verwenden diese Information, um das eigene Verhalten festzulegen.

Multi-State-Ressourcen

Noch komplexer sind Multi-State-Ressourcen. Dies sind Ressourcen, die in verschiedenen Zuständen vorliegen können. Als Beispiel dienen hier immer die *Distributed Redundant Block Devices* (DRBD). Diese Ressource ermöglicht die Spiegelung der Informationen auf Ebene der Block-Devices. Die erste Beispielanwendung in Kapitel 9 im Abschnitt »Die Nutzung von DRBDs« auf Seite 308 verwendet diese Ressource, um einen hochverfügbaren NFS-Fileserver anzubieten. Die Nutzung dieser Ressource wird dort genauer besprochen.

Damit die Daten zwischen den Knoten repliziert werden können, muss die Ressource auf beiden Knoten laufen. In der aktuellen Version 8 der Software können zwar beide DRBDs im Zustand `Master` vorliegen, bevor man diese Fähigkeit aber ausnutzt, muss man sich überlegen, ob die Knoten tatsächlich gleichzeitig auf die Daten zugreifen müssen. In diesem Fall muss das Dateisystem sich des gleichzeiti-

gen Zugriffs gewahr sein und Dateien für den exklusiven Zugriff reservieren können. Diesen Trick beherrschen nur explizite Clusterdateisysteme wie OCFS2 oder GFS. Ein einfaches `ext4` oder `xfs` beherrscht das nicht. Zusätzlich muss ein Locking-Daemon verteilt auf allen Knoten laufen, der den Wunsch nach Blockierung auch auf allen Knoten umsetzt. Man erkennt schnell, dass solche Master/Master-Konfigurationen nicht so einfach aufzusetzen sind. Die Überlegung, ob das wirklich notwendig ist, lohnt sich also!

Auf normalen Clustern läuft das DRBD auf beiden Knoten, einmal als `Master` und einmal als `Slave`. Im Fehlerfall wird das `Slave`-Device einfach zum `Master` befördert und kann auf dem zweiten Knoten mit allen Applikationsdaten genutzt werden, die zuvor noch der erste Knoten geschrieben hatte.

Natürlich kann man den Zustand des Block-Device abfragen und abhängige Ressourcen (zum Beispiel ein Dateisystem) nur dort starten, wo auch der `Master` läuft.

Realisiert ist diese Art von Ressource im Cluster als ein Superset von Klon-Ressourcen. Deshalb stehen auch alle Attribute einer Klon-Ressource zur Verfügung. Zusätzlich kommen noch folgende zwei Attribute hinzu:

`master-max`
: Anzahl von Ressourcen im Cluster, die sich im Zustand `Master` befinden dürfen. Sinnvollerweise ist dieses Attribut meistens auf `1` gesetzt.

`master-node-max`
: Anzahl von Ressourcen auf einem Knoten, die sich im Zustand `Master` befinden können. Sinnvollerweise ist dieses Attribut meistens auf `1` gesetzt.

Eine Beispielkonfiguration einer DRBD-Ressource könnte wie folgt aussehen:

```
<master id="msDRBD">
  <meta_attributes id="ms_DRBD-meta_attributes">
    <nvpair id="msDRBD_clone-max" name="clone-max" value="2"/>
    <nvpair id="msDRBD_clone-node-max" name="clone-node-max" value="1"/>
    <nvpair id="msDRBD_master-max" name="master-max" value="1"/>
    <nvpair id="msDRBD_master-node-max" name="master-node-max" value="1"/>
    <nvpair id="msDRBD-notify" name "notify" value="true"/>
  </meta_attributes>
  <primitive id="resDRBD" class="ocf" provider="linbit" type="drbd">
    <instance_attributes id="resDRBD-instance_attributes">
      <nvpair id="resDRBD_drbd_resource" name="drbd_resource" value="r0"/>
    </instance_attributes>
    <operations id="resDRBD-operations">
      <op id="resDRBD-op-Master" name="monitor" role="Master" interval="30s" />
      <op id="resDRBD-op-Slave" name="monitor" role="Slave" interval="300s" />
      <op id="resDRBD-op-Start" name="start" timeout="240s" />
      <op id="resDRBD-op-Stop" name="stop" timeout="100s" />
    </operations>
  </primitive>
</master>
```

Bitte beachten Sie, dass der Provider der *Resource Agents* nicht mehr `heartbeat` ist, sondern dieses Programm von der Firma Linbit beigesteuert wurde. Die DRBD-Ressource funktioniert auch nicht ohne die Benachrichtigung der Ressourcen untereinander. Deshalb muss `notify="true"` gesetzt werden. Die Konfiguration kann man auch kurz so ausdrücken:

```
primitive resDRBD \
  params drbd_resource="r0" \
  op monitor role="Master" interval="30s" \
  op monitor role="Slave" interval="300s" \
  op start timeout="240s" \
  op stop timeout="100s"
ms msDRBD meta master-max="1" master-node-max="1" \
   clone-max="2" clone-node-max="1" notify="true"
```

In einem Cluster wird eine Ressource vom Typ `drbd` zweimal ausgeführt, maximal einmal pro Knoten. Nur eine Ressource davon kann sich im Zustand `Master` befinden. Als Parameter wird der DRBD-Ressource der Wert `r0` als Name für die zu verwendende Konfiguration übergeben. Alle 30 Sekunden wird die Ressource geprüft, wenn sie `Master` ist, ansonsten nur alle 5 Minuten.

Ein Beispiel zur Nutzung des Agenten für einen hochverfügbaren NFS-Server wird in Kapitel 9 im Abschnitt »Anwendung: Ein hochverfügbarer NFS-Server« auf Seite 323 gezeigt.

Klone mit verschiedenen Aufgaben bzw. Multi-State-Ressourcen auf verschiedenen Knoten müssen sich auf den verschiedenen Knoten abgleichen können. Dazu müssen solche Agenten zusätzlich noch die Operation `notify` beherrschen. Dies ist zum Beispiel für die DRBD-Ressource wichtig, damit die laufenden Instanzen während des Starts untereinander vereinbaren können, wer `Master` wird und wer `Slave` bleibt.

Wenn das Attribut `notify="true"` gesetzt ist, erhalten alle Agenten `pre-` und `post-`Meldungen (vorher und nachher) von den anderen Ressourcen, bevor diese eine Operationen durchführen. So können sie mögliche Fehler, die dem CRM eigentlich nicht passieren dürften, erkennen und Probleme abfangen. DRBD-Agenten können prüfen, ob schon ein `Master` im Cluster existiert, wenn eine Ressource zum Primary befördert werden soll, und notfalls noch einmal einschreiten.

Der Status der jeweils anderen Ressourcen wird wiederum in Umgebungsvariablen an den Agenten übergeben. Die genauen Namen der Variablen für Klone (Abschnitt »Klone« auf Seite 90) und Multi-State-Ressourcen (Abschnitt »Multi-State-Ressourcen« auf Seite 92) werden auf den entsprechenden Webseiten des Projekts[1] diskutiert.

Ressourcen migrieren

Ressourcen werden normalerweise bei einer Migration auf dem einen Knoten angehalten und auf dem anderen Knoten neu gestartet. Natürlich brechen alle bestehen-

1 *http://www.linux-ha.org/v2/Concepts/Clones* und *http://www.linux-ha.org/v2/Concepts/MultiState*

den Verbindungen ab und müssen vom Client zum Server auf dem neuen Knoten neu aufgebaut werden. Einige Ressourcen versuchen diese »harte« Migration zu vermeiden, wenn es irgendwie möglich ist. Dazu muss der *Resource Agent* mitspielen, das heißt dafür ausgelegt sein. Prädestiniert für eine solche »weiche« Migration sind komplette virtuelle Rechner, die im Cluster als Ressourcen existieren. Unter bestimmten Umständen kann der Cluster versuchen, diese virtuellen Rechner als Ganzes im laufenden Betrieb umzusiedeln. Der Benutzer eines solchen virtuellen Rechners bekommt vom Umzug (fast) nichts mit.

Die *Resource Agents* müssen natürlich mitspielen und neben dem neuen Meta-Attribut `allow-migrate` die Operationen `migrate-from` und `migrate-to` verstehen. Setzt der Administrator das Meta-Attribut auf `true`, gibt er dem Cluster zu verstehen, dass er eine weiche Migration wünscht, wenn es möglich ist. Dann ruft der Cluster den Agenten mit `migrate-to` auf dem Knoten auf, auf dem die Ressource bisher gelaufen ist, auf dem anderen Knoten entsprechend mit `migrate-from`. Mehr zu dieser spannenden Technik erfahren Sie in Kapitel 9 im Abschnitt »Live-Migration« auf Seite 339.

Bedingungen für Fortgeschrittene

Wie oben bereits angedeutet, kann der Administrator beliebige Bedingungen definieren, um Ressourcen auf bestimmten Knoten laufen zu lassen oder Verknüpfungen zwischen Ressourcen zu erreichen. Anhand von einigen Beispielen möchte ich diese Möglichkeiten darstellen:

- Anordnungen von Bedingungen.
- Bedingungen im Zusammenhang mit Multi-State-Ressourcen.
- Bedingungen, die sich auf Attribute der Knoten beziehen.
- Zeitliche Vorgaben für Bedingungen.
- Erreichbarkeit im Netz.
- Frei definierbare Kriterien.
- Umschalten erst nach »N« Fehlern. Zuerst soll der CRM versuchen, die Ressource auf dem aktuellen Knoten neu zu starten.
- Systemgesundheit.

Anordnungen von Bedingungen

Mit dem Cluster-Manager `pacemaker` kann der Administrator relativ komplexe Anordnungen von Bedingungen eingeben. Ressourcen in Gruppen wurden immer sequenziell und auf demselben Knoten ausgeführt. Die Anordnungen sind eine Verallgemeinerung dieser Regeln.

Als Erstes sollen die Ressourcen `resIP`, `resDatabase` und `resApache` angeordnet werden (siehe dazu auch Abbildung 4-1).

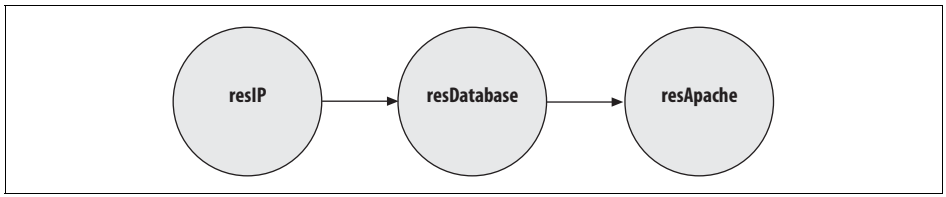


Abbildung 4-1: Anordnung von Bedingungen. Die Ressource resIP soll zuerst gestartet werden, danach resDatabase und zum Schluss erst die Ressource resApache.

Anstelle einer Gruppe oder der einfachen Bedingungen

```
order ordIPDatabase inf: resIP resDatabase
order ordDatabaseApache inf: resDatabase resApache
```

kann man das auch mit einem `resource_set` aus den Ressourcen innerhalb der `rsc_order` angeben:

```
<rsc_order id="oderAll">
  <resource_set id="orderBeispiel" sequential="true">
    <resource_ref id="resIP"/>
    <resource_ref id="resDatabase"/>
    <resource_ref id="resApache"/>
  </resource_set>
</rsc_order>
```

oder in einfacher Notation:

```
order orderAll inf: resIP resDatabase resApache
```

Das mag bei drei Ressourcen noch nicht so interessant sein, aber die Folgen der Einführung einer solchen Anordnung in `pacemaker` sind gewaltig. Wenn noch ein DNS-Server dazukommt, hören sich die Anordnungen schon wesentlich komplexer an: IP-Adresse und Datenbank sollen gleichzeitig starten, um Zeit zu sparen, aber der Webserver und der DNS-Server sind von beiden Ressourcen abhängig, nicht aber untereinander. Dem Cluster kann man das relativ einfach mitteilen:

```
<rsc_order id="orderAll">
  <resource_set id="orderIPDatabase" sequential="false">
    <resource_ref id="resIP"/>
    <resource_ref id="resDatabase"/>
  </resource_set>
  <resource_set id="ordApacheDNS" sequential="false">
    <resource_ref id="resApache"/>
    <resource_ref id="resDNS"/>
  </resource_set>
</rsc_order>
```

In der alten Notation würden hierfür zwei Gruppen benötigt. In der Subshell wird das mit Klammern ausgedrückt:

```
order ordAll inf: (resIP resDatabase) (resApache resDNS)
```

Grafisch aufbereitet, sieht die Bedingung nun wie in Abbildung 4-2 dargestellt aus.

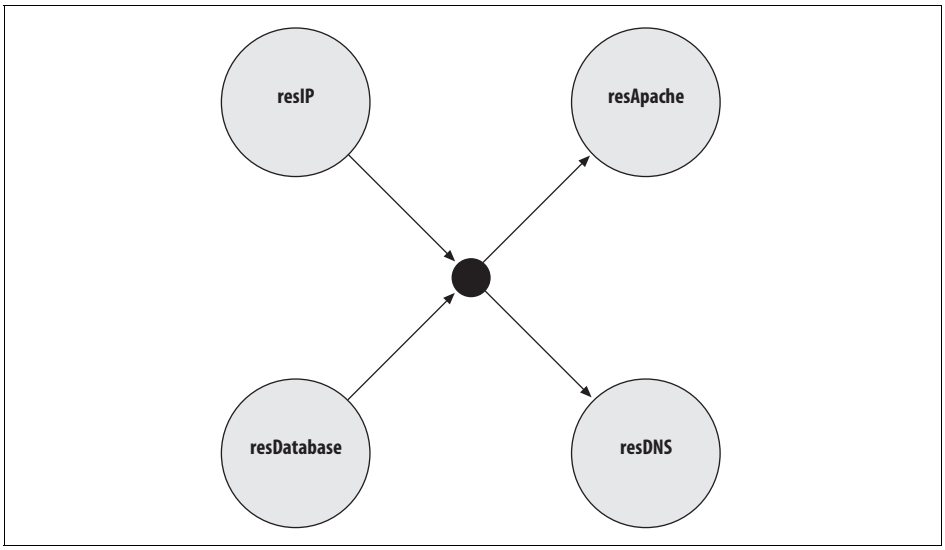


Abbildung 4-2: Nicht nur einfache Anordnungen lassen sich mit einem resource_set darstellen, sondern auch komplexere Zusammenhänge.

Natürlich ersetzt die obige Definition eine Gruppe nicht ganz, da die Co-Lokation noch fehlt. Diese Bedingung kann man ebenfalls in der neuen Notation darstellen. Der Wunsch des Administrators, alle Ressourcen auf einem Knoten laufen zu lassen, lautet nun:

```
colocation colAll inf: resDNS resApache resDatabase resIP
```

Mit dieser Bedingung laufen alle vier Ressourcen auf demselben Knoten (Co-Lokation!). Falls die Ressource `resDatabase` nicht laufen kann, wird auch der Webserver angehalten, da ja die Ressource, von der er abhängt, auf keinem Knoten läuft.

Allgemein gelten für Sätze von Co-Lokationsbedingungen folgende zwei Grundsätze:

1. Damit *ein* Mitglied eines Co-Lokationsregelsatzes läuft, müssen *alle* Mitglieder des in der Konfiguration *folgenden* Regelsatzes laufen.
2. Wenn einem Regelsatz das Attribut `sequential="true"` zugewiesen wurde, muss eine Ressource laufen, damit die im *selben* Regelsatz *vorhergehende* Ressource laufen kann. Bei `sequential="false"` fällt diese Bedingung weg.

Diese beiden Grundsätze sind vielleicht nicht ganz einfach zu verstehen, und manchmal verhalten sich die Ressourcen nicht so, wie sich der Administrator das zuerst gedacht hatte. Wenn man sich die Anordnung der Bedingungen anhand der

beiden Sätze oben noch einmal genau überlegt, erscheint das Verhalten der Ressourcen aber plötzlich ganz logisch. Bei der Beurteilung des Verhaltens einer Ressource muss man zwei Punkte unterscheiden: ihr Verhältnis zu der *darauffolgenden* Ressource im selben Satz von Ressourcen (`resource_set`) und die Beziehung zu *allen* Ressourcen im darauffolgenden Ressourcensatz.

Damit können dann auch wesentlich komplexere Abhängigkeiten abgebildet werden als mit einfachen Gruppen.

Bedingungen im Zusammenhang mit Multi-State-Ressourcen

Manchmal möchte der Administrator, dass der `Master` einer Multi-State-Ressource auf einem bestimmten Knoten läuft. Diese Bedingung kann realisiert werden, da eine Regel auch noch ein Attribut »Rolle« zulässt. Dieses Attribut lässt eine Regel nur dann wirksam werden, wenn die Ressourcen, für die die Platzierung eingerichtet wird, in der angegebenen Rolle vorliegen. Mögliche Werte dieses Attributs sind `Started`, `Stopped`, `Slave` oder `Master`. Sinnvoll wird die gewünschte Konfiguration natürlich mit dem Wert `Master`.

Der Wunsch des Administrators, den DRBD-`Master` auf Knoten `node1` laufen zu lassen, lässt sich also wie folgt ausdrücken:

```
location loc_DRBD_Node1 msDRBD rule role="Master" 100: #uname eq node1
```

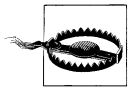
Durch diese Bedingung erhält eine Ressource 100 Punkte, wenn sie als `Master` auf dem Knoten `node1` läuft. Werden keine Punkte für andere Knoten vergeben, wird der `Master` dieser Ressource auf dem vorgegebenen Knoten laufen.

Diese Rollen kann man auch in den anderen Bedingungen verwenden: Es ist zum Beispiel sinnvoll, ein Dateisystem nur dort zu starten, wo auch das DRBD im Zustand `Master` vorliegt. Andernfalls würde der Cluster einer Fehler melden. Mit einer kleinen Erweiterung der Bedingung Co-Lokation mit einem `with-rsc-role`-Attribut lässt sich das bewerkstelligen, indem man wiederum die Rolle `Master` nutzt. Gleichzeitig darf das Dateisystem erst nach der Beförderung der DRBD-Instanz zum `Master` gestartet werden. Dies kann über die `first-action` einer Anordnung erreicht werden. Neben `start` und `stop` kennen Multi-State-Ressourcen auch noch die Aktion `promote`, die dazu dient, Ressourcen in den Zustand `Master` zu befördern. Natürlich existiert auch die Aktion `demote`, die die Ressource vom `Master` zum `Slave` herabstuft. Folgende Bedingungen starten ein Filesystem nur auf einem Knoten, auf dem die DRBD-Ressource als `Master` läuft, in der richtigen Reihenfolge:

```
colocation col_Filesystem_DRBD inf: resFilesystem msDRBD:Master
order ord_Filesystem_DRBD inf: msDRBD:promote resFilesystem:start
```

Solche Bedingungen werden häufig genutzt, um ein Dateisystem und Applikationen auf dem Knoten zu starten, auf dem auch das DRBD im Zustand `Master` vorliegt. Die Applikation kann sich dabei auf die Ressource `Filesystem` beziehen.

Das Beispiel »hochverfügbarer NFS-Server« in Kapitel 9 im Abschnitt »Anwendung: Ein hochverfügbarer NFS-Server« auf Seite 323 macht von diesen Bedingungen Gebrauch.



Die Vorgabe für die Aktion der then-Ressource ist die Aktion der from-Ressource und nicht start. Wenn die Aktion von DRBD, wie im Beispiel oben, promote ist, kann der *Resource Agent* des Dateisystems herzlich wenig damit anfangen. In diesem Fall muss start explizit angegeben werden.

Bedingungen, die sich auf Knoten-Attribute beziehen

Anstelle des Attributs score einer Regel kann man auch das Attribut score_attribute verwenden, um eine zusätzliche Flexibilität der Konfiguration zu erreichen. Dieses Attribut veranlasst den Cluster-Manager, bei der Auswertung der Regel und der Punktevergabe einen Attributnamen, der dem vorgegebenen score_attribute entspricht, in der CIB zu suchen und den Wert dieses Attributs als Punktzahl zu verwenden.

Als Beispiel soll die Ressource resMyRes auf dem Knoten laufen, auf dem am meisten Speicher installiert ist:

```
location loc_MyRes resMyRes rule mem_installed: defined mem_installed
```

Der Ausdruck sorgt dafür, dass das Attribut im Kontext eines Knotens auch definiert ist. Gleichzeitig wurde in der Definition der Knoten das Attribut mem_installed als zusätzliches Attribut zur Beschreibung der Knoten eingegeben:

```
# crm node attribute node1 set mem_installed 4096
# crm node attribute node2 set mem_installed 2048
```

Bei der Bewertung der Platzierung sucht der CRM jetzt bei jedem Knoten ein Attribut mit dem Namen mem_installed. Die Punktzahl, die der Knoten erhält, entspricht dem Wert dieses Attributs. Im obigen Fall würde der Knoten node1 also 4.096 Punkte erhalten, node2 nur die Hälfte. Die Ressource würde folglich auf dem ersten Knoten gestartet.

Die Attribute der Knoten im Beispiel oben wurden statisch bei der Definition der Knoten eingegeben. Natürlich lassen sich auch dynamisch generierte Informationen aus dem Status-Abschnitt der CIB nutzen. Dies soll am Beispiel der vorhin angesprochenen Ressource SysInfo dargestellt werden. Sie liefert ein aktuelles Abbild des Systemzustands beim Aufruf mit monitor. Diese Werte können als Kriterien für Bedingungen herangezogen werden.

Die Ressource resMyRes soll möglichst auf dem Knoten gestartet werden, auf dessen *root*-Partition mehr als 2 GByte frei sind. Die folgende Platzierung verteilt –100 Punkte, wenn weniger Platz frei ist:

```
location locRootFree resMyRes rule -100: root_free lt 2
```

Natürlich muss die Ressource SysInfo konfiguriert sein und für jeden Knoten Werte liefern, damit diese Bedingung sinnvoll eingesetzt werden kann. Eine solche Konfiguration erreicht man am einfachsten durch einen Klon.

Zeitliche Vorgaben für Bedingungen

Die Definition der CIB lässt noch ganz andere Möglichkeiten für Attribute zu. Jedes Attribut, egal ob es ein meta_attribut oder instance_attribut ist, lässt sich mit einer Punktzahl und einer Regel versehen. Regeln können entweder die oben genannten Vergleiche sein oder aber *zeitliche Regeln*. So kann sich der Cluster zu verschiedenen Zeiten unterschiedlich verhalten. Ein Beispiel wird weiter unten gezeigt.

Zeitliche Regeln sind anstelle eines einfachen Ausdrucks (expression) durch einen sogenannten Datumsausdruck (date_expression) gekennzeichnet. Dieser besteht je nach Kontext aus einer Datumsspezifikation (date_spec) und/oder einer Dauer (duration). Die möglichen Attribute einer date_expression sind:

start
: Angabe einer Startzeit. Das Format muss ISO 8601 entsprechen.

end
: Angabe einer Endzeit im ISO 8601-Format. Dieses Attribut kann bei Angabe von Startzeitpunkt und Dauer ausgelassen werden.

operation
: Die Operation vergleicht je nach Kontext die aktuelle Zeit bzw. das Datum mit den Angaben oben.

Die Operatoren bedeuten dabei im Einzelnen:

gt
: Bei der Operation gt wird die aktuelle Zeit (now) mit der Spezifikation des Startzeitpunkts verglichen. Der Ausdruck ist demnach wahr, wenn die Startzeit schon vorbei ist.

lt
: Entsprechend ist ein Datumsausdruck mit dem Operator lt wahr, wenn now vor dem Schlusszeitpunkt liegt.

in_range
: Die Operation in_range vergleicht now mit dem Start- und dem Endzeitpunkt und ist wahr, wenn now zwischen diesen beiden Spezifikationen liegt. Dieser Operator ist auch die Vorgabe, falls nichts anderes angegeben wurde.

date_spec
: Der letzte Operator ist date_spec, der now mit einer Datumsspezifikation auf eine ähnliche Weise vergleicht, wie es auch cron macht. Mit diesem Operator

können also wiederkehrende Ereignisse eingegeben werden. Zusätzlich achtet dieser Operator auch noch auf den Start- und den Endzeitpunkt.

Start- und Endzeitpunkt werden in einer Notation eingegeben, die der ISO-Norm 8601 entspricht. Attribute der Datumsspezifikationen können wie folgt angegeben werden:

`hours`
: Werte von 0 bis 23.

`monthdays`
: Erlaubte Werte von 0 bis 31, je nach Monat.

`weekdays`
: Tage der Woche, Wertebereich 1 bis 7, wobei 1 Montag entspricht.

`yeardays`
: Tag des Jahres, Wertebereich von 1 bis 366.

`months`
: Monat von 1 bis 12.

`weeks`
: Woche von 1 bis 53.

`years`
: Das Kalenderjahr.

`moon`
: Mondphase, wobei 0 Neumond entspricht und 4 Vollmond bedeutet.

Zeitdauern werden wie Zeitspezifikationen eingegeben.

Beispiel 1: Gültig für einen Zeitpunkt innerhalb des Jahres 2011:

```
<rule id="in_2011">
  <date_expression id="date_expr1" start="2011" operation="in_range">
    <duration years="1"/>
  </date_expression>
</rule>
```

Äquivalent ist dieser Ausdruck:

```
<rule id="in_2011">
  <date_expression id="date_expr2" operation="date_spec">
    <date_spec years="2011"/>
  </date_expression>
</rule>
```

Beispiel 2: Die normale Arbeitszeit wird wie folgt angegeben:

```
<rule id="mon-fri_9_to_5">
  <date_expression id="date_expr3" operation="date_spec">
  <date_spec hours="9-17" days="1-5"/>
  </date_expression>
</rule>
```

Wenn man den halben Samstag auch noch arbeiten will, kann man das so ausdrücken:

```
<rule id="working_hard" boolean_op="or">
  <date_expression id="date_expr4-1" operation="date_spec">
    <date_spec hours="9-17" days="1-5"/>
  </date_expression>
  <date_expression id="date_expr4-2" operation="date_spec">
    <date_spec hours="9-13" days="6"/>
  </date_expression>
</rule>
```

Für abergläubische Administratoren sei noch folgendes Beispiel aufgeführt:

```
<rule id="superstition">
  <date_expression id="date_expr5" operation="date_spec">
    <date_spec weekdays="5" monthdays="13" moon="4"/>
  </date_expression>
</rule>
```

Nützlich sind solche zeitlichen Begrenzungen, wenn man den Platzwechsel von Ressourcen zwischen den Knoten steuern will, da ja bei jedem Wechsel eine gewisse Auszeit herrscht, auch wenn diese nur sehr kurz ist. Abhängig von der Applikation, reißen auch alle bestehenden Verbindungen ab. Wenn es zum Beispiel in der Nacht von Mittwoch auf Donnerstag Probleme mit einem Knoten gab und die Ressourcen auf andere Knoten gewechselt sind, will man nicht gleich wieder eine (kurze) Unterbrechung riskieren, nachdem der Schaden am Donnerstag behoben ist. Besser wäre es, die Ressourcen erst in der Nacht von Sonntag auf Montag zwischen 2 und 3 Uhr sich ihren angestammten Platz suchen zu lassen.

Diese Operation kann man mit einem zeitabhängigen Wert der `resource-stickiness` einer Ressource durchführen, wenn er unter der Woche relativ hoch ist und nur zu einem bestimmten Zeitpunkt so niedrig wird, dass der Cluster die Ressourcen neu verteilen kann. Die entsprechende Konfiguration der CIB würde dann so aussehen:

```
<primitive id="resIP" class="ocf" provider="heartbeat" type="IPaddr2">
  <meta_attributes id="resIP-meta_attributes-override" score="100">
    <rule id="rule_failover">
      <date_expression id="expr_failover" operation="date_spec">
        <date_spec id="date_spec-night" hours="2-3" weekdays="1"/>
      </date_expression>
    </rule>
    <nvpair id="stickiness-night" name="resource-stickiness" value="0"/>
  </meta_attributes>
  <meta_attributes id="meta_attrs_default" score="10">
    <nvpair id="sickiness-day" name="resource-stickiness" value="100"/>
  </meta_attributes>
  <instance_attributes>
    <nvpair id="id_resIP_ip" name="ip" value="192.168.188.95"/>
  </instance_attributes>
</primitive>
```

Montags zwischen 2 und 3 Uhr morgens erhält der Abschnitt meta_attributes der virtuellen IP- Adresse 100 Punkte. Natürlich will der gierige Cluster diese Punkte und setzt damit die resource-stickiness auf 0. Zu allen anderen Zeiten erhält der Cluster nur 10 Punkte, und die resource-stickiness wird auf 100 gesetzt. Alternativ kann man sich vorstellen, eine gründliche Überprüfung einer Ressource (z. B. fsck), die eine hohe Systemlast verursacht, in die Nachtstunden zu verlegen.

Die obigen Regeln für die resource-stickiness gelten natürlich nur für eine einfache Ressource. Im Abschnitt rsc_defaults der Konfiguration für den gesamten Cluster (crm_config) kann der Administrator aber Werte vorgeben, die dann für alle Ressourcen gelten, die die resource-stickiness überschreiben. Zeitabhängige globale Werte würden demnach wie folgt eingegeben:

```
<crm_config>
  <cluster_property_set id="cib-bootstrap-options">
    <rsc_defaults>
      <meta_attributes id="moday2to3" score="100">
        <rule id="rule_monday2to3">
          <date_expression id="expr_monday2to3" operation="date_spec">
            <date_spec hours="2-3" weekdays="1"/>
          </date_expression>
        </rule>
        <nvpair id="failover-stickiness" name="resource-stickiness" value="0"/>
      </meta_attributes>
      <meta_attributes id="default_meta_attrs" score="10">
        <nvpair id="normal_stickiness" name="resource-stickiness" value="100"/>
      </meta_attributes>
    </rsc_defaults>
  </cluster_property_set>
</crm_config>
```

Alle Änderungen im Cluster werden üblicherweise durch Ereignisse, wie zum Beispiel den Ausfall eines Knotens, gesteuert. Damit der Cluster auch auf zeitgesteuerte Ereignisse reagieren kann, muss das Attribut cluster-recheck-interval in der Konfiguration des Clusters entsprechend kurz gesetzt werden. So berechnet der CRM zum Beispiel zusätzlich alle 15 Minuten, ob er die Ressourcen neu verteilen muss, obwohl kein Knoten den Status geändert hat.

Ein anderes Beispiel für solche Regeln zeigt, wie unterschiedliche Hardware im Cluster genutzt werden kann. Die Ressource resIP soll auf dem Knoten node1 auf eth0 erscheinen, auf dem Knoten node2 aber auf eth1. Dazu muss man das instance_attribute nic der Ressource abhängig vom uname des Knotens setzen:

```
<primitive id="resIP" class="ocf" provider="heartbeat" type="IPAddr2">
  <instance_attributes id="special-node2" score="2">
    <rule id="node2-special-case">
      <expression id="node2-special-case-expr"
        attribute="#uname" operation="eq" value="node2"/>
```

```
    </rule>
    <nvpair id="node2-interface" name="nic" value="eth1"/>
  </instance_attributes>
  <instance_attributes id="defaults" score="1">
    <nvpair id="default-interface" name="nic" value="eth0"/>
  </instance_attributes>
</primitive>
```

Leider haben alle diese Beispiele keine Entsprechung in der `crm`-Subshell, sodass hier nur noch die XML-Notation dargestellt werden kann.

Erreichbarkeit im Netz

Ein wichtiges Kriterium für die Entscheidung, ob ein Knoten Ressourcen übernehmen soll oder nicht, ist die Erreichbarkeit des Knotens im Netz. Diese Entscheidung kann nicht allein über den Status der Netzwerkschnittstelle (`up/down`) getroffen werden. Vielmehr sind die Knoten mit Kabeln an den nächsten Switch angebunden. Switches oder Router auf dem Weg zum Client können ausfallen. Deshalb ist es sinnvoller, einen oder mehrere Hosts anzupingen, um so die Verfügbarkeit des gesamten Netzes zu beurteilen. Diese sogenannten *Pingnodes* werden über eine Ressource `ping` eingerichtet (siehe Kapitel 8, Abschnitt »ping« auf Seite 260). Dazu ist eine Klon-Ressource nützlich, die wie folgt definiert ist:

```
<clone id="clonePing">
  <primitive id="clone_ping-primitive" class="ocf" provider="pacemaker"
    type="ping">
    <instance_attributes id="clone_ping-primitive-instance_attributes">
      <nvpair id="ping-host_list" name="host_list"
        value="192.168.188.2 192.168.188.3"/>
      <nvpair id="ping-dampen" name="dampen" value="5s"/>
      <nvpair id="ping-multiplier" name="multiplier" value="100"/>
    </instance_attributes>
    <operations>
      <op id="ping-monitor" interval="2s">
    </operations>
  </primitive>
</clone>
```

Oder in CRM-Notation:

```
primitive resPing ocf:pacemaker:ping \
  params host_list="192.168.188.2 192.168.188.3" dampen="5s" multiplier="100" \
  op monitor interval="2s"
clone clonePing resPing
```

Die Klon-Ressource `clonePing` ist so definiert, dass sie auf jedem Knoten genau einmal eine einfache Ressource `ping` startet. Im Beispiel werden die Hosts 192.168.188.2 und 192.168.188.3 angesprochen. Bei Erfolg multipliziert sie die Anzahl der erreichbaren Knoten mit dem Faktor 100 (`multiplier`) und schreibt dies nach einer Schonzeit von 5 Sekunden (`dampen`) in die CIB. Die Schonzeit wird eingebaut, damit

ein einzelner verlorener Ping nicht gleich den ganzen Cluster verunsichert. Wenn der nächste Ping rechtzeitig ankommt, wird der eine Fehler ignoriert.

Die Ressource startet alle `interval` Sekunden einen `ping`, der alle Rechner anspricht, die im Attribut `host_list` aufgeführt sind. Danach wird der Status-Abschnitt des entsprechenden Knotens auf den neuesten Stand gebracht:

```
<status>
  <node_state id="node1" uname="node1" ha="active" in_ccm="true" crmd="online"
    join="member" expected="member" crm-debug-origin="do_update_resource"
    shutdown="0">
    <transient_attributes id="node1">
      <instance_attributes id="status-node1">
        (…)
        <nvpair id="status-node1-pingd" name="pingd" value="200"/>
      </instance_attributes>
    </transient_attributes>
    (…)
  </node_state>
</status>
```

Da beide Ping-Knoten erreichbar sind, erhält der Knoten 200 Punkte für das Attribut `pingd`. Als Beispiel soll nun die Ressource `resMyService` nur auf den Knoten laufen dürfen, die mindestens einen Ping-Knoten erreichen können. Die Regel der Platzierungsbedingung vergibt `-INFINITY` Punkte, wenn das Attribut `pingd` im Kontext des Knotens nicht definiert ist oder weniger als ein Ping-Knoten erreicht wird:

```
<rsc_location id="loc_MyService" rsc="resMyService">
  <rule id="connectedMyService" score="-INFINITY" boolean_op="or">
    <expression id="loc-attr" attribute="pingd" operation="lte" value="0"/>
    <expression id="loc-def" attribute="pingd" operation="not_defined"/>
  </rule>
</rsc_location>
```

In diesem Beispiel wird auch gleich die Verknüpfung von zwei Ausdrücken mit Booleschen Operatoren gezeigt.

Der Name `pingd` für das Attribut stammt noch aus der Zeit, als die Ressource tatsächlich einen eigenen Daemon im Hintergrund startete. In der modernen Variante ruft die Ressource nur noch den `ping`-Befehl jedes Intervalls auf und notiert das Ergebnis.

Noch besser wäre es, den Knoten mit der höchsten Punktzahl auszuwählen anstelle eines Knotens, der nur irgendwie erreichbar ist. Für diesen Spezialfall kann anstelle des Tags `score`, das einen direkten Zahlenwert (hier: `-INFINITY`) angibt, wenn die Regel erfüllt ist, das Tag `score_attribute` für die Bewertung der Regel genutzt werden. Dieses Attribut verweist auf ein anderes Attribut, und der Wert dieses Attributs ergibt die Punktzahl für die Regel. Für die Erreichbarkeit der Knoten kann man das einfacher ausdrücken: Das Attribut `pingd` hat auf jedem Knoten einen Wert

(Anzahl der erreichbaren Knoten * Multiplikator). Der Wert dieses Attributs kann als Punktzahl für die Platzierung direkt übernommen werden. Natürlich muss überprüft werden, ob pingd im Kontext eines Knotens überhaupt definiert ist. Deshalb lautet die bessere Version der Platzierung mit score_attribute:

```
<rsc_location id="loc_MyService" rsc="resMyService">
  <rule id="pingd_rule" score_attribute="pingd">
    <expression id="defined" attribute="pingd" operation="defined"/>
  </rule>
</rsc_location>
```

oder auch wieder in Kurzschreibweise:

```
location loc_MyService resMyService rule pingd: defined pingd
```

Nach dieser Definition wird die Ressource resMyService auf dem Knoten gestartet, der die beste Verbindung zum Netzwerk hat, also die höchste Anzahl an Ping-Knoten sieht. Sinnvoll ist eine solche Definition natürlich dort, wo mehrere Ping-Knoten definiert sind. Damit die Ressource bei kleinen Schwankungen der Erreichbarkeit nicht willkürlich zwischen den Knoten springt, kann im Beispiel eine resource-stickiness von 100 Punkten definiert werden. Dann muss der Unterschied in der Erreichbarkeit schon zwei Ping-Knoten betragen, damit eine Ressource auch den Knoten wechselt.

Frei definierbare Kriterien

Das Beispiel des pingd zeigt noch etwas anderes: Externe Programme (hier die Ressource ping) können eigene Attribute im Status eines Knotens setzen und verändern. Dazu wird das Programm attrd_updater genutzt. Die genaue Befehlssyntax ist in Kapitel 5 im Abschnitt »attrd_updater« auf Seite 141 beschrieben. Die Attribute werden normalerweise im Abschnitt <transient_attributes> des Status des Knotens aufgelistet, auf dem der Befehl ausgeführt wurde. Ein

```
node02:~# attrd_updater -n myTestAttribute -v true
```

hinterlässt im Status von node02 folglich:

```
<transient_attributes id="id-transient_attributes">
  <instance_attributes id="id-instance_attributes">
    <attributes>
      <nvpair id="id-status" name="myTestAttribute" value="true"/>
    </attributes>
  </instance_attributes>
</transient_attributes>
```

Ein eigenes Skript kann den Befehl nutzen, um Informationen an die CIB weiterzugeben. Diese Einträge in der CIB können von Bedingungen (speziell Platzierungen) ausgewertet werden. Wenn der Administrator die Ressource resRadius bevorzugt dort laufen lassen will, wo myTestAttribute="true" ist, könnte die Bedingung also wie folgt lauten:

```
<rsc_location id="loc_Radius" rsc="resRadius">
  <rule id="prefered_location_radius" score="100">
    <expression id="id-expression" attribute="myTestAttribute" operation="eq"
      value="true"/>
  </rule>
</rsc_location>
```

oder in Kurzform:

```
location loc_Radius resRadius rule 100: myTestAttribute eq true
```

Failover erst nach »N« Fehlern

Oft will der Administrator eine Ressource bei einem Fehler auf einem Knoten erst einmal durchstarten, bevor sie auf einen anderen Knoten umzieht. Dazu bietet der Cluster-Manager einen Zähler, der registriert, wie oft eine bestimmte Ressource auf einem bestimmten Knoten einen Fehler gemeldet hat.

Jede Ressource hat auf jedem Knoten ihren eigenen Fehlerzähler (`failcounter`). Zusätzlich existiert ein Meta-Attribut (`migration-threshold`), das direkt der Anzahl Fehler entspricht, die Ressourcen auf einem Knoten haben dürfen.

Bei jedem Fehler, den die Überwachung einer Ressource zurückmeldet, zählt der Fehlerzähler der Ressource eins weiter. Bei einem Fehler beim Start einer Ressource wird der Fehlerzähler sofort auf `INFINITY` gesetzt, und die Ressource versucht, auf einem anderen Knoten zu starten. Falls ein Fehler beim Anhalten einer Ressource (`stop`) auftritt, hat der Cluster zwei Möglichkeiten: Wenn STONITH konfiguriert ist, versucht der Cluster, den Knoten abzuschalten und die Ressource auf einem anderen Knoten neu zu starten. Anderenfalls markiert der Cluster-Manager die Ressource für diesen Knoten mit `is-managed="false"`.

Wenn zum Beispiel die Ressource `resIP` auf einem Knoten bei der Überwachung (`monitor` Operation) einen Fehler meldet, hinterlässt das folgenden Eintrag im Status-Abschnitt der CIB für den entsprechenden Knoten:

```
<status>
  <node_state id="node1" uname="node1" (...)>
    <transient_attributes id="node1">
      <instance_attributes id="status-node1">
        <nvpair id="status-node1-fail-count-resIP" name="fail-count-resIP"
          value="1"/>
        <nvpair id="status-node1-last-failure-resIP" name="last-failure-resIP"
          value="1306813592"/>
      </instance_attributes>
    </transient_attributes>
    (…)
  </node_state>
</status>
```

Der Cluster merkt sich nicht nur die Anzahl der Fehler, sondern auch, wann der letzte Fehler aufgetreten ist. Das macht die Fehlersuche in den Logdateien natürlich besonders einfach.

Falls der Wert des Fehlerzählers größer als die migration-threshold oder gleich ist, wird sich die Ressource einen neuen Knoten suchen. Folgendes Beispiel zum Fehlerzähler: pacemaker überwacht eine Ressource, die wie folgt definiert ist, alle 10 Sekunden:

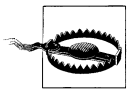
```
primitive resIP ocf:heartbeat:IPaddr2 \
  params ip="172.56.19.254" nic="eth0" cidr_netmask="24" \
  op monitor interval="10s" \
  meta migration-threshold="3"
```

Sollte der Knoten aus irgendeinem Grund die IP-Adresse vergessen haben, setzt der Cluster die Adresse einfach neu (on-fail="restart" ist die Vorgabe!). Am einfachsten kann man das mit einem

```
# ip addr delete 172.56.19.254/24 dev eth0
```

ausprobieren.

Das macht der Cluster genau dreimal. Beim dritten Fehler wechselt die Ressource den Knoten. Überlicherweise muss dann der Administrator eingreifen und sollte die tatsächliche Ursache für den Fehler suchen und beseitigen. Erst danach darf er den Fehlerzähler manuell zurücksetzen. Der Cluster nimmt ihm diese Arbeit nicht ab.



Der Fehlerzähler wird nicht automatisch durch den Cluster zurückgesetzt. Somit kann eine Ressource normalerweise nicht auf den ursprünglichen Knoten zurückkehren, da ja der Fehlerzähler immer noch über dem Grenzwert liegt. Dies soll die Rückkehr der Ressource auf einen Knoten verhindern, auf dem die Ursache für das Versagen nicht beseitigt ist. Sobald sich der Administrator darum gekümmert hat, kann er ja auch den Zähler zurücksetzen.

Der Fehlerzähler kann mit dem Kommando

```
# crm resource failcount resIP show node1
scope=status name=fail-count-resIP value=1
```

angezeigt und auch mit

```
# crm resource failcount resIP delete node1
```

wieder zurückgesetzt werden. Näheres zu diesem Kommando finden Sie in Kapitel 5 im Abschnitt »Die Subshell zum CRM« auf Seite 151. Bei Problemen im Cluster sollte der Administrator diesen Fehlerzähler immer abfragen. Einen schnellen Überblick über alle Fehlerzähler bietet auch das Kommando

```
# crm_mon -1f
```

In pacemaker haben die Entwickler aber noch eine Hintertür für eine mögliche automatische Bereinigung eingebaut. Neben dem Meta-Attribut migration-threshold einer Ressource kann man ein failure-timeout definieren. Da sich der Cluster neben dem Stand des Fehlerzählers auch den Zeitpunkt des letzten Fehlers merkt,

bewirkt ein `failure-timeout="300s"`, dass die Fehler nach 5 Minuten wieder »vergessen« werden und die Ressource nach dieser Zeit auf den ursprünglichen Knoten zurückkehren kann.

Systemgesundheit

In Version 1.0.4 von `pacemaker` ist eine weitere Möglichkeit hinzugekommen, den Cluster über den Gesundheitszustand von Knoten zu unterrichten. Der Cluster kann aufgrund dieser Informationen Ressourcen verschieben.

Externe Dienste überwachen alle Knoten des Clusters permanent und hinterlegen ihre Ergebnisse als Werte in der CIB analog zum `ping`. Die Attribute, die für die Überprüfung des Gesundheitszustands verwendet werden, beginnen alle mit der Kennung `#health`. So können verschiedene Dienste diese Schnittstelle im Cluster nutzen. Denkbar sind Daemons für die Hardware (z. B. `#health-ipmi`), für die Überprüfung von Festplatten mit `smartd` (`#health-smartd`) oder eigene Skripte des Administrators (`#health-myscript`). Jeder dieser Dienste schreibt seinen Status in der Form `red`, `yellow` oder `green` in diese Attribute. Der Cluster-Manager bezieht sie in seine Überlegung zur optimalen Platzierung von Ressourcen im Cluster mit ein. Der Administrator gibt die Richtung mit dem Attribut `node-health-strategy` in der Konfiguration des Clusters an. Möglich sind die Werte:

`none`
: Dieser Wert ist die Vorgabe und gibt an, dass der Cluster die zusätzlichen Informationen nicht auswerten soll.

`migrate-on-red`
: Wenn ein einziges der `#health`-Attribute den Wert `red` hat, erhalten alle Ressourcen auf diesem Knoten `-INFINITY` Punkte und müssen sich daher einen anderen Knoten suchen. Sind die Attribute `yellow` oder `green`, gibt es keine Punkte.

`only-green`
: Die Höchststrafe von `-INFINITY` Punkten gibt es nicht nur bei `red`, sondern auch schon bei `yellow`. Die Ressourcen können also nur auf einem vollständig gesunden Knoten laufen.

`progressive`
: Die Punkte, die bei `red`, `yellow` oder `green` vergeben werden, übernimmt die *Policy Engine* aus der Konfiguration des Clusters. Die Werte finden sich im Abschnitt `crm_config`. Die Vorgaben für die einzelnen Zustände sind:

```
node-health-red         -INFINITY
node-health-yellow      0
node-health-green       0
```

`custom`
: Bei dieser Einstellung kann der Administrator die Regeln selbst bestimmen.

Da das ganze System der Gesundheitsüberprüfung der Knoten noch relativ neu ist, existieren auch noch keine Dienste, die die `#health`-Attribute entsprechend setzen können. Es ist geplant, entsprechende Agenten zu veröffentlichen.

Jeder Administrator kann aber seine eigenen Skripte schreiben, die den `attrd_updater` nutzen. Zum Testen wird einfach eine `Dummy`-Ressource angelegt und gestartet. Mit der Einstellung

```
# crm configure property node-health-strategy="migrate-on-red"
```

gibt der Administrator vor, dass der Cluster die Attribute auswerten soll, und infiziert einen Knoten anschließend vorsätzlich:

```
# attrd_updater -n "#health-my" -v red
```

Der Cluster-Manager verschiebt alle Ressourcen auf einen anderen Knoten. Um den Knoten zu heilen, löscht man einfach das Attribut wieder:

```
# attrd_updater -n "#health-my" -D
```

Leider funktioniert die Idee, den Cluster per Systemgesundheit zu überwachen, noch nicht perfekt. Der Systemstatus wird üblicherweise mit Ressourcen überwacht. Diese Ressourcen sind natürlich ebenso vom allgemeinen Umzug bei einem Fehler betroffen. Auf dem Knoten verbleibt also keine Ressource, die das entsprechende `#health`-Attribut wieder auf `green` setzen könnte, wenn das Problem behoben ist. Das erfordert dann noch den manuellen Eingriff des Administrators. Alternativ könnte man die Überwachung auch einem `cron`-job übertragen.

Die Entwickler wollen das Problem in der nächsten Version 1.3 angehen, indem sie eine neue Klasse für die `health`-Agenten definieren. Diese Klasse wäre dann von der `node-health-strategy` nicht betroffen.