Red Hat Enterprise Linux 6

Pacemaker を使用した Red Hat High Availability Add-On の設定

Red Hat Enterprise Linux 6 向け High Availability Add-On リファレンス
エディッション 1

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 Pacemaker を使用した Red Hat High Availability Add-On の設定

Red Hat Enterprise Linux 6 向け High Availability Add-On リファレンス
エディッション 1

法律上の通知

概要

Pacemaker を使用した Red Hat High Availability Add-On の設定 では Pacemaker を使って Red Hat High Availability Add-On を設定する方法について説明しています。

目次

はじめに

本ガイドでは Red Hat High Availability Add-On コンポーネントのインストール、設定、管理を行う方法について説明しています。Red Hat High Availability Add-On コンポーネントを利用すると複数のコンピューター (*ノード* または *メンバー* と呼ばれる) のグループを繋げクラスターとして機能させることができます。本ガイドで **クラスター** または **クラスター群** と表記している場合、Red Hat High Availability Add-On を実行しているコンピューターの集合を指しています。

本書は Red Hat Enterprise Linux について高度な運用知識があり、クラスター、ストレージ、サーバーコンピューティングの概念を理解している方を対象としています。

Red Hat Enterprise Linux 6 の詳細については、以下の資料を参照してください。

- 『Red Hat Enterprise Linux インストールガイド』 — Red Hat Enterprise Linux 6 のインストールに関して詳しく記載しています。
- 『Red Hat Enterprise Linux 導入ガイド』 — Red Hat Enterprise Linux 6 の導入、設定、管理に関して詳しく記載しています。

Red Hat Enterprise Linux 6 向けの High Availability Add-On および関連製品については、以下の資料を参照してください。

- 『High Availability Add-On の概要』 — Red Hat High Availability Add-On について高いレベルで概要を説明しています。
- 『クラスターの管理』 — High Availability Add-On のインストール、設定、および管理に関して詳しく記載しています。
- 『論理ボリュームマネージャーの管理』 — クラスター化した環境での論理ボリュームマネージャー (LVM) の実行方法など、LVM について詳しく記載しています。
- 『Global File System 2: 設定と管理』 — Resilient Storage Add-On に同梱されている Red Hat GFS2 (Red Hat Global File System 2) のインストール、設定、メンテナンスに関して詳しく記載しています。
- 『DM マルチパス機能』 — Red Hat Enterprise Linux 6 の機能となる Device-Mapper Multipath (デバイスマッパーマルチパス) の使い方について説明しています。
- 『ロードバランサーの管理』 — 複数の実サーバー全体で IP の負荷分散を行う Linux Virtual Servers (LVS) を提供している統合ソフトウェアコンポーネントセット、Load Balancer Add-On を使ってパフォーマンスの高いシステムおよびサービスを構成する方法について記載しています。
- 『リリースノート』 — Red Hat 製品の現在のリリースに関する情報を提供します。

Red Hat Cluster Suite ドキュメントおよび他の Red Hat ドキュメントは、HTML、PDF、RPM の各形式で Red Hat Enterprise Linux ドキュメント CD またはオンラインの https://access.redhat.com/site/documentation/ からご覧いただけます。

1. フィードバック

本書内で誤字、脱字を発見された場合、また本書に対する改善のご意見などがございましたらぜひ弊社にご連絡ください。ご報告は Bugzilla (http://bugzilla.redhat.com/bugzilla/) でお受けしています。製品には **Red Hat Enterprise Linux 6**、コンポーネントには **doc-Cluster_General** を選択してください。

以下のマニュアル識別子を忘れないようご記入お願いします。

```
Configuring_High_Availability_With_Pacemaker(EN)-6 (2014-8-7T16:26)
```

マニュアル識別子を記入して頂くことでご報告いただいているガイドのバージョンを確認することができます。

本書に関して改善のご提案がある場合は、できるだけ具体的にご提案頂けるようお願いします。エラーを発見された場合は、該当のセクション番号および前後の文章も含めて報告頂けるとより迅速に対応することができます。

第1章 RED HAT HIGH AVAILABILITY ADD-ON の設定と管理のリファレンス概要

Pacemaker を使用した Red Hat High Availability Add-On で対応しているオプションと機能について解説しています。

Red Hat Enterprise Linux リリース 6.6 以降の **pcs** 設定インターフェースの使い方について説明しています。

注記

High Availability Add-On および Red Hat Global File System 2 (GFS2) を使用した Red Hat Enterprise Linux クラスターの導入やアップグレードの成功事例については Red Hat カスタマーポータルに掲載の「Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Best Practices」の記載を参照してください。
(https://access.redhat.com/kb/docs/DOC-40821)

1.1. PACEMAKER 設定ツールのインストール

以下の **yum install** コマンドを使って Red Hat High Availability Add-On ソフトウェアのパッケージおよび利用可能なフェンスエージェントを High Availability チャンネルからインストールします。

```
# yum install pcs fence-agents
```

lvm2-cluster と **gfs2-utils** のパッケージは ResilientStorage チャンネルの一部になります。必要に応じて次のコマンドでインストールを行ってください。

```
# yum install lvm2-cluster gfs2-utils
```

警告

Red Hat High Availability Add-On パッケージのインストール後、必ずソフトウェア更新に関する設定で自動インストールが行われないよう設定してください。実行中のクラスターでインストールが行われると予期しない動作の原因となる場合があります。

1.2. ファイアウォールでクラスターコンポーネントを許可する IPTABLES 設定

Red Hat High Availability Add-On を使用する場合は次のポートを有効にしておく必要があります。

- TCP: ポート 2224、3121、21064
- UDP: ポート 5405

1.3. クラスターと PACEMAKER の設定ファイル

Red Hat High Availability add-on の設定ファイルは **cluster.conf** と **cib.xml** です。設定ファイルは直接編集せず、**pcs** や **pcsd** などのインターフェースを使用してください。

Pacemaker がビルドするクラスターマネージャーの **corosync** で使用されるクラスターパラメーターは **cluster.conf** ファイルで提供されます。

クラスターの構成およびクラスター内の全リソースの現状を表すのが **cib.xml** ファイルです。Pacemaker のクラスター情報ベース (Cluster Information Base -CIB) で使用されます。CIB のコンテンツはクラスター全体で継続的に自動同期されます。

第2章 PCS コマンドラインインターフェース

pcs コマンドラインインターフェースで **corosync** や **pacemaker** の制御や設定を行うことができます。

pcs コマンドの一般的な形式を以下に示します。

```
pcs [-f file] [-h] [commands]...
```

2.1. PCS コマンド

pcs コマンドを以下に示します。

- **cluster**

  クラスターオプションおよびノードの設定を行います。**pcs cluster** コマンドの詳細については 3章 *クラスターの作成と管理* を参照してください。

- **resource**

  クラスターリソースの作成と管理を行います。**pcs cluster** コマンドの詳細については 5章 *クラスターリソースの設定*、7章 *クラスターリソースの管理*、8章 *高度なリソースタイプ* などを参照してください。

- **stonith**

  Pacemaker との使用に備えてフェンスデバイスを設定します。**pcs stonith** コマンドについては 4章 *フェンス機能: STONITH の設定* を参照してください。

- **constraint**

  リソースの制約を管理します。**pcs constraint** コマンドについては 6章 *リソースの制約* を参照してください。

- **property**

  Pacemaker のプロパティを設定します。**pcs property** コマンドでプロパティを設定する方法については 10章 *Pacemaker クラスターのプロパティ* を参照してください。

- **status**

  現在のクラスターとリソースの状態を表示します。**pcs status** コマンドについては 「状態の表示」 を参照してください。

- **config**

  ユーザーが読みやすい形式でクラスターの全設定を表示します。**pcs config** コマンドについては 「全クラスター設定の表示」 を参照してください。

2.2. PCS の使用に関するヘルプ表示

pcs の **-h** オプションを使用すると **pcs** のパラメーターとその詳細を表示させることができます。例えば、次のコマンドでは **pcs resource** コマンドのパラメーターが表示されます。次の例は出力の一部です。

```
# pcs resource -h
```

```
Usage: pcs resource [commands]...
Manage pacemaker resources
Commands:
    show [resource id] [--all]
        Show all currently configured resources or if a resource is specified
        show the options for the configured resource.  If --all is specified
        resource options will be displayed

    start <resource id>
        Start resource specified by resource_id
...
```

2.3. 生のクラスター設定の表示

クラスター設定ファイルは直接編集すべきではありませんが、**pcs cluster cib** コマンドで生のクラスター設定を表示させることができます。

「設定の変更をファイルに保存する」 に記載されているように、**pcs cluster cib *filename*** を使うと生のクラスター設定を指定ファイルに保存することができます。

2.4. 設定の変更をファイルに保存する

pcs コマンドを使用する際、**-f** オプションを使うとアクティブの CIB に影響を与えずに設定変更をファイルに保存することができます。

クラスターの設定を既に行っているためアクティブな CIB が存在する場合は次のコマンドを使って生の xml ファイルを保存することができます。

```
pcs cluster cib filename
```

例えば、次のコマンドを使用すると CIB の生 xml が **testfile** という名前のファイルに保存されます。

```
pcs cluster cib testfile
```

次のコマンドでは **testfile1** ファイル内にリソースをひとつ作成しています。ただし、そのリソースは現在実行中のクラスター構成には追加されません。

```
# pcs -f testfile1 resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.120
cidr_netmask=24 op monitor interval=30s
```

次のコマンドで **testfile** の現在のコンテンツを CIB にプッシュします。

```
pcs cluster cib-push filename
```

2.5. 状態の表示

次のコマンドでクラスターおよびクラスターリソースの状態を表示します。

```
pcs status commands
```

commands パラメーターを指定しないとクラスターおよびリソースの全情報が表示されます。**resources**、**groups**、**cluster**、**nodes**、**pcsd** などを指定すると特定のクラスターコンポーネントの状態のみを表示させることができます。

2.6. 全クラスター設定の表示

現在の全クラスター設定を表示させる場合は次のコマンドを使います。

```
pcs config
```

2.7. 現在の PCS バージョンの表示

実行中の **pcs** の現行バージョンを表示します。

```
pcs --version
```

第3章 クラスターの作成と管理

本章ではクラスターの作成、クラスターコンポーネントの管理、クラスターの状態表示など Pacemaker で行うクラスターの基本的な管理について見ていきます。

3.1. クラスターの作成

クラスターを作成するため次のステップを行って行きます。

1. クラスターを構成するノードを認証します。
2. クラスターノードの設定と同期を行います。
3. クラスターノードでクラスターサービスを起動します。

次のセクションでは、上記の手順で使用するコマンドについて詳しく見ていきます。

3.1.1. クラスターノードの認証

次のコマンドではクラスター内のノード上にある **pcs** デーモンに対して **pcs** の認証を行います。

- **pcs** 管理者のユーザー名はすべてのノードで **hacluster** にしてください。ユーザー **hacluster** のパスワードも各ノードで同じパスワードを使用されることをお勧めします。
- ユーザー名やパスワードを指定しないと、コマンドを実行した際に各ノードごとのユーザー名やパスワードのパラメーター入力が求められます。
- ノードを指定しないと、前回実行した **pcs cluster setup** コマンドで指定されているノードの **pcs** を認証することになります。

```
pcs cluster auth [node] [...] [-u username] [-p password]
```

認証トークンが **~/.pcs/tokens** (または **/var/lib/pcsd/tokens**) ファイルに格納されます。

3.1.2. クラスターノードの設定と起動

次のコマンドでクラスター設定ファイルの構成、指定ノードに対する設定の同期を行います。

- **--start** オプションを使用すると指定ノードでクラスターサービスが起動されます。必要に応じて、別途 **pcs cluster start** コマンドを使ってクラスターサービスを起動させることもできます。
- **--local** オプションを使用するとローカルノードでのみ変更を実行します。

```
pcs cluster setup [--start] [--local] --name cluster_ name node1 [node2] [...]
```

次のコマンドは指定ノード (複数指定可) でクラスターサービスを起動します。

- **--all** オプションを使用すると全ノードでクラスターサービスを起動します。
- ノードを指定しないとクラスターサービスはローカルのノードでしか起動されません。

```
pcs cluster start [--all] [node] [...]
```

3.2. クラスターノードの管理

次のセクションではクラスターサービスの起動や停止、クラスターノードの追加や削除などクラスターノードの管理で使用するコマンドについて説明します。

3.2.1. クラスターサービスの停止

次のコマンドで指定ノード (複数指定可) のクラスターサービスを停止します。**pcs cluster start** と同様に **--all** オプションを使うと全ノードのクラスターサービスが停止されます。ノードを指定しない場合はローカルノードのクラスターサービスのみが停止されます。

```
pcs cluster stop [--all] [node] [...]
```

次のコマンドでローカルノードでのクラスターサービスの停止を強制することができます。このコマンドは **kill -9** コマンドを実行します。

```
pcs cluster kill
```

3.2.2. クラスターサービスを有効または無効にする

指定ノード (複数指定可) の起動時にクラスターサービスが実行されるよう設定する場合は次のコマンドを使用します。

- **--all** オプションを使用すると全ノードでクラスターサービスが有効になります。
- ノードを指定しないとローカルノードでのみクラスターサービスが有効になります。

```
pcs cluster enable [--all] [node] [...]
```

指定ノード (複数指定可) の起動時にクラスターサービスが実行されないよう設定する場合は次のコマンドを使用します。

- **--all** オプションを使用すると全ノードでクラスターサービスが無効になります。
- ノードを指定しないとローカルノードでのみクラスターサービスが無効になります。

```
pcs cluster disable [--all] [node] [...]
```

3.2.3. クラスターノードの追加と削除

次のコマンドで既存のクラスターに新規ノードを追加します。このコマンドは追加しているノードを含めクラスター内の全ノードに対して **cluster.conf** クラスター設定ファイルの同期も行います。

```
pcs cluster node add node
```

次のコマンドは指定ノードをシャットダウンしてクラスター内の他のノードの **cluster.conf** クラスター設定ファイルからシャットダウンしたノードを削除します。クラスターノードからクラスターに関する全情報を完全に削除し永久的にクラスターを破棄する方法については 「クラスター設定の削除」を参照してください。

```
pcs cluster node remove node
```

3.2.4. スタンバイモード

次のコマンドでは指定したノードをスタンドバイモードにします。指定ノードはリソースのホストが行えなくなります。このノード上で現在実行中のリソースはすべて別のノードに移行されます。**--all** を使用すると全ノードがスタンバイモードになります。

リソースのパッケージを更新する場合にこのコマンドを使用します。また、設定をテストして実際にはノードのシャットダウンを行わずにリカバリーのシュミレーションを行う場合にも使用できます。

```
pcs cluster standby node | --all
```

次のコマンドは指定したノードのスタンバイモードを外します。コマンドを実行すると指定ノードはリソースをホストできるようになります。**--all** を使用すると全ノードのスタンバイモードを外します。

```
pcs cluster unstandby node | --all
```

pcs cluster standby コマンドを実行すると指定したノードでのリソースの実行を阻止する制約が追加されることになります。制約を取り除く場合は **pcs cluster unstandby** コマンドを実行します。このコマンドは必ずしもリソースを指定ノードに戻すわけではありません。最初にどのようにリソースを設定したかにより、その時点で実行できるノードに移動されます。リソースの制約については 6章 *リソースの制約* を参照してください。

3.3. ユーザーのパーミッション設定

Red Hat Enteprise Linux 6.6 からは **pcs acl** コマンドを使ったローカルユーザーのパーミッション設定が可能になり、アクセス制御一覧でクラスター設定に対する読み取り専用アクセスや読み取りと書き込みのアクセスを許可することができるようになります。

ローカルユーザーのパーミッションを設定するには以下の 2 ステップに従います。

1. **pcs acl role create...** コマンドを実行して *role* を作成しそのロールのパーミッションを定義します。
2. **pcs acl user create** コマンドで作成したロールをユーザーに割り当てます。

以下の例では **rouser** というローカルユーザーにクラスター設定に対する読み取り専用アクセスを与えています。

1. この場合、**rouser** ユーザーがローカルシステムに存在していること、また **rouser** ユーザーが **hacluster** グループのメンバーでなければなりません。

   ```
   # adduser rouser
   # usermod -a -G hacluster rouser
   ```

2. **enable-acl** クラスタープロパティを使って Pacemaker ACL を有効にします。

   ```
   # pcs property set enable-acl=true --force
   ```

3. cib に対して読み取り専用のパーミッションを持つ **read-only** という名前のロールを作成します。

   ```
   # pcs acl role create read-only description=”Read access to cluster” read xpath
   /cib
   ```

4. pcs ACL システム内に **rouser** というユーザーを作成し、**read-only** ロールを割り当てます。

```
# pcs acl user create rouser read-only
```

5. 現在の ACL を表示させます。

```
# pcs acl
User: rouser
  Roles: read-only
Role: read-only
  Description: Read access to cluster
  Permission: read xpath /cib (read-only-read)
```

次の例では **wuser** と言うローカルユーザーにクラスター設定に対する書き込みアクセスを与えています。

1. この場合、**wuser** ユーザーがローカルシステムに存在していること、また **wuser** ユーザーが **hacluster** グループのメンバーでなければなりません。

```
# adduser wuser
# usermod -a -G hacluster wuser
```

2. **enable-acl** クラスタープロパティを使って Pacemaker ACL を有効にします。

```
# pcs property set enable-acl=true --force
```

3. cib に対して書き込みパーミッションを持つ **write-access** という名前のロールを作成します。

```
# pcs acl role create write-access description="Full access" write xpath /cib
```

4. pcs ACL システム内に **wuser** というユーザーを作成し、**write-access** ロールを割り当てます。

```
# pcs acl user create wuser write-access
```

5. 現在の ACL を表示させます。

```
# pcs acl
User: rouser
  Roles: read-only
User: wuser
  Roles: write-access
Role: read-only
  Description: Read access to cluster
  Permission: read xpath /cib (read-only-read)
Role: write-access
  Description: Full Access
  Permission: write xpath /cib (write-access-write)
```

クラスター ACL の詳細については **pcs acl** コマンドのヘルプ画面を参照してください。

3.4. クラスター設定の削除

クラスター設定ファイルをすべて削除し全クラスターサービスを停止、クラスターを永久的に破棄する場合は次のコマンドを使用します。

警告

作成したクラスター設定をすべて永久に削除します。先に **pcs cluster stop** を実行してからクラスターの破棄を行うことを推奨しています。

```
pcs cluster destroy
```

3.5. クラスターの状態表示

次のコマンドで現在のクラスターとクラスターリソースの状態を表示します。

```
pcs status
```

次のコマンドを使用すると現在のクラスターの状態に関するサブセット情報を表示させることができます。

このコマンドはクラスターの状態は表示しますがクラスターリソースの状態は表示しません。

```
pcs cluster status
```

クラスターリソースの状態は次のコマンドで表示させます。

```
pcs status resources
```

第4章 フェンス機能: STONITH の設定

STONITH とは Shoot-The-Other-Node-In-The-Head の略です。問題のあるノードや同時アクセスによるデータ破損を防止します。

ノードが無反応だからと言ってデータにアクセスしていないとは限りません。STONITH を使ってノードを排他処理することが唯一 100% 確実にデータの安全を確保する方法になります。排他処理することによりそのノードを確実にオフラインにしてから、別のノードに対してデータへのアクセスを許可することができます。

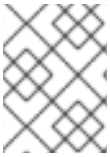
STONITH はクラスター化したサービスを停止できない場合にも役に立ちます。このようなことが発生した場合は、STONITH で全ノードを強制的にオフラインにしてからサービスを別の場所で開始すると安全です。

4.1. STONITH (フェンス) エージェント

次のコマンドは利用できる全 STONITH エージェントを表示します。フィルターを指定するとフィルターに一致する STONITH エージェントのみを表示します。

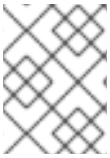
```
pcs stonith list [filter]
```

4.2. フェンスデバイスの汎用プロパティ



注記

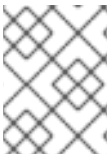
フェンスデバイスやフェンスリソースを無効にする場合にも通常のリソースと同様、**target-role** を設定することができます。



注記

特定のノードにフェンスデバイスを使用させないようにする場合はフェンスリソースに場所の制約を使用すると期待通りに動作します。

フェンスデバイスに設定できる汎用プロパティについては 表4.1「フェンスデバイスの汎用プロパティ」 に記載されています。特定のフェンスデバイスに設定できるフェンスプロパティについては「デバイス固有のフェンスオプションを表示する」 を参照してください。



注記

より高度なフェンス設定プロパティについては 「その他のフェンス設定オプション」 を参照してください。

表4.1 フェンスデバイスの汎用プロパティ

フィールド	タイプ	デフォルト	説明
stonith-timeout	時間	60s	STONITH 動作の完了を待機させる時間を stonith デバイスごと指定します。**stonith-timeout** クラスタープロパティが上書きされます。

フィールド	タイプ	デフォルト	説明
priority	整数	0	stonith リソースの優先度です。高い優先度のデバイスから順番に試行されます。
pcmk_host_map	文字列		ホスト名に対応していないデバイスのポート番号とホスト名をマッピングします。例えば、**node1:1;node2:2,3** なら node1 にはポート 1 を使用し node2 にはポート 2 と 3 を使用するようクラスターに指示します。
pcmk_host_list	文字列		このデバイスで制御するマシンの一覧です (オプション、**pcmk_host_check=static-list** を除く)。
pcmk_host_check	文字列	dynamic-list	デバイスで制御するマシンを指定します。使用できる値: **dynamic-list** (デバイスに問い合わせ)、**static-list** (**pcmk_host_list** 属性をチェック)、なし (すべてのデバイスで全マシンのフェンスが可能とみなされる)

4.3. デバイス固有のフェンスオプションを表示する

指定した STONITH エージェントのオプションを表示するには次のコマンドを使用します。

```
pcs stonith describe stonith_agent
```

次のコマンドでは telnet または SSH 経由の APC 用フェンスエージェントのオプションを表示します。

```
# pcs stonith describe fence_apc
Stonith options for: fence_apc
  ipaddr (required): IP Address or Hostname
  login (required): Login Name
  passwd: Login password or passphrase
  passwd_script: Script to retrieve password
  cmd_prompt: Force command prompt
  secure: SSH connection
  port (required): Physical plug number or name of virtual machine
  identity_file: Identity file for ssh
  switch: Physical switch number on device
  inet4_only: Forces agent to use IPv4 addresses only
  inet6_only: Forces agent to use IPv6 addresses only
  ipport: TCP port to use for connection with device
  action (required): Fencing Action
  verbose: Verbose mode
  debug: Write debug information to given file
  version: Display version information and exit
  help: Display help and exit
  separator: Separator for CSV created by operation list
  power_timeout: Test X seconds for status change after ON/OFF
  shell_timeout: Wait X seconds for cmd prompt after issuing command
  login_timeout: Wait X seconds for cmd prompt after login
```

```
power_wait: Wait X seconds after issuing ON/OFF
delay: Wait X seconds before fencing is started
retry_on: Count of attempts to retry power on
```

4.4. フェンスデバイスを作成する

次のコマンドで stonith デバイスを作成します。

```
pcs stonith create stonith_id stonith_device_type [stonith_device_options]
```

```
# pcs stonith create MyStonith fence_virt pcmk_host_list=f1 op monitor interval=30s
```

ノードごと異なるポートを使って複数のノードに一つのフェンスデバイスを使用する場合は各ノードごと別々にデバイスを作成する必要はありません。**pcmk_host_map** オプションを使ってポートとノードのマッピングを指定することができます。例えば、次のコマンドでは **myapc-west-13** と言う名前のフェンスデバイスを一つ作成し、**west-apc** という名前の APC 電源スイッチを使用、**west-13** というノードにポート 15 を使用させます。

```
# pcs stonith create myapc-west-13 fence_apc pcmk_host_list="west-13" ipaddr="west-apc"
login="apc" passwd="apc" port="15"
```

ただし、次の例では **west-apc** と言う名前の APC 電源スイッチを使用してノード名 **west-13** をポート 15で、ノード名 **west-14** をポート 17で、ノード名 **west-15** をポート 18 で、ノード名 **west-16** をポート 19 でそれぞれ排他処理します。

```
# pcs stonith create myapc fence_apc pcmk_host_list="west-13,west-14,west-15,west-16"
pcmk_host_map="west-13:15;west-14:17;west-15:18;west-16:19" ipaddr="west-apc" login="apc"
passwd="apc"
```

4.5. ストレージベースのフェンスデバイスをアンフェンスを使って設定する

SAN やストレージフェンスデバイス (つまり電源ベース以外のフェンスエージェント) を作成する場合には、**stonith** デバイスを作成する際にメタオプション **provides=unfencing** を設定する必要があります。これにより排他処理されるノードが再起動前に確実にアンフェンスされ、クラスターサービスがそのノードで起動されるようになります。

電源ベースのフェンスデバイスを設定する場合はデバイス自体がノードの起動 (およびクラスターへの再ジョイン試行) に電力を供給するため **provides=unfencing** メタオプションを設定する必要はありません。この場合の起動とはアンフェンスが発生してから起動するという意味です。

次のコマンドでは **fence_scsi** フェンスエージェントを使用する **my-scsi-shooter** と言う名前の stonith デバイスを設定、デバイスのアンフェンスを有効にしています。

```
pcs stonith create my-scsi-shooter fence_scsi devices=/dev/sda meta provides=unfencing
```

4.6. フェンスデバイスを表示する

次のコマンドは現在設定されている全フェンスデバイスを表示します。*stonith_id* を指定すると設定されたその stonith デバイスのオプションのみを表示します。**--full** オプションを指定すると設定された全 stonith オプションを表示します。

```
pcs stonith show [stonith_id] [--full]
```

4.7. フェンスデバイスの修正と削除

現在設定されているフェンスデバイスのオプションを修正したり、新たにオプションを追加する場合は次のコマンドを使用します。

```
pcs stonith update stonith_id [stonith_device_options]
```

現在の設定からフェンスデバイスを削除する場合は次のコマンドを使用します。

```
pcs stonith delete stonith_id
```

4.8. フェンスデバイスが接続されているノードの管理

手作業でのノードの排他処理は次のコマンドで行うことができます。**--off** を使用すると stonith に対して **off** API コールが使用されノードは再起動ではなく電源オフになります。

```
pcs stonith fence node [--off]
```

次のコマンドで指定したノードの電源が現在オフになっているのかどうかを確認することができます。

注記

指定したノードがクラスターソフトウェアや通常クラスターで制御しているサービスを実行中だった場合はデータ破損やクラスター障害が発生するので注意してください。

```
pcs stonith confirm node
```

4.9. その他のフェンス設定オプション

フェンスデバイスに設定できるその他のオプションを 表4.2「フェンスデバイスの高度なプロパティ」にまとめています。その他のオプションは高度な設定を行う場合に限られます。

表4.2 フェンスデバイスの高度なプロパティ

フィールド	タイプ	デフォルト	説明
pcmk_host_argument	文字列	port	ポートの代わりに与える代替のパラメーターです。標準的なポートパラメーターに対応していないデバイスやポート以外の別のパラメーターを提供しているデバイスがあります。このような場合、このパラメーターを使って排他処理するマシンを示すデバイス固有の代替パラメーターを指定します。追加パラメーターを与えないことを指示する場合は値を **none** にします。

フィールド	タイプ	デフォルト	説明
pcmk_reboot_action	文字列	reboot	**reboot** の代わりに実行する代替のコマンドです。標準的なコマンドに対応していないデバイスや別のコマンドを提供しているデバイスがあります。このような場合、このパラメーターを使って再起動の動作を実行するデバイス固有の代替コマンドを指定します。
pcmk_reboot_timeout	時間	60s	**stonith-timeout** の代わりに再起動の動作に対して使用する代替タイムアウトです。再起動が完了するまでに通常より長い時間を要するデバイスもあれば通常より短い時間で完了するデバイスもあります。再起動の動作に対してデバイス固有の代替タイムアウトを指定する場合に使用します。
pcmk_reboot_retries	整数	2	タイムアウト期間内で **reboot** コマンドを再試行させる最大回数です。複数接続に対応していないデバイスがあります。別のタスクでビジー状態になるとデバイスが動作に失敗する場合があるため、Pacemaker は残り時間内で動作を自動的に再試行させます。Pacemaker による再起動の動作の再試行回数を変更する場合に使用します。
pcmk_off_action	文字列	オフ	**off** の代わりに実行する代替コマンドです。標準的なコマンドに対応していないデバイスや別のコマンドを提供しているデバイスがあります。このような場合、このパラメーターを使ってオフの動作を実行するデバイス固有の代替コマンドを指定します。
pcmk_off_timeout	時間	60s	**stonith-timeout** の代わりにオフの動作に対して使用する代替タイムアウトです。オフに通常より長い時間を要するデバイスもあれば通常より短い時間でオフするデバイスもあります。オフの動作に対してデバイス固有の代替タイムアウトを指定する場合に使用します。
pcmk_off_retries	整数	2	タイムアウト期間内で off コマンドを再試行させる最大回数です。複数接続に対応していないデバイスがあります。別のタスクでビジー状態になるとデバイスが動作に失敗する場合があるため、Pacemaker は残り時間内で動作を自動的に再試行させます。Pacemaker によるオフ動作の再試行回数を変更する場合に使用します。

フィールド	タイプ	デフォルト	説明
pcmk_list_action	文字列	list	list の代わりに実行する代替のコマンドです。標準的なコマンドに対応していないデバイスや別のコマンドを提供しているデバイスがあります。このような場合、このパラメーターを使って list の動作を実行するデバイス固有の代替コマンドを指定します。
pcmk_list_timeout	時間	60s	stonith-timeout の代わりに list の動作に対して使用する代替タイムアウトです。list の完了に通常より長い時間を要するデバイスもあれば通常より短い時間で list するデバイスもあります。list の動作に対してデバイス固有の代替タイムアウトを指定する場合に使用します。
pcmk_list_retries	整数	2	タイムアウト期間内で list コマンドを再試行させる最大回数です。複数接続に対応していないデバイスがあります。別のタスクでビジー状態になるとデバイスが動作に失敗する場合があるため、Pacemaker は残り時間内で動作を自動的に再試行させます。Pacemaker による list 動作の再試行回数を変更する場合に使用します。
pcmk_monitor_action	文字列	monitor	monitor の代わりに実行する代替のコマンドです。標準的なコマンドに対応していないデバイスや別のコマンドを提供しているデバイスがあります。このような場合、このパラメーターを使ってモニタリングの動作を実行するデバイス固有の代替コマンドを指定します。
pcmk_monitor_timeout	時間	60s	stonith-timeout の代わりにモニターの動作に対して使用する代替タイムアウトです。モニターに通常より長い時間を要するデバイスもあれば通常より短い時間でオフするデバイスもあります。モニターの動作に対してデバイス固有の代替タイムアウトを指定する場合に使用します。
pcmk_monitor_retries	整数	2	タイムアウト期間内で monitor コマンドを再試行させる最大回数です。複数接続に対応していないデバイスがあります。別のタスクでビジー状態になるとデバイスが動作に失敗する場合があるため、Pacemaker は残り時間内で動作を自動的に再試行させます。Pacemaker によるモニター動作の再試行回数を変更する場合に使用します。

フィールド	タイプ	デフォルト	説明
pcmk_status_action	文字列	status	**status** の代わりに実行する代替のコマンドです。標準的なコマンドに対応していないデバイスや別のコマンドを提供しているデバイスがあります。このような場合、このパラメーターを使って status の動作を実行するデバイス固有の代替コマンドを指定します。
pcmk_status_timeout	時間	60s	**stonith-timeout** の代わりに status の動作に対して使用する代替タイムアウトです。status に通常より長い時間を要するデバイスもあれば通常より短い時間でオフするデバイスもあります。status の動作に対してデバイス固有の代替タイムアウトを指定する場合に使用します。
pcmk_status_retries	整数	2	タイムアウト期間内で status コマンドを再試行させる最大回数です。複数接続に対応していないデバイスがあります。別のタスクでビジー状態になるとデバイスが動作に失敗する場合があるため、Pacemaker は残り時間内で動作を自動的に再試行させます。Pacemaker による status 動作の再試行回数を変更する場合に使用します。

4.10. フェンスのレベルを設定する

Pacemaker はフェンストポロジーと呼ばれる機能で複数のフェンスデバイスを搭載したノードの排他処理に対応します。トポロジーを実装する場合は通常通りにそれぞれのフェンスデバイスを作成した後、設定内のフェンストポロジーセクションでフェンスレベルの指定を行います。

- 各レベルは 1 から昇順で試行されていきます。
- 任意のフェンスデバイスに障害が発生すると、現行レベルの排他処理は終了します。同レベルのデバイスには試行されず、次のレベルが試行されます。
- すべてのデバイスの排他処理が正常に完了するとそのレベルが継承され他のレベルは試行されません。
- 任意のレベルで成功するまたはすべてのレベルが試行される (失敗する) と動作は終了します。

ノードにフェンスレベルを追加する場合に次のコマンドを使用します。複数デバイスの指定は stonith ID を使ってコンマで区切ります。指定されたデバイスが指定されたレベルで試行されます。

```
pcs stonith level add level node devices
```

次のコマンドを使用すると現在設定されている全フェンスレベルが表示されます。

```
pcs stonith level
```

次の例では、**rh7-2** と言うノードに **my_ilo** という ilo デバイスと **my_apc** という apc デバイスの 2 種類

のフェンスデバイスが設定されています。**my_ilo** デバイスに障害が発生しノードの排他処理が行えなくなった場合は **my_apc** デバイスが使用されるようフェンスレベルが設定されています。また、次の例ではフェンスレベルの設定をした後に発行した **pcs stonith level** コマンドの出力も示しています。

```
# pcs stonith level add 1 rh7-2 my_ilo
# pcs stonith level add 2 rh7-2 my_apc
# pcs stonith level
 Node: rh7-2
  Level 1 - my_ilo
  Level 2 - my_apc
```

次のコマンドは指定ノードとデバイスから指定のフェンスレベルを削除します。ノードやデバイスを指定していないと全ノードから指定のフェンスレベルが削除されます。

```
pcs stonith level remove level [node_id] [stonith_id] ... [stonith_id]
```

次のコマンドを使用すると指定ノードや stonith id のフェンスレベルが消去されます。ノードや stonith id を指定しないと全フェンスレベルが消去されます。

```
pcs stonith level clear [node|stonith_id(s)]
```

複数の stonith IDを指定する場合はコンマで区切って指定します。空白は入れないでください。例を示します。

```
# pcs stonith level clear dev_a,dev_b
```

次のコマンドで定義したフェンスデバイスとノードが実際に存在しているか確認します。

```
pcs stonith level verify
```

第5章 クラスターリソースの設定

本章ではクラスター内にリソースを設定する方法について説明していきます。

5.1. リソースの作成

次のコマンドを使用してクラスターリソースを作成します。

```
pcs resource create resource_id standard:provider:type|type [resource options]
```

例えば、次のコマンドでは VirtualIP という名前で標準が **ocf**、プロバイダーが **heartbeat**、タイプが **IPaddr2** のリソースを作成しています。リソースのフローティングアドレスは 192.168.0.120、リソースが実行しているかどうかの確認が 30 秒毎に行われます。

```
# pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.120 cidr_netmask=24 op
monitor interval=30s
```

standard と *provider* のフィールドを省略して次のようにすることもできます。標準とプロバイダーはそれぞれ **ocf** と **heartbeat** にデフォルト設定されます。

```
# pcs resource create VirtualIP IPaddr2 ip=192.168.0.120 cidr_netmask=24 op monitor
interval=30s
```

設定したリソースを削除する場合は次のコマンドを使用します。

```
pcs resource delete resource_id
```

例えば、次のコマンドでは **VirtualIP** というリソース ID の既存リソースを削除しています。

```
# pcs resource delete VirtualIP
```

- **pcs resource create** コマンドのフィールド、*resource_id*、*standard*、*provider*、*type* については 「リソースのプロパティ」 を参照してください。
- リソースごとにパラメーターを指定する方法については 「リソース固有のパラメーター」 を参照してください。
- リソースの動作をクラスターが決定する場合に使用するリソースのメタオプションを定義する方法については 「リソースのメタオプション」 を参照してください。
- リソースで行う動作を定義する方法については 「リソースの動作」 を参照してください。

5.2. リソースのプロパティ

リソースに定義するプロパティを使ってリソースに使用するスクリプト、スクリプトの格納先、準拠すべき標準をクラスターに指示します。表5.1「リソースのプロパティ」 に詳細を示します。

表5.1 リソースのプロパティ

フィールド	説明
resource_id	リソースの名前

フィールド	説明
standard	スクリプトが準拠する標準、使用できる値: **ocf**、**service**、**upstart**、**systemd**、**lsb**、**stonith**
type	使用するリソースエージェントの名前、例えば **IPaddr**、**Filesystem** など
provider	OCF 仕様により複数のベンダーで同じ ResourceAgent が供給可能、Red Hat で配信するエージェントのほとんどはプロバイダーに **heartbeat** を使用

リソースのプロパティ表示に使用するコマンドは 表5.2「リソースプロパティを表示させるコマンド」を参照してください。

表5.2 リソースプロパティを表示させるコマンド

pcs 表示コマンド	出力
pcs resource list	利用できる全リソースの一覧を表示
pcs resource standard	利用できるリソースエージェントの標準を表示
pcs resource providers	利用できるリソースエージェントのプロバイダーを表示
pcs resource list ***string***	利用できるリソースを指定文字列でフィルターした一覧を表示、標準名、プロバイダー名、タイプ名などでフィルターした一覧を表示させることが可能

5.3. リソース固有のパラメーター

次のコマンドを使用するとリソースごとに設定できるパラメーターを表示させることができます。

```
# pcs resource describe standard:provider:type|type
```

例えば、次のコマンドでは **LVM** タイプのリソースに設定できるパラメーターを表示しています。

```
# pcs resource describe LVM
Resource options for: LVM
  volgrpname (required): The name of volume group.
  exclusive: If set, the volume group will be activated exclusively.
  partial_activation: If set, the volume group will be activated even
  only partial of the physicalvolumes available. It helps to set to
  true, when you are using mirroring logical volumes.
```

5.4. リソースのメタオプション

リソース固有のパラメーターの他にも追加のリソースオプションを設定することができます。オプションはリソースの動作を決定するためクラスターによって使用されます。表5.3「リソースのメタオプション」 に詳細を示します。

表5.3 リソースのメタオプション

フィールド	デフォルト	説明
priority	**0**	すべてのリソースを実行できない場合は優先度の高いリソースの実行を維持するため低い方のリソースを停止します
target-role	**Started**	維持するリソースの状態、使用できる値: * **Stopped** - リソースの強制停止 * **Started** - リソースの起動を許可 (多状態リソースの場合マスターには昇格されません) * **Master** - リソースの起動を許可、また適切であれば昇格されます
is-managed	**true**	クラスターによるリソースの起動と停止を許可または許可しない、使用できる値: **true**、**false**
resource-stickiness	0	リソースをそのノードに留まらせる優先度の値
requires	Calculated	リソースの起動を許可する条件 以下の条件の場合を除き **fencing** にデフォルト設定、可能な値: * **nothing** - クラスターによるリソースの起動を常に許可 * **quorum** - 設定されているノードの過半数がアクティブな場合にのみクラスターによるこのリソースの起動を許可、**stonith-enabled** が **false** に設定されている、またはリソースの **standard** が **stonith** に設定されている場合はこのオプションがデフォルト * **fencing** -設定されているノードの過半数がアクティブで **且つ** 障害が発生しているノードや不明なノードの電源がすべてオフになっている場合にのみ、クラスターによるこのリソースの起動を許可 * **unfencing** - 設定されているノードの過半数がアクティブで **且つ** 障害が発生しているノードや不明なノードの電源がすべてオフになっている場合にのみ **アンフェンス** が行われたノードに **限定** してこのリソースの起動を許可、フェンスデバイスに **provides=unfencing stonith** メタオプションが設定されていた場合はこれがデフォルト値 (**provides=unfencing stonith** メタオプションの詳細は 「ストレージベースのフェンスデバイスをアンフェンスを使って設定する」 を参照)
migration-threshold	**INFINITY** (無効)	任意のノードでの指定リソースの失敗回数、この回数を越えるとそのノードでの指定リソースのホストは不適格とする印が付けられる (**migration-threshold** オプションの設定は 「障害発生のためリソースを移動する」 を参照)

フィールド	デフォルト	説明
failure-timeout	**0** (無効)	**migration-threshold** オプションと併用、障害は発生していなかったとして動作するまでに待機させる秒数、リソースの実行に失敗したノードで再度そのリソースの実行を許可する可能性がある (**failure-timeout** オプションの設定は 「障害発生のためリソースを移動する」 を参照)
multiple-active	**stop_start**	リソースが複数のノードで実行していることが検出された場合にクラスターに行わせる動作、使用できる値: * **block** - リソースに unmanaged のマークを付ける * **stop_only** - 実行しているインスタンスをすべて停止する (これ以上の動作は行わない) * **stop_start** - 実行中のインスタンスをすべて停止してから一ヶ所でのみ起動する

リソースオプションのデフォルト値を変更する場合は次のコマンドを使用します。

```
pcs resource defaults options
```

例えば、次のコマンドでは **resource-stickiness** のデフォルト値を 100 にリセットしています。

```
# pcs resource defaults resource-stickiness=100
```

pcs resource defaults の *options* パラメーターを省略すると現在設定しているリソースオプションのデフォルト値の一覧を表示します。次の例では **resource-stickiness** のデフォルト値を 100 にリセットした後の出力を示しています。

```
# pcs resource defaults
resource-stickiness:100
```

リソースのメタオプションのデフォルト値をリセットしていたかいないかによって、リソース作成の際に特定リソースのリソースオプションをデフォルト以外の値に設定することができます。リソースのメタオプションの値を指定する時に使用する **pcs resource create** コマンドの形式を以下に示します。

```
pcs resource create resource_id standard:provider:type|type [resource options] [meta
meta_options...]
```

例えば、次のコマンドでは **resource-stickiness** の値を 50 に設定したリソースを作成しています。

```
# pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.120 cidr_netmask=24
meta resource-stickiness=50
```

また、次のコマンドを使用すると既存のリソース、グループ、クローン作成したリソース、マスターリソースなどのリソースメタオプションの値を作成することもできます。

```
pcs resource meta resource_id | group_id | clone_id | master_id  meta_options
```

次の例では、既存の **dummy_resource** と言うリソースに **failure-timeout** メタオプションの値を 20 秒に設定しているコマンドの例を示します。これにより 20 秒でリソースが同じノード上で再起動試行できるようになります。

```
# pcs resource meta dummy_resource failure-timeout=20s
```

上記のコマンドを実行した後、**failure-timeout=20s** が設定されているか確認するためリソースの値を表示させることができます。

```
# pcs resource show dummy_resource
 Resource: dummy_resource (class=ocf provider=heartbeat type=Dummy)
  Meta Attrs: failure-timeout=20s
  Operations: start interval=0s timeout=20 (dummy_resource-start-timeout-20)
              stop interval=0s timeout=20 (dummy_resource-stop-timeout-20)
              monitor interval=10 timeout=20 (dummy_resource-monitor-interval-10)
```

リソースの clone メタオプションについては 「リソースのクローン」 を参照してください。リソースの master メタオプションについては 「多状態のリソース: 複数モードのリソース」 を参照してください。

5.5. リソースの動作

リソースに健全性を維持させるためリソースの定義にモニタリングの動作を追加することができます。モニタリングの動作を指定しないと、**pcs** コマンドはデフォルトでモニタリングの動作を作成します。モニタリングの間隔はリソースエージェントで確定されます。リソースエージェントでデフォルトのモニタリング間隔が提供されない場合は pcs コマンドにより 60 秒間隔のモニタリング動作が作成されます。

表5.4「動作のプロパティ」 にリソースのモニタリング動作のプロパティを示します。

表5.4 動作のプロパティ

フィールド	説明
id	動作に固有となる名前、動作を設定するとシステムによって割り当てられる
name	実行する動作、一般的な値:**monitor**、**start**、**stop**
interval	動作実行の頻度 (秒単位)、デフォルト値:**0** (なし)
timeout	動作が失敗したことを示すまでの待機時間 (起動、停止および起動時の反復性のないモニタリング動作の実行などに時間がかかるリソースがシステムに含まれていることが確認され、起動動作に失敗したことを示すまでにシステムが許容している以上に時間を要する場合にはこの値をデフォルト値 20 または "op defaults" に指定されている **timeout** 値から増やします)

フィールド	説明
on-fail	この動作が失敗した場合に実行する動作、使用できる値: * **ignore** - リソースが失敗していなかったように振る舞う * **block** - リソースでこれ以上、一切の動作を行わない * **stop** - リソースを停止して別の場所で起動しない * **restart** - リソースを停止してから再起動する (おそらく別の場所) * **fence** - 失敗したリソースがあるノードを STONITH する * **standby** - 失敗したリソースがあるノード上の **すべてのリソース** を移動させる STONITH が有効で**block** に設定されていると **stop** 動作のデフォルトは **fence** になります。これ以外は **restart** にデフォルト設定されます。
enabled	**false** に設定するとその動作はないものとして処理される、使用できる値: **true**、**false**

次のコマンドでリソースを作成するとモニタリングの動作を設定することができます。

```
pcs resource create resource_id standard:provider:type|type [resource_options] [op
operation_action operation_options [operation_type operation_options]...]
```

例えば、次のコマンドはモニタリング動作付きの **IPaddr2** リソースを作成します。新しいリソースには **VirtualIP** という名前が付けられ、**eth2** で IP アドレス 192.168.0.99、ネットマスク 24 になります。モニタリング動作は 30 秒毎に実施されます。

```
# pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.99 cidr_netmask=24
nic=eth2 op monitor interval=30s
```

```
# pcs resource create my_address IPaddr2 ip=10.20.30.40 cidr_netmask=24 op monitor
```

また、次のコマンドで既存のリソースにモニタリング動作を追加することもできます。

```
pcs resource op add resource_id operation_action [operation_properties]
```

設定されているリソースの動作を削除する場合は次のコマンドを使用します。

```
pcs resource op remove resource_id operation_name operation_properties
```

注記

既存の動作を確実に削除するため正確な動作プロパティを指定してください。

モニタリングオプションの値を変更する場合は既存の動作を削除してから新しい動作を追加します。例えば、次のコマンドでは **VirtualIP** を作成しています。

```
# pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.99 cidr_netmask=24
nic=eth2
```

デフォルトでは次の動作を作成します。

```
Operations: start interval=0s timeout=20s (VirtualIP-start-timeout-20s)
            stop interval=0s timeout=20s (VirtualIP-stop-timeout-20s)
            monitor interval=10s timeout=20s (VirtualIP-monitor-interval-10s)
```

stop の timeout 動作を変更する場合は次のコマンドを実行します。

```
# pcs resource op remove VirtualIP stop interval=0s timeout=20s
# pcs resource op add VirtualIP stop interval=0s timeout=40s

# pcs resource show VirtualIP
 Resource: VirtualIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: ip=192.168.0.99 cidr_netmask=24 nic=eth2
  Operations: start interval=0s timeout=20s (VirtualIP-start-timeout-20s)
              monitor interval=10s timeout=20s (VirtualIP-monitor-interval-10s)
              stop interval=0s timeout=40s (VirtualIP-name-stop-interval-0s-timeout-40s)
```

モニタリング動作にグローバルのデフォルト値を設定する場合は次のコマンドを使用します。

```
pcs resource op defaults [options]
```

例えば、次のコマンドはすべてのモニタリング動作にグローバルデフォルトの **timeout** 値を 240s で設定しています。

```
# pcs resource op defaults timeout=240s
```

現在設定されているモニタリング動作のデフォルト値を表示させる場合はオプションを付けずに **pcs resource op defaults** コマンドを実行します。

例えば、次のコマンドは **timeout** が 240s で設定されているクラスターのモニタリング動作のデフォルト値を表示しています。

```
# pcs resource op defaults
timeout: 240s
```

5.6. 設定されているリソースの表示

設定されているリソースの全一覧を表示する場合は次のコマンドを使用します。

```
pcs resource show
```

例えば、**VirtualIP** と言う名前のリソースと **WebSite** という名前のリソースでシステムを設定していた場合、**pcs resource show** コマンドを実行すると次のような出力が得られます。

```
# pcs resource show
 VirtualIP (ocf::heartbeat:IPaddr2): Started
 WebSite (ocf::heartbeat:apache): Started
```

設定されているリソース、そのリソースに設定されているパラメーターの一覧を表示する場合は、次のように **pcs resource show** コマンドの **--full** オプションを使用します。

```
# pcs resource show --full
 Resource: VirtualIP (type=IPaddr2 class=ocf provider=heartbeat)
  Attributes: ip=192.168.0.120 cidr_netmask=24
  Operations: monitor interval=30s
 Resource: WebSite (type=apache class=ocf provider=heartbeat)
  Attributes: statusurl=http://localhost/server-status
configfile=/etc/httpd/conf/httpd.conf
  Operations: monitor interval=1min
```

設定されているリソースのパラメーターを表示する場合は次のコマンドを使用します。

```
pcs resource show resource_id
```

例えば、次のコマンドは現在設定されているリソース **VirtualIP** のパラメーターを表示しています。

```
# pcs resource show VirtualIP
 Resource: VirtualIP (type=IPaddr2 class=ocf provider=heartbeat)
  Attributes: ip=192.168.0.120 cidr_netmask=24
  Operations: monitor interval=30s
```

5.7. リソースパラメーターの変更

設定されているリソースのパラメーターを変更する場合は次のコマンドを使用します。

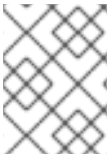
```
pcs resource update resource_id [resource_options]
```

設定されているリソース **VirtualIP** のパラメーターの値を示すコマンドと初期値を表示している出力、**ip** パラメーターの値を変更するコマンド、変更されたパラメーター値を表示している出力を以下に示します。

```
# pcs resource show VirtualIP
 Resource: VirtualIP (type=IPaddr2 class=ocf provider=heartbeat)
  Attributes: ip=192.168.0.120 cidr_netmask=24
  Operations: monitor interval=30s
# pcs resource update VirtualIP ip=192.169.0.120
# pcs resource show VirtualIP
 Resource: VirtualIP (type=IPaddr2 class=ocf provider=heartbeat)
  Attributes: ip=192.169.0.120 cidr_netmask=24
  Operations: monitor interval=30s
```

5.8. 複数のモニタリング動作

リソースエージェントが対応する範囲で一つのリソースに複数のモニタリング動作を設定することができます。これにより毎分、表面的なヘルスチェックを行ったり、徐々に頻度を上げてより正確なチェックを行うこともできます。



注記

複数のモニタリング動作を設定する場合は 2 種類の動作が同じ間隔で実行されないよう注意してください。

リソースに異なるレベルで徹底的なヘルスチェックに対応する追加モニタリング動作を設定するには **OCF_CHECK_LEVEL=*n*** オプションを追加します。

例えば、 以下のように **IPaddr2** リソースを設定するとデフォルトでは 10 秒間隔でタイムアウト値が 20 秒のモニタリング動作が作成されます。

```
# pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.99 cidr_netmask=24
nic=eth2
```

仮想 IP で深さ 10 の異なるチェックに対応する場合は、次のコマンドを発行すると Packemaker で 10 秒間隔の通常仮想 IP チェックの他に 60 秒間隔の高度なモニタリングチェックが実行されるようになります。(説明されているように 10 秒間隔の追加モニタリング動作は設定しないでください。)

```
# pcs resource op add VirtualIP monitor interval=60s OCF_CHECK_LEVEL=10
```

5.9. クラスターリソースの有効化と無効化

次のコマンドは **resource_id** で指定されているリソースを有効にします。

```
pcs resource enable resource_id
```

次のコマンドは **resource_id** で指定されているリソース無効にします。

```
pcs resource disable resource_id
```

5.10. クラスターリソースのクリーンアップ

リソースに障害が発生するとクラスターの状態を表示する際に障害発生を告げるメッセージが表示されます。障害が解決したら、**pcs resource cleanup** コマンドで障害の状態を消去します。次のコマンドはリソースの状態と障害発生数をリセット、リソースの動作履歴の記録を消去して現在の状態を再検出するよう指示しています。

次のコマンドは **resource_id** で指定されているリソースのクリーンアップを行います。

```
pcs resource cleanup resource_id
```

第6章 リソースの制約

リソースの制約を設定することでクラスター内のそのリソースの動作を決めることができます。設定できる制約は以下のカテゴリーになります。

- **location** 制約 — 場所の制約はリソースを実行できるノードを決めます。場所の制約については「場所の制約」 で説明しています。
- **order** 制約 — 順序の制約はリソースが実行される順序を決めます。順序の制約については 「順序の制約」 で説明しています。
- **colocation** 制約 — コロケーションの制約は他のリソースと相対的となるリソースの配置先を決めます。コロケーションの制約については 「リソースのコロケーション」 で説明しています。

複数リソース一式を一緒に配置、それらを順番に起動させ、また逆順で停止させるため複数の制約を設定する場合、その簡易な方法として Pacemaker ではリソースグループという概念に対応しています。リソースグループについては 「リソースグループ」 を参照してください。

6.1. 場所の制約

場所の制約ではリソースを実行させるノードを指定します。場所の制約を設定することで特定のノードで優先してリソースを実行する、または特定のノードでのリソースの実行を避けるなどの指定を行うことができます。

表6.1「場所の制約オプション」 では場所の制約を設定する場合のオプションについて簡単に示します。

表6.1 場所の制約オプション

フィールド	説明
id	制約に付けられる固有の名前、**pcs** で場所の制約を設定するとシステムによって付けられる
rsc	リソース名
node	ノード名
score	優先度を示す値、任意のノードでのリソースの実行を優先または避ける **INFINITY** の値は "すべき" から "しなければならない" に変化、 **INFINITY** がリソースの場所制約のデフォルト score 値

次のコマンドはリソースが指定ノードで優先して実行される場所の制約を作成します。

```
pcs constraint location rsc prefers node[=score] ...
```

次のコマンドはリソースが指定ノードを避けて実行される場所の制約を作成します。

```
pcs constraint location rsc avoids node[=score] ...
```

リソースの実行を許可するノード指定には上記以外にも 2 種類の方法があります。

- オプトインクラスター — クラスターを設定し、デフォルトではいずれのノードでもリソース実行を許可せず、特定のリソース用に選択的に許可ノードを有効にします。オプトインクラスターの設定方法は 「「オプトイン」のクラスターを設定する」 で説明しています。
- オプトアウトクラスター — クラスターを設定し、デフォルトでは全ノードでリソース実行を許可してから、特定ノードでの実行を許可しない場所の制約を作成します。オプトアウトクラスターの設定方法は 「「オプトアウト」のクラスターを設定する」 で説明しています。

オプトインまたはオプトアウトのクラスターを設定するかどうかはユーザーの嗜好やクラスターの構成により異なるところです。ほとんどのリソースをほとんどのノードで実行させて構わない場合はオプトアウトのクラスター設定を行うとシンプルな設定になるでしょう。一方、ほとんどのリソースの実行を限定的な複数ノードに限るような場合にはオプトインのクラスター設定を行うとシンプルな設定になります。

6.1.1. 「オプトイン」のクラスターを設定する

オプトインクラスターを作成する場合はクラスタープロパティ **symmetric-cluster** を **false** に設定してデフォルトではリソースの実行をいずれのノードでも許可しないようにします。

```
# pcs property set symmetric-cluster=false
```

リソースごとにノードを有効にします。次のコマンドは場所の制約を設定するため、**Webserver** リソースは **example-1** ノードでの実行を優先させ、**Database** リソースは **example-2** ノードでの実行を優先させるようになります。また、いずれのリソースも優先ノードに障害が発生した場合は **example-3** ノードにフェールオーバーすることができます。

```
# pcs constraint location Webserver prefers example-1=200
# pcs constraint location Webserver prefers example-3=0
# pcs constraint location Database prefers example-2=200
# pcs constraint location Database prefers example-3=0
```

6.1.2. 「オプトアウト」のクラスターを設定する

オプトアウトクラスターを作成する場合はクラスタープロパティ **symmetric-cluster** を **true** に設定しデフォルトではリソースの実行をすべてのノードに許可します。

```
# pcs property set symmetric-cluster=true
```

次のコマンドを実行すると 「「オプトイン」のクラスターを設定する」 の例と同じ設定になります。全ノードの score は暗黙で 0 になるため、優先ノードに障害が発生した場合はいずれのリソースも **example-3** ノードにフェールオーバーすることができます。

```
# pcs constraint location Webserver prefers example-1=200
# pcs constraint location Webserver avoids example-2=INFINITY
# pcs constraint location Database avoids example-1=INFINITY
# pcs constraint location Database prefers example-2=200
```

上記コマンドでは score に INFINITY を指定する必要はありません。INFINITY が score のデフォルト値になります。

6.2. 順序の制約

順序の制約はリソースの実行順序を指定します。順序の制約を設定することでリソースの起動と停止の順序を指定することができます。

次のコマンドを使って順序の制約を設定します。

```
pcs constraint order [action] resource_id then [action] resource_id [options]
```

表6.2「順序の制約のプロパティ」 では順序の制約を設定する場合のプロパティとオプションについて簡単に示します。

表6.2 順序の制約のプロパティ

フィールド	説明
resource_id	動作を行うリソースの名前
action	リソースで行う動作、*action* プロパティで使用できる値: * **start** - リソースを起動する * **stop** - リソースを停止する * **promote** - スレーブリソースからマスターリソースにリソースの昇格を行う * **demote** - マスターリソースからスレーブリソースにリソースの降格を行う 動作を指定しないとデフォルトの動作 **start** が設定されます。マスターリソースとスレーブリソースについての詳細は 「多状態のリソース: 複数モードのリソース」 を参照してください。
kind オプション	制約の実施方法、**kind** オプションで使用できる値: * **Optional** - いずれのリソースも起動中や停止中の場合にのみ適用 (「勧告的な順序付け」 を参照) * **Mandatory** - 常に実施 (デフォルト値)、1 番目に指定したリソースが停止しているまたは起動できない場合、2 番目に指定されているリソースを停止しなければなりません (「強制的な順序付け」 を参照) * **Serialize** - リソースセットに対して 2 種類の動作、停止と起動が同時に発生しないようにする
symmetrical オプション	デフォルトの true に設定されていると逆順でリソースの停止を行ないます。デフォルト値: **true**

6.2.1. 強制的な順序付け

mandatory 制約では 1 番目に指定しているリソースが実行されない限り 2 番目に指定しているリソースは実行できません。 これが **kind** オプションのデフォルトです。デフォルト値のままにしておくと 1 番目に指定しているリソースの状態が変化した場合、2 番目に指定したリソー スが必ず反応するようになります。

- 1 番目に指定している実行中のリソースを停止すると 2 番目に指定しているリソースも停止されます (実行していれば)。

- 1 番目に指定しているリソースが実行されていない状態でまた起動できない場合には 2 番目に指定しているリソースが停止されます (実行していれば)。
- 2 番目に指定しているリソースの実行中に 1 番目に指定しているリソースが再起動されると、2 番目に指定しているリソースが停止され再起動されます。

6.2.2. 勧告的な順序付け

kind=Optional のオプションを順序の制約で指定すると、その制約はオプションとみなされ両方のリソースが停止中または起動中の場合にのみ適用されます。1 番目に指定しているリソースの状態が変化しても 2 番目に指定しているリソースには影響しません。

次のコマンドは **VirtualIP** リソースと **dummy_resource** リソースに勧告的な順序付けの制約を設定しています。

```
# pcs constraint VirtualIP then dummy_resource kind=Optional
```

6.2.3. 順序付けされたリソースセット

一般的な状況として管理者は複数リソースの順序付けで連鎖して動作するリソースチェーンを作成します。例えば、リソース A が起動してからリソース B が起動、リソース B が起動してからリソース C が起動するというような連鎖です。このような連鎖して動作するチェーンは次のコマンドで設定します。複数のリソースが指定した順序で起動するようになります。

```
pcs constraint order set resource1 resource2 [resourceN]... [options] [set resource1
resource2 ...]
```

D1、**D2**、**D3** という 3 つのリソースがあると仮定した場合、次のコマンドはこの 3 つのリソースを順序付けされたひとつのリソースセットとして設定します。

```
# pcs constraint order set D1 D2 D3
```

6.2.4. 順序の制約からリソースを削除する

次のコマンドを使用するとすべての順序の制約からリソースを削除します。

```
pcs constraint order remove resource1 [resourceN]...
```

6.3. リソースのコロケーション

任意のリソースの場所を別のリソースの場所に依存するよう定義するのがコロケーションの制約です。

2 つのリソース間にコロケーションの制約を作成する場合は重要な副作用がある点に注意してください。ノードにリソースを割り当てる順序に影響します。つまり、リソース B の場所がわからないとリソース B に相対的となるようリソース A を配置することはできません。このため、コロケーションの制約を作成する場合は、リソース A をリソース B に対してコロケートするのか、リソース B をリソース A に対してコロケートするのかが重要となります。

また、コロケーションの制約を作成する際に注意しておきたい事項がもう一つあります。リソース A をリソース B に対してコロケートすると仮定した場合、クラスターはリソース B に選択するノードを決定する際、リソース A の優先傾向も考慮に入れます。

次のコマンドはコロケーションの制約を作成します。

```
pcs constraint colocation add [master|slave] source_resource with [master|slave]
target_resource [score] [options]
```

マスターリソース、スレーブリソースの詳細は 「多状態のリソース: 複数モードのリソース」 を参照してください。

表6.3「コロケーション制約のプロパティ」 にコロケーション制約設定用のプロパティおよびオプションを示します。

表6.3 コロケーション制約のプロパティ

フィールド	説明
source_resource	コロケーションソース、制約の条件を満たせない場合はこのリソースの実行を全く許可しないと判断されることがあります。
target_resource	コロケーションターゲット、このリソースの実行先が最初に決定されてからソースリソースの実行先が決定されます。
score	正の値にするとリソースを同じノードで実行する、負の値にするとリソースを同じノードで実行しない設定になります。+ **INFINITY** がデフォルト値になります。つまり *source_resource* を *target_resource* と同じノードで実行させるということです。- **INFINITY** の値にすると *source_resource* を *target_resource* と同じノードで実行させないということになります。

6.3.1. 強制的な配置

制約スコアが **+INFINITY** または **-INFINITY** の場合は常に強制的な配置が発生します。制約条件が満たされないと *source_resource* の実行が許可されません。**score=INFINITY** の場合、*target_resource* がアクティブではないケースが含まれます。

myresource1 を常に **myresource2** と同じマシンで実行する場合は次のような制約を追加します。

```
# pcs constraint colocation add myresource1 with myresource2 score=INFINITY
```

INFINITY を使用しているため **myresource2** がクラスターのいずれのノードでも実行できない場合には (理由の如何に関わらず) **myresource1** の実行は許可されません。

また、逆の設定、つまり **myresource1** が **myresource2** と同じマシンでは実行できないようクラスターを設定することもできます。この場合は **score=-INFINITY** を使用します。

```
# pcs constraint colocation add myresource1 myresource2 with score=-INFINITY
```

-INFINITY を指定することで制約が結合しています。このため、実行できる場所として残っているノードで **myresource2** がすでに実行されている場合には **myresource1** はいずれのノードでも実行できなくなります。

6.3.2. 勧告的な配置

強制的な配置が必ず行われる必須の配置とすると、勧告的な配置はできれば優先して行わせる配置になります。スコアが **-INFINITY** より大きく **INFINITY** より小さい値の制約の場合、希望の適用を試行しますが適用によりクラスターリソースのいくつかを停止することになる場合は無視されます。勧告的なコロケーション制約は設定の他の構成要素と合わせて強制的な制約のように動作させることができます。

6.3.3. 複数のリソースをコロケートする

複数のリソースにコロケーション制約を作成する場合は次のコマンドを使用します。 **sequential** オプションを **true** または **false** に設定することでコロケートするリソース群に順序付けするかどうかを指定できます。

```
colocation set resource1 resource2 [resourceN]... [setoptions name=value] ... [set
resourceX resourceY ...] [setoptions name=value...]
```

role オプションを **master** または **slave** に設定することができます。多状態のリソースについては 「多状態のリソース: 複数モードのリソース」 を参照してください。

6.3.4. コロケーション制約を削除する

コロケーション制約を削除する場合はコマンドに *source_resource* を付けて使用します。

```
pcs constraint colocation remove source_resource target_resource
```

6.4. 制約の表示

設定した制約を表示させるコマンドがいくつかあります。

次のコマンドは現在の場所、順序、ロケーションの制約を表示します。

```
pcs constraint list|show
```

次のコマンドは現在の場所の制約を表示します。

- **resources** を指定すると場所制約がリソースごとに表示されます。デフォルトの動作です。
- **nodes** を指定すると場所制約がノードごとに表示されます。
- 特定のリソースまたはノードを指定するとそのリソースまたはノードの情報のみが表示されます。

```
pcs constraint location [show resources|nodes [specific nodes|resources]] [--full]
```

次のコマンドは現在の全順序制約を表示します。**--full** オプションを指定すると制約の内部 ID を表示します。

```
pcs constraint order show [--full]
```

次のコマンドは現在の全コロケーション制約を表示します。**--full** オプションを指定すると制約の内部 ID を表示します。

```
pcs constraint colocation show [--full]
```

次のコマンドは特定リソースを参照する制約を表示します。

```
pcs constraint ref resource ...
```

6.5. リソースグループ

クラスターのもっとも一般的な構成要素の一つが複数リソースのセットです。リソースセットは一緒に配置し、順番に起動、その逆順で停止する必要があります。この設定を簡略化するために Pacemaker ではグループという概念に対応しています。

次のコマンドでリソースグループを作成、グループに含ませるリソースを指定します。グループが存在していない場合はグループが作成され、グループが存在している場合はそのグループにリソースを追加します。リソースはコマンドで指定した順序で起動し、起動とは逆順で停止します。

```
pcs resource group add group_name resource_id...
```

また、次のコマンドを使用すると新規リソースの作成時にそのリソースを既存のグループに追加することもできます。作成したリソースは *group_name* という名前のグループに追加されます。

```
pcs resource create resource_id standard:provider:type|type [resource_options] [op
operation_action operation_options] --group group_name
```

グループからのリソースの削除は次のコマンドで行います。グループにリソースがない場合はグループ自体が削除されます。

```
pcs resource group remove group_name resource_id...
```

次のコマンドは現在、設定されているリソースグループを表示します。

```
pcs resource group list
```

次の例では **shortcut** という名前のリソースグループを作成します。このリソースグループには既存リソースの **IPaddr** と **Email** が含まれます。

```
# pcs resource group add shortcut IPaddr Email
```

グループに含ませることができるリソース数に制限はありません。グループの基本的なプロパティを以下に示します。

- リソースの起動はコマンドで指定した順序で行われます (この例の場合は **IPaddr** が起動されてから **Email** か起動)。
- リソースの停止はコマンドで指定した順序の逆順で行われます (**Email** が停止されてから **IPaddr** が停止)。

グループ内の任意のリソースがどのノードでも実行できないと、そのリソースより後に指定されているリソースの実行は許可されません。

- **IPaddr** をどのノードでも実行できない場合は **Email** も実行できません。
- ただし、**Email** を実行できない場合は **IPaddr** に影響はありません。

グループが大きくなればなるほど、当然、リソースグループ作成に関する設定作業の簡略化は重要となってきます。

6.5.1. グループオプション

リソースグループに含まれるリソースのオプション、**priority**、 **target-role**、**is-managed** はリソースグループに継承されます。リソースオプションについては 表5.3「リソースのメタオプション」 を参照してください。

6.5.2. グループの粘着性

グループの付加的なオプションとなる粘着性とはリソースをその場所に留まらせる尺度になります。グループで実行中のリソースはすべて粘着性の値を持ちグループの合計として合算されます。デフォルトの **resource-stickiness** が 100、グループには 7 リソースメンバーが属しそのうち 5 リソースメンバーが実行中だった場合、グループ全体としては現在の場所に留まる粘着性スコアが 500 ということになります。

第7章 クラスターリソースの管理

本章ではクラスターのリソース管理に使用する各種コマンドについて説明します。次のような手順が含まれます。

- 「リソースを手作業で移動する」
- 「障害発生のためリソースを移動する」
- 「クラスターのリソースを有効にする、無効にする、禁止する」
- 「モニタリングの動作を無効にする」

7.1. リソースを手作業で移動する

クラスターの設定を無視して強制的にリソースを現在の場所から移動させることができます。次のような２タイプの状況が考えられます。

- ノードのメンテナンスのためそのノードで実行中の全リソースを別のノードに移動する必要がある
- リソースを一つだけ移動する必要がある

ノードで実行中の全リソースを別のノードに移動する場合はそのノードをスタンバイモードにします。クラスターノードをスタンバイモードにする方法については 「スタンバイモード」 を参照してください。

現在実行中の任意のリソースをノードから移動させる場合は次のコマンドを使用し、ノードの **resource_id** を指定します。

```
pcs resource move resource_id
```

移動しているリソースの実行先を指定する場合は次のコマンドを使って **destination_node** を指定します。

```
pcs resource move resource_id destination_node
```

元々実行していたノードにリソースを戻す場合は次のコマンドを使用し、クラスターが通常動作を再開できるようにします。**move resource_id** コマンドの制約が削除されます。

```
pcs resource clear resource_id [node]
```

pcs resource move コマンドを実行すると指定したノードでのリソースの実行を阻止する制約が追加されることになります。制約を取り除く場合は **pcs resource clear** コマンドを実行します。このコマンドは必ずしもリソースを指定ノードに戻すわけではありません。最初にどのようにリソースを設定したかにより、その時点で実行できるノードに移動されます。リソースの制約については 6章 *リソースの制約* を参照してください。

7.2. 障害発生のためリソースを移動する

リソースの作成時、リソースに **migration-threshold** オプションをセットし、セットした回数の障害が発生するとリソースが新しいノードに移動されるよう設定することができます。このしきい値に一旦達してしまうと、このノードは障害が発生したリソースを実行できなくなります。解除には以下が必要になります。

- 管理側で **pcs resource failcount** コマンドを使ったリソース障害回数のリセットを手作業で行う
- リソースの **failure-timeout** 値に達する

デフォルトではしきい値は定義されません。

注記

リソースの **migration-threshold** を設定するのと、リソースの状態を維持しながら別の場所に移動させるリソースの移行設定とは異なります。

次の例では **dummy_resource** というリソースに移行しきい値 10 を追加しています。この場合、障害が 10 回発生するとそのリソースは新しいノードに移動されます。

```
# pcs resource meta dummy_resource migration-threshold=10
```

次のコマンドを使用するとクラスター全体にデフォルトの移行しきい値を追加することができます。

```
# pcs resource defaults migration-threshold=10
```

リソースの現在の障害回数とリミットを確認するには **pcs resource failcount** コマンドを使用します。

移行しきい値の概念には 2 種類の例外があります。リソース起動時の失敗と停止時の失敗です。起動時の失敗が発生すると障害回数が **INFINITY** に設定されるため、常にリソースの移動が直ちに行われることになります。

停止時の失敗は起動時とは若干異なり重大です。リソースの停止に失敗し STONITH が有効になっている場合、リソースを別のノードで起動できるようクラスターによるノードの排他処理が行われます。STONITH を有効にしていない場合にはクラスターに続行する手段がないため別のノードでのリソース起動は試行されません。ただし、障害タイムアウト後に停止が再度試行されます。

7.3. 接続状態が変化したためリソースの移動を行う

外部の接続が失われた場合、次の 2 ステップでクラスターがリソースを移動するよう設定します。

1. クラスターに **ping** リソースを追加します。**ping** リソースは同じ名前のシステムユーティリティーを使って一覧のマシン (DNS ホスト名または IPv4/IPv6 アドレスで指定) へのアクセスが可能かをテストし、その結果を使って **pingd** というノード属性を維持管理します。
2. 接続が失われたときに別のノードにリソースを移動させるためのリソース場所制約を設定します。

表5.1「リソースのプロパティ」 では **ping** リソースに設定できるプロパティを示します。

表7.1 ping リソースのプロパティ

フィールド	説明
dampen	さらに変化が起こった場合に備えて待機させる時間 (dampen)、ノードによって接続が失われたことを認識する時間にずれが生じた場合にクラスター内のノード間で何度も移動が行われないようにするための待機時間です

フィールド	説明
multiplier	接続している ping ノード数にこの値をかけてスコアを取得、ping ノードが複数設定されている場合に役立ちます
host_list	現在の接続状態を確認するため通信を行うマシン、解決可能な DNSホスト名、 IPv4 や IPv6 アドレスなどが値として使用できます

次のコマンド例では **www.example.com** への接続性を検証する **ping** リソースが作成されます。実際には使用しているネットワークのゲートウェイやルーターへの接続性を検証することになります。リソースがクラスターの全ノードで実行されるよう **ping** リソースはクローンとして設定します。

```
# pcs resource create ping ocf:pacemaker:ping dampen=5s multiplier=1000
host_list=www.example.com --clone
```

次の例では既存の **Webserver** と言うリソースに場所の制約ルールを設定しています。 **Webserver** リソースを現在実行しているホストが **www.example.com** に ping できない場合、**www.example.com** に ping できるホストに移動されます。

```
# pcs constraint location Webserver rule score=-INFINITY pingd lt 1 or not_defined pingd
```

7.4. クラスターのリソースを有効にする、無効にする、禁止する

「リソースを手作業で移動する」 で説明している **pcs resource move** コマンドの他にもクラスターのリソース動作制御に使用できるコマンドが各種あります。

実行中のリソースを手作業で停止し、その後そのリソースがクラスターにより再起動されないようにする場合は次のコマンドを使用します。他の設定 (制約、オプション、障害など) によってはリソースが起動したままになることがあります。**--wait** オプションを指定すると **pcs** はリソースの停止を最長で 30 秒間 (または「n」を使って秒数を指定する) 待ってから、リソースが停止した場合には 0、リソースが停止しなかった場合には 1 を返します。

```
pcs resource disable resource_id [--wait[=n]]
```

クラスターによるリソースの起動を許可する場合は次のコマンドを使用します。他の設定によってはリソースが起動したままになることがあります。**--wait** オプションを指定すると **pcs** はリソースの停止を最長で 30 秒間 (または「n」を使って秒数を指定する) 待ってから、リソースが停止した場合には 0、リソースが停止しなかった場合には 1 を返します。

```
pcs resource enable resource_id [--wait[=n]]
```

指定したノードでリソースが実行されないようにする場合は次のコマンドを使用します。ノードを指定しないと現在実行中のノードになります。

```
pcs resource ban resource_id [node]
```

pcs resource ban コマンドを実行すると指定したノードでのリソースの実行を阻止する制約が追加されることになります。制約を取り除く場合は **pcs resource clear** を実行します。このコマンドは必ずしもリソースを指定ノードに戻すわけではありません。最初にどのようにリソースを設定したかにより、その時点で実行できるノードに移動されます。リソースの制約については *6章 リソースの制約* を参照してください。

```
pcs resource clear resource_id [node]
```

指定したリソースを現在のノードで強制起動する場合は **pcs resource** コマンドの **debug-start** パラメーターを使用します。クラスターの推奨は無視され強制起動しているリソースからの出力を表示します。このコマンドは主にリソースをデバッグする際に使用されます。クラスターでのリソースの起動はほぼ毎回、Pacemaker で行われるため、直接 **pcs** コマンドを使った起動は行われません。リソースが起動しない場合、大抵はリソースが誤って設定されている、リソースが起動しないよう制約が設定されている、リソースが無効になっているのいずれかが原因です。このような場合に、このコマンドを使ってリソースの設定をテストすることができます。ただし、クラスター内でのリソースの通常起動には使用しないでください。

debug-start コマンドの形式を以下に示します。

```
pcs resource debug-start resource_id
```

7.5. モニタリングの動作を無効にする

モニタリングが繰り返し起きないようにするにはその動作を削除するのが一番簡単な方法ですが、永久削除ではなく一時的に無効にしたい場合があります。このような場合には、動作の定義に **enabled="false"** を追加します。モニタリングの動作を復活させたい場合は**enabled="true"** を設定します。

7.6. 管理リソース

リソースをアンマネージのモードに設定することができます。つまり、リソースは構成に含まれているが Pacemaker ではそのリソースの管理は行わないということです。

次のコマンドでは指定リソースをアンマネージのモードに設定します。

```
pcs resource unmanage resource1 [resource2] ...
```

次のコマンドではリソースをマネージのモードに設定します。これがデフォルトの状態です。

```
pcs resource manage resource1 [resource2] ...
```

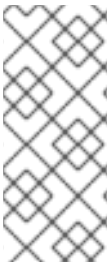
pcs resource manage コマンドまたは **pcs resource unmanage** コマンドではリソースグループの名前を指定することができます。コマンドはグループ内の全リソースで動作するため、一つのコマンドでグループ内の全リソースをアンマネージモードまたはマネージモードにしてから、含まれているリソースを個別に管理することができます。

第8章 高度なリソースタイプ

本章では Pacemaker で対応している高度なリソースタイプについて説明しています。

8.1. リソースのクローン

リソースのクローンを作成することでそのリソースを複数のノードで実行することができます。たとえば、クローンのリソースを使って複数の IP インスタンスを設定しクラスターノード全体の負荷を分散します。リソースエージェントで対応しているリソースはすべてクローン作成が可能です。クローンとは 1 リソースまたは 1 リソースグループで構成されます。



注記

クローンに適しているのは同時に複数のノードで実行することができるリソースのみです。たとえば、共有ストレージデバイスから **ext4** などのクラスター化していないファイルシステムをマウントする **Filesystem** リソースなどのクローンは作成しないでください。**ext4** パーティションはクラスターを認識しないため、同時に複数のノードから繰り返し行われる読み取りや書き込み操作には適していません。

8.1.1. クローンリソースの作成と削除

リソースの作成とそのリソースのクローン作成を同時に行う場合は次のコマンドを使用します。

```
pcs resource create resource_id standard:provider:type|type [resource options] ¥
--clone [meta clone_options]
```

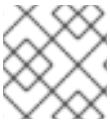
クローンの名前は ***resource_id*-clone** になります。

リソースグループの作成とそのリソースグループのクローン作成は一つのコマンドではできません。

作成済みリソースまたはリソースグループのクローンは次のコマンドで作成できます。

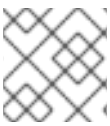
```
pcs resource clone resource_id | group_name [clone_options]...
```

クローンの名前は ***resource_id*-clone** または ***group_name*-clone** になります。



注記

リソース設定の変更が必要なのは一つのノードのみです。



注記

制約を設定する場合はグループ名またはクローン名を必ず使用します。

リソースのクローンを作成すると、その名前はリソース名に **-clone** を付けた名前が付けられます。次のコマンドではタイプが **apache** の **webfarm** というリソースとそのクローンとして **webfarm-clone** というリソースを作成します。

```
# pcs resource create webfarm apache clone
```

リソースまたはリソースグループのクローンを削除する場合は次のコマンドを使用します。リソースやリソースグループ自体は削除されません。

```
pcs resource unclone resource_id | group_name
```

リソースオプションについては 「リソースの作成」 を参照してください。

クローンのリソースに指定できるオプションを 表8.1「クローンのリソース用オプション」 に示します。

表8.1 クローンのリソース用オプション

フィールド	説明
priority, target-role, is-managed	クローン元のリソースから継承されるオプション、表5.3「リソースのメタオプション」 を参照
clone-max	開始するリソースのコピー数、クラスター内のノード数がデフォルト設定
clone-node-max	単一ノードで起動可能なリソースのコピー数、デフォルト値は **1**
notify	クローンのコピーの起動または停止を行う場合、動作の前と動作が正しく行われたときに他の全コピーに通知する、使用できる値: **false**、**true** (デフォルト値は **false**)
globally-unique	各クローンのコピーに異なる機能を行わせるかどうか、使用できる値: **false**、**true** このオプションの値が **false** の場合はリソースが実行しているいずれのノードであってもすべて同じ動作を行うため、1 台のマシンごと実行できるクローンのコピーは 1 つ このオプションの値が **true** の場合は任意のマシンで実行中のクローンのコピーは別のマシンまたは同じマシンで実行している他のコピーとは同じにならない、**clone-node-max** の値が「1」より大きい場合にはデフォルト値は **true** になり、これ以外の場合は **false** がデフォルト値になる
ordered	複数のコピーは順番に起動される (同時には起動しない)、使用できる値: **false**、**true** (デフォルト値は **false**)
interleave	順序の制約の動作を変更 (クローンとマスター間) 、クローンの 1 コピーが起動／停止したらそのクローンの残りのコピーの起動／停止を待たなくても同じノードにある別のクローンのコピーは起動／停止を開始できる、使用できる値: **false**、**true** (デフォルト値は **false**)

8.1.2. 制約のクローン作成

ほとんどの場合、アクティブなクラスターノードに対してクローンのコピーはひとつです。ただし、**clone-max** にはクラスター内のノード合計数より小さい数をリソースのクローン数の値として設定することができます。この場合、リソースの場所制約を付けたコピーを優先的に割り当てるノードを示すことができます。クローンの ID を使用する点以外、制約は通常のリソースの場合と全く変わらずクローンに記述されます。

次のコマンドでは場所の制約を作成し、リソースのクローン **webfarm-clone** が **node1** に優先的に割り当てられるようにしています。

```
# pcs constraint location webfarm-clone prefers node1
```

順序の制約の動作はクローンの場合、若干異なります。以下の例では、**webfarm-stats** は先に起動すべき **webfarm-clone** のコピーがすべて起動完了するのを待ってから起動します。起動できる **webfarm-clone** のコピーがない場合にのみ **webfarm-stats** の作動が阻止されます。また、停止の場合には **webfarm-clone** は **webfarm-stats** が停止完了するのを待ってから停止します。

```
# pcs constraint order start webfarm-clone then webfarm-stats
```

通常のリソース (またはリソースグループ) をクローンとコロケートすると、そのリソースはクローンの複数コピーが実行されている複数マシンいずれでも実行できるということになります。どのコピーが実行しているマシンでそのリソースを実行させるかはクローンが実行している場所とそのリソース自体の場所の優先度に応じて選択されます。

クローン同士でのコロケーションも可能です。この場合、クローンに対して許可できる場所はそのクローンが実行中のノード (または実行するノード) に限定されます。割り当ては通常通り行われます。

次のコマンドでは場所の制約を作成して、**webfarm-stats** リソースが必ず **webfarm-clone** の実行中のコピーと同じノードで実行されるようにしています。

```
# pcs constraint colocation add webfarm-stats with webfarm-clone
```

8.1.3. 粘着性のクローン作成

安定性のある割り当てパターンにするためクローンはデフォルトで若干の粘着性を備えています。**resource-stickiness** に値を与えないとクローンは 1 の値を使用します。値を小さくすることで他のリソースのスコア計算への阻害を最小限に抑えながら、Pacemaker によるクラスターノード内での不必要なコピーの移動を適切に阻止することができます。

8.2. 多状態のリソース: 複数モードのリソース

多状態リソースはクローンのリソースの特性です。 **Master** と **Slave**、2 種類の動作モードのいずれかになれます。インスタンス起動時は **Slave** 状態でなければならないという制限以外、モード名に特別な意味はありません。

次のコマンド一つでクローンをマスター／スレーブクローンとして作成することができます。

```
pcs resource create resource_id standard:provider:type|type [resource options] ¥
--master [meta master_options]
```

マスター／スレーブクローンの名前は ***resource_id*-master** になります。

また、作成済みのリソースまたはリソースグループからマスター／スレーブリソースを作成することもできます。このコマンドを使用する場合はマスター／スレーブクローンの名前を指定することができます。名前を指定しない場合は ***resource_id*-master** または ***group_name*-master** になります。

```
pcs resource master master/slave_name resource_id|group_name [master_options]
```

リソースオプションについては 「リソースの作成」 を参照してください。

多状態のリソースに指定できるオプションを 表8.2「多状態リソースのプロパティ」 に示します。

表8.2 多状態リソースのプロパティ

フィールド	説明
id	多状態リソースに付ける名前
priority、target-role、is-managed	表5.3「リソースのメタオプション」 を参照
clone-max、clone-node-max、notify、globally-unique、ordered、interleave	表8.1「クローンのリソース用オプション」 を参照
master-max	master 状態への昇格を許可するリソースのコピー数、デフォルトは 1
master-node-max	単一ノードで master 状態への昇格を許可するリソースのコピー数、デフォルトは 1

8.2.1. 多状態リソースのモニタリング

マスターリソースにのみモニタリングの動作を追加する場合は、リソースに別のモニタリング動作を追加します。ただし任意のリソース上のモニタリング動作にはすべて異なる間隔を設定しなければなりません。

次の例では ms_resource のマスターリソースに 11 秒間隔のモニタリング動作を設定しています。10 秒間隔のデフォルトのモニタリング動作に対して追加となるモニタリング動作になります。

```
# pcs resource op add ms_resource interval=11s role=Master
```

8.2.2. 多状態の制約

ほとんどの場合、多状態のリソースはアクティブな各クラスターノードごとコピーを一つ持っています。各ノードごとにはコピーを持たせていない場合は、リソースの場所制約を使ってコピーを優先的に割り当てるノードを指定します。制約の記述については通常のリソースの場合と全く変わりません。

リソースの場所制約については 「場所の制約」 を参照してください。

リソースをマスターにするかスレーブにするかを指定するコロケーション制約を作成することができます。次のコマンドはリソースのコロケーション制約を作成しています。

```
pcs constraint colocation add [master|slave] source_resource with [master|slave]
target_resource [score] [options]
```

リソースのコロケーション制約については 「リソースのコロケーション」 を参照してください。

多状態のリソースを含む順序制約を設定する場合、そのリソースに指定できる動作の一つがリソースをスレーブからマスターに昇格させる promote です。また、マスターからスレーブに降格させる demote の動作を指定することもできます。

順序制約を設定するコマンドを以下に示します。

```
pcs constraint order [action] resource_id then [action] resource_id [options]
```

リソースの順序制約については 「順序の制約」 を参照してください。

8.2.3. 多状態の粘着性

安定性のある割り当てパターンにするため多状態リソースはデフォルトで若干の粘着性を備えています。**resource-stickiness** に値を与えないと多状態リソースは 1 の値を使用します。値を小さくすることで他のリソースのスコア計算への阻害を最小限に抑えながら、Pacemaker によるクラスターノード内での不必要なコピーの移動を適切に阻止することができます。

8.3. モニタリングのリソースを使ったイベント通知

Pacemaker クラスターはイベント駆動のシステムです。イベントとはリソースの障害や設定の変更を指します。**ocf:pacemaker:ClusterMon** リソースではクラスターの状態をモニタリングしクラスターイベントのたびに警告を発します。このリソースは標準の間隔で **crm_mon** をバックグラウンドで実行し、**crm_mon** の機能を使って email メッセージ (SMTP) か SNMP トラップを送信します。また、**extra_options** パラメータを使って外部プログラムを実行することもできます。

次の例では email 通知を送信する **ClusterMon-SMTP** と言う名前の **ClusterMon** リソースを設定しています。Pacemaker イベントが発生すると email が **pacemaker@nodeX.example.com** から **pacemaker@example.com** へ送信されます。メールホストには **mail.example.com** が使用されます。このリソースはクローンとして作成されるためクラスター内のすべてのノードで実行されます。

```
# pcs resource create ClusterMon-SMTP ClusterMon --clone user=root update=30 ¥
extra_options="-T pacemaker@example.com -F pacemaker@nodeX.example.com ¥ -P PACEMAKER -H
mail.example.com"
```

次の例では **ClusterMon-SNMP** と言う名前の **ClusterMon** リソースを設定しています。このリソースの場合は root SNMP ユーザーで SNMP トラップを **snmphost.example.com** ホストに送信します。クローンとして作成されるためクラスター内のすべてのノードで実行されます。

```
# pcs resource create ClusterMon-SNMP ClusterMon user=root update=30 ¥ extra_options="-S
snmphost.example.com -C public" --clone
```

次の例では **ClusterMon-External** と言う名前の **ClusterMon** リソースを設定しています。このリソースの場合はクラスター通知を受け取った場合に行う動作を判断する **/usr/local/bin/example.sh** プログラムを実行します。クローンとして作成されるためクラスター内のすべてのノードで実行されます。

```
# pcs resource create ClusterMon-External ClusterMon --clone user=root ¥ update=30
extra_options="-E /usr/local/bin/example.sh -e 192.168.12.1"
```

8.4. PACEMAKER_REMOTE サービス

pacemaker_remote サービスを使用するとノードに **corosync** を実行させることなくクラスターに統合、クラスター側でリソースをクラスターノードのように管理させることができるようになります。これにより、仮想環境で **pacemaker** や **corosync** を実行することなく、Pacemaker クラスターで仮想環境 (KVM/LXC) およびその仮想環境内に存在するリソース群を管理させることができるようになります。

pacemaker_remote サービスの説明では次のような用語を使用しています。

- **クラスターノード** -High Availability サービス (**pacemaker** と **corosync**) を実行しているノード
- **リモートノード** — **pacemaker_remote** を実行しているノード、**corosync** クラスターメンバーシップを必要とせずクラスターにリモートで統合

- **コンテナー** — 追加リソースを含んでいる Pacemaker リソース (例えば webserver リソースを含んでいる KVM 仮想マシンリソースなど)
- **コンテナーリモートノード** — **pacemaker_remote** サービスを実行している仮想ゲストのリモートノード、クラスター管理の仮想ゲストリソースがクラスターで起動されリモートノードとしてクラスターに統合されるというようなリモートノード事例を指す場合に使用
- **pacemaker_remote** — Pacemaker クラスター環境、スタンドアローン (非クラスター) 環境、いずれの環境下でもゲストノード (KVM and LXC) 内のアプリケーションをリモートで管理できるサービスデーモン (Pacemaker のローカルリソース管理デーモン (LRMD) の拡張バージョンで、ゲスト上にある LSB、OCF、upstart、systemd などのリソースの管理およびモニタリングをリモートで行うことが可能、またリモートノードで **pcs** を使用することも可)
- **LXC** — **libvirt-lxc** Linux コンテナードライバーで定義される Linux Container

pacemaker_remote サービスを実行している Pacemaker クラスターには次のような特徴があります。

- 仮想リモートノードは **pacemaker_remote** サービスを実行します (仮想マシン側ではほとんど設定を必要としません)。
- クラスターノードで実行しているクラスタースタック (**pacemaker** と **corosync**) は仮想マシンを起動し直ちに **pacemaker_remote** サービスへ接続、クラスター内に仮想マシンを統合できるようにします。

仮想マシンリモートノードとクラスターノードとの主な違いとは、リモートノードはクラスタースタックを実行していないという点です。つまり、リモートノードは定足数を採用していません。これはリモートノードがクラスタースタックに関連するスケーラビリティの制限には縛られないということにもなります。定足数の制限以外、リモートノードはリソース管理に関してはクラスターノードと同じように動作します。クラスターでは各リモートノード上のリソースを完全に管理、モニタリングすることができます。リモートノードに対して制約を作り、スタンバイモードにしたり、クラスターノードで行うような動作をリモートノードに行わせることもできます。リモートノードはクラスターノード同様クラスターの状態出力に表示されます。

8.4.1. コンテナーリモートノードのリソースオプション

仮想マシンまたは LXC リソースがリモートノードとして動作するよう設定している場合には **VirtualDomain** リソースを作成して仮想マシンを管理させます。**VirtualDomain** リソースに設定できるオプションの詳細については次のコマンドで確認してください。

```
# pcs resource describe VirtualDomain
```

VirtualDomain リソースオプションの他にもリソースをリモートノードとして有効にして接続パラメーターを定義するメタデータオプションを設定することもできます。表8.3「KVM/LXC リソースをリモートノードとして設定するメタデータオプション」 にメタデータオプションを示します。

表8.3 KVM/LXC リソースをリモートノードとして設定するメタデータオプション

フィールド	デフォルト	説明
remote-node	<none>	このリソースで定義するリモートノードの名前、リソースをリモートノードとして有効にしてノードの識別に使用する固有名を定義する (他にパラメーターが設定されていない場合はこの値がポート 3121 で接続するホスト名とみなされる) **警告**: この値はいずれのリソース、ノード ID とも重複不可

フィールド	デフォルト	説明
remote-port	3121	**pacemaker_remote** へのゲスト接続に使用するカスタムのポートを設定
remote-addr	**remote-node** value used as hostname	リモートノード名がゲストのホスト名ではない場合に接続する IP アドレスまたはホスト名
remote-connect-timeout	60s	保留中のゲスト接続がタイムアウトするまでの時間

次のコマンドでは **vm-guest1** と言う名前の **VirtualDomain** リソースを作成しています。**remote-node** メタ属性を使ってリソースを実行することができるリモートノードになります。

```
# pcs resource create vm-guest1 VirtualDomain hypervisor="qemu:///system" config="vm-guest1.xml" meta remote-node=guest1
```

8.4.2. ホストとゲストの認証

クラスターノードとリモートノード間の接続の認証および暗号化は TCP ポート 3121 の PSK 暗号化／認証を使った TLS で行われます。つまり、クラスターノードとリモートノードの両方が同じプライベートキーを共有しなければならないということです。デフォルトでは、このキーはクラスターノード、リモートノードいずれも **/etc/pacemaker/authkey** に配置する必要があります。

8.4.3. デフォルトの pacemaker_remote オプションの変更

デフォルトのポートまたは Pacemaker や **pacemaker_remote** いずれかの **authkey** の場所を変更する必要がある場合は、両方のデーモンに反映させることができる環境変数を設定することができます。以下のように **/etc/sysconfig/pacemaker** ファイルに配置するとこの環境変数を有効にすることができます。

```
#==#==# Pacemaker Remote
# Use a custom directory for finding the authkey.
PCMK_authkey_location=/etc/pacemaker/authkey
#
# Specify a custom port for Pacemaker Remote connections
PCMK_remote_port=3121
```

8.4.4. 設定の概要: KVM リモートノード

本セクションでは、**libvirt** と KVM 仮想ゲストを使って Pacemaker に仮想マシンを起動させてからそのマシンをリモートノードとして統合させる手順についての概要を簡単に説明しています。

1. 仮想化ソフトウェアのインストールと、クラスターノードでの **libvirtd** サービスの有効化が完了したら、すべてのクラスターノードと仮想マシンに **/etc/pacemaker/authkey** のパスで **authkey** を配置します。これにより安全なリモート通信と認証が行えるようになります。

   次のコマンドで **authkey** を作成します。

   ```
   # dd if=/dev/urandom of=/etc/pacemaker/authkey bs=4096 count=1
   ```

2. **pacemaker_remote** パッケージのインストール、**pacemaker_remote** サービスの起動、このサービスの起動時の実行を有効化、ファイアウォールでの TCP ポート 3121 の通信の許可をすべての仮想マシンでそれぞれ行います。

```
# yum install pacemaker-remote resource-agents
# systemctl start pacemaker_remote.service
# systemctl enable pacemaker_remote.service
# firewall-cmd --add-port 3121/tcp --permanent
```

3. 各仮想マシンに静的ネットワークアドレスと固有のホスト名を与えます。

4. 仮想マシン管理用の **VirtualDomain** リソースエージェントを作成するため、Pacemaker にディスク上のファイルにダンプする仮想マシンの XML 設定ファイルが必要になります。例えば、**guest1** と言う名前の仮想マシンを作成している場合は、次のコマンドを使ってホスト上のいずれかの場所にあるファイルに XML をダンプします。

```
# virsh dumpxml guest1 > /virtual_machines/guest1.xml
```

5. **VirtualDomain** リソースを作成し、**remote-note** リソースメタオプションを設定して仮想マシンがリソースを実行できるリモートノードであることを示します。

   以下の例では、メタ属性の **remote-node=guest1** 使って、このリソースが **guest1** と言うホスト名でクラスターに統合可能なリモートノードであることを pacemaker に伝えています。仮想マシンが起動すると、クラスターによりホスト名 **guest1** で仮想マシンの **pacemaker_remote** サービスへの通信が試行されます。

```
# pcs resource create vm-guest1 VirtualDomain hypervisor="qemu:///system"
config="vm-guest1.xml" meta remote-node=guest1
```

6. **VirtualDomain** リソースの作成が完了したら、リモートノードはクラスター内の他のノードと全く同じように扱うことができるようになります。例えば、リソースを作成してリソースの制約をそのリソースに配置しリモートノードで実行させることができます。

```
# pcs resource create webserver apache params
configfile=/etc/httpd/conf/httpd.conf op monitor interval=30s
# pcs constraint webserver prefers guest1
```

   リモートノードがクラスターに統合されたら、リモートノードで Pacemaker を実行しているかのようにリモートノード自体で **pcs** コマンドを実行できるようになります。

第9章 PACEMAKER ルール

構成をより動的にする場合にルールを利用することができます。よくある例の一つとして、就業時間内は **resource-stickiness** に任意の値を設定して最優先の場所に戻されるのを防ぎ、週末など誰もアウテージに気付かない間に別の値を設定します。

これ以外にも、時間に応じて異なる処理グループにマシンを割り当て(ノード属性を使用)、場所の制約を作成する時にその属性を使用する方法もあります。

各ルールには日付の式など各種の式の他、他のルールも含ませることができます。各種の式の結果が **boolean-op** フィールドに応じて処理され、最終的にそのルールが **true** または **false** どちらの評価になるかが確定されます。ルールが使用された状況に応じて次に起こる動作は異なります。

表9.1 ルールのプロパティ

フィールド	説明
role	ルールの適用をリソースが指定ロールの状態の場合に限定する、使用できる値: **Started**、**Slave**、**Master** (注意: **role="Master"** の付いたルールはクローンのコピーの初期の場所は判断できません。実行中のどのコピーを昇格させるかを指定するだけです。)
score	ルールが **true** の評価になった場合に適用するスコア
score-attribute	ルールが **true** の評価になった場合に検索しスコアとして使用するノードの属性、場所の制約の一部となるルールにのみ使用を限定
boolean-op	複数の式オブジェクトからの結果の処理方法、使用できる値: **and** と **or**、デフォルトは **and**

9.1. ノード属性の式

ノードで定義される属性に応じてリソースを制御する場合にノード属性の式を使用します。

表9.2 式のプロパティ

フィールド	説明
value	比較のためユーザーによって入力される値
attribute	テスト用のノード属性
type	値のテスト方法を指定、使用できる値: **string**、**integer**、**version**

フィールド	説明
operation	実行する比較動作、使用できる値: * **lt** - ノード属性の値が **value** 未満の場合に True * **gt** - ノード属性の値が **value** を越える場合に True * **lte** - ノード属性の値が **value** 未満または同等になる場合に True * **gte** - 属性の値が **value** を越えるまたは同等になる場合に True * **eq** - ノード属性の値が **value** と同等になる場合に True * **ne** - ノード属性の値が **value** と同等ではない場合に True * **defined** - ノードに指定属性がある場合に True * **not_defined** - ノードに指定属性がない場合に True

9.2. 時刻と日付ベースの式

現在の日付と時刻に応じてリソースまたはクラスターオプションを制御する場合に日付の式を使用します。オプションで日付の詳細を含ませることができます。

表9.3 日付の式のプロパティ

フィールド	説明
start	ISO8601 仕様に準じた日付と時刻
end	ISO8601 仕様に準じた日付と時刻
operation	状況に応じて現在の日付と時刻を start の日付と end の日付のいずれかまたは両方と比較する、使用できる値: * **gt** - 現在の日付と時刻が **start** 以降の場合は True * **lt** - 現在の日付と時刻が **end** 以前の場合は True * **in-range** - 現在の日付と時刻が **start** 以降で且つ **end** 以前の場合は True * **date-spec** - 現在の日付と時刻に対して cron のような比較を行う

9.3. 日付の詳細

時刻に関する cron 系の式を作成する場合は日付の詳細を使用します。各フィールドには単一の数字または範囲を入力することができます。0 にデフォルト設定する代わりに未入力しておくとそのフィールドは無視されます。

例えば、**monthdays="1"** とすると毎月第一日目になり、**hours="09-17"** とすると 9am から 5pm の間の時間ということになります (9am と 5pm を含む)。ただし、**weekdays="1,2"** や **weekdays="1-2,5-6"** は複数の範囲が含まれるため使用できません。

表9.4 日付詳細のプロパティ

フィールド	説明
id	日付の固有となる名前
hours	使用できる値: 0-23
monthdays	使用できる値: 0-31 (月および年による)
weekdays	使用できる値: 1-7 (1=月曜日、7=日曜日)
yeardays	使用できる値: 1-366 (年による)
months	使用できる値: 1-12
weeks	使用できる値: 1-53 (weekyear による)
years	グレゴリオ暦 (新暦) に準じる
weekyears	グレゴリオ暦 (新暦) とは異なる場合がある、例: **2005-001 Ordinal** は **2005-01-01 Gregorian** であり **2004-W53-6 Weekly** でもある
moon	使用できる値: 0-7 (0 は新月、4 は満月)

9.4. 期間

operation=in_range で **end** の値が与えられていない場合は期間を使ってその値を算出します。**date_spec** オブジェクトと同じフィールドがありますが制限はありません (つまり 19 ヶ月の期間を持たせることが可能)。**date_specs** 同様、未入力のフィールドは無視されます。

9.5. PCS を使ってルールを設定する

ルールを設定する場合は次のコマンドを使用します。**score** を省略すると INFINITY にデフォルト設定されます。**id** を省略すると *constraint_id* で生成されます。*rule_type* は **expression** または **date_expression** のいずれかにしてください。

```
pcs constraint rule add constraint_id [rule_type] [score=score [id=rule_id]
expression|date_expression|date_spec options
```

ルールを削除する場合は次のコマンドを使用します。削除するルールがその制約内で最後のルールになる場合はその制約も削除されることになります。

```
pcs constraint rule remove rule_id
```

9.6. 時刻ベースの式のサンプル

次のコマンド設定の場合は 2005 年以内ならいつでも true となります。

```
# pcs constraint location Webserver rule score=INFINITY date-spec years=2005
```

次の式では月曜日から金曜日の 9am から 5pm の間が true となる式を設定しています。hours の値 16 には時間の値が一致する 16:59:59 まで含まれます。

```
# pcs constraint location Webserver rule score=INFINITY date-spec hours="9-16"
weekdays="1-5"
```

次の式では 13 日の金曜日に満月となるときに true になる式を設定しています。

```
# pcs constraint location Webserver rule date-spec weekdays=5 monthdays=13 moon=4
```

9.7. リソースの場所の確定にルールを使用する

次のコマンドを使用するとルールを使ってリソースの場所を確定することができます。

```
pcs resource constraint location resource_id rule [rule_id] [role=master|slave]
[score=score expression]
```

expression には以下のいずれかを使用します。

- **defined|not_defined *attribute***
- ***attribute* lt|gt|lte|gte|eq|ne *value***
- **date [start=*start* [end=*end* operation=gt|lt|in-range**
- **date-spec *date_spec_options***

第10章 PACEMAKER クラスターのプロパティ

クラスター動作中に起こる可能性がある状況に直面した場合にクラスターのプロパティでクラスターの動作を制御します。

- 表10.1「クラスターのプロパティ」 ではクラスターのプロパティオプションを説明します。
- 「クラスターのプロパティの設定と削除」 ではクラスタープロパティの設定方法について説明します。
- 「クラスタープロパティ設定のクエリー」 では現在設定されているクラスタープロパティを表示させる方法について説明します。

10.1. クラスタープロパティとオプションの要約

Pacemaker クラスターのプロパティのデフォルト値および設定可能な値などを 表10.1「クラスターのプロパティ」 で簡単に示します。

注記

表に記載しているプロパティ以外にもクラスターソフトウェアで公開されるクラスタープロパティがあります。このようなプロパティについては、そのデフォルト値を別の値には変更しないよう推奨しています。

表10.1 クラスターのプロパティ

オプション	デフォルト	説明
batch-limit	30	TE (Transition Engine) に並列実行を許可するジョブ数（ネットワークおよびクラスターノードの負荷および速度により正しい値は異なる)
migration-limit	-1 (無制限)	一つのノードで TE に並列実行を許可する移行ジョブ数
no-quorum-policy	stop	クラスターが定足数を持たない場合に行う動作、使用できる値: * ignore - 全リソースの管理を続行 * freeze - リソースの管理は続行するが、影響を受けるパーティション外のノードのリソースは復帰させない * stop - 影響を受けるクラスターパーティション内の全リソースを停止する * suicide - 影響を受けるクラスターパーティション内の全ノードを排他処理する
symmetric-cluster	true	デフォルトでいずれのノード上でもリソースの実行を許可するかどうかを指定

オプション	デフォルト	説明
stonith-enabled	true	障害が発生しているノードおよび停止できないリソースがあるノードを排他処理するかどうかを指定、データ保護が必要な場合は **true** に設定すること **true** または未設定の場合、STONITH リソースが設定されていない限りクラスターによりリソースの起動が拒否される
stonith-action	reboot	STONITH デバイスに送信する動作、使用できる値: **reboot**、**off** (**poweroff** の値も使用できるがレガシーデバイスでの使用に限定)
cluster-delay	60s	ネットワーク経由の往復遅延 (動作の実行は除く)、ネットワークおよびクラスターノードの負荷および速度により正しい値は異なる
stop-orphan-resources	true	削除したリソースの停止を行うかどうかを指定
stop-orphan-actions	true	削除した動作の取り消しを行うかどうかを指定
start-failure-is-fatal	true	起動の失敗をリソースに対して致命的と処理するかどうかを指定、**false** に設定するとリソースの **failcount** と **migration-threshold** の値を使用する (リソース用の **migration-threshold** オプションの設定は 「障害発生のためリソースを移動する」 を参照)
pe-error-series-max	-1 (all)	ERROR で保存する PE インプットの数、問題の報告時に使用
pe-warn-series-max	-1 (all)	WARNING で保存する PE インプットの数、問題の報告時に使用
pe-input-series-max	-1 (all)	保存する通常の PE インプットの数、問題の報告時に使用
cluster-infrastructure		Pacemaker が現在実行中のメッセージングスタック、情報および診断目的で使用 (ユーザー設定不可)
dc-version		クラスターの DC (Designated Controller) 上にある Pacemaker のバージョン、診断目的で使用 (ユーザー設定不可)
last-lrm-refresh		Local Resource Manager による最後のリフレッシュ (epoca のため秒単位）、診断目的で使用 (ユーザー設定不可)
cluster-recheck-interval	60	オプション、リソースのパラメーター、制約に対する時間ベースの変更のポーリング間隔、使用できる値: ゼロの場合ポーリング無効、正の値の場合は秒単位の間隔 (5min など他の SI 単位の指定がない限り)
default-action-timeout	20s	Pacemaker 動作のタイムアウト値、クラスターオプションとして設定されるデフォルト値よりリソース自体の動作に対するセッティングの方が常に優先される

オプション	デフォルト	説明
maintenance-mode	false	クラスターに無干渉モードになり別途指示があるまでサービスの起動や停止を行わないよう指示、メンテナンスモードが完了するとサービスの現在の状態に関する整合性チェックを行ってから必要に応じてそのサービスの停止または起動を行う
shutdown-escalation	20min	指定した時間を過ぎると正しいシャットダウンの試行を中断し終了 (高度な使用目的に限定)
stonith-timeout	60s	STONITH 動作の完了まで待機する時間
stop-all-resources	false	クラスターによる全リソースの停止
default-resource-stickiness	5000	リソースを現在の場所に優先的に留まらせる値 (この値はクラスターオプションではなくリソースまたは動作として設定することを推奨)
is-managed-default	true	リソースの起動および停止をクラスターに許可するかどうかを指定 (この値はクラスターオプションではなくリソースまたは動作として設定することを推奨)
enable-acl	false	(Red Hat Enterprise Linux 6.6 以降) クラスターに **pcs acl** コマンドで設定したアクセス制御リストの使用を許可するかどうかを指定

10.2. クラスターのプロパティの設定と削除

クラスタープロパティの値を設定する場合は次の **pcs** コマンドを使用します。

```
pcs property set property=value
```

例えば、**symmetric-cluster** の値を **false** に設定する場合は次のコマンドを使用します。

```
# pcs property set symmetric-cluster=false
```

設定からクラスタープロパティを削除する場合は次のコマンドを使用します。

```
pcs property unset property
```

代わりに **pcs property set** コマンドの値フィールドを空白にしてもクラスタープロパティを削除することができます。これによりそのプロパティの値がデフォルト値に戻されます。例えば、以前に **symmetric-cluster** プロパティを **false** に設定したことがある場合は、次のコマンドにより設定した値が削除され、**symmetric-cluster** の値がデフォルト値の **true** に戻されます。

```
# pcs property set symmetic-cluster=
```

10.3. クラスタープロパティ設定のクエリー

ほとんどの場合、各種のクラスターコンポーネントの値を表示するため **pcs** コマンドを使用する

際、**pcs list** または **pcs show** を交互に使用することができます。次の例では **pcs list** は複数プロパティのすべての設定の全一覧表示に使用する形式になります。一方、**pcs show** は特定のプロパティの値を表示する場合に使用する形式になります。

クラスターに設定されたプロパティ設定の値を表示する場合は次の **pcs** コマンドを使用します。

```
pcs property list
```

明示的には設定されていなかったプロパティ設定のデフォルト値も含め、クラスターのプロパティ設定のすべての値を表示する場合は次のコマンドを使用します。

```
pcs property list --all
```

特定のクラスタープロパティの現在の値を表示する場合は次のコマンドを使用します。

```
pcs property show property
```

例えば、**cluster-infrastructure** プロパティの現在の値を表示する場合は次のコマンドを実行します。

```
# pcs property show cluster-infrastructure
Cluster Properties:
 cluster-infrastructure: cman
```

情報としてプロパティの全デフォルト値の一覧を表示させ、その値がデフォルト以外で設定されているかどうかを確認することができます。次のコマンドを使用します。

```
pcs property [list|show] --defaults
```

付録A クラスター作成 - RED HAT ENTERPRISE LINUX リリース 6.5 および RED HAT ENTERPRISE LINUX リリース 6.6

Red Hat Enterprise Linux 6.6 で Pacemaker を使用して Red Hat High Availability Cluster を構成する場合には、Red Hat Enterprise Linux 6 で **rgmanager** を使用してクラスターを構成する方法とは異なる管理インターフェースを備えた別の設定ツールセットが必要になります。各種のクラスターコンポーネントでの設定の違いを 「クラスター作成 - rgmanager と Pacemaker」 で簡単に示します。

Red Hat Enterprise Linux 6.6 のリリースでは Pacemaker を使ったクラスター構成に関する新機能が提供されます。Red Hat Enterprise Linux release 6.5 での **pcs** 対応と Red Hat Enterprise Linux release 6.6 での **pcs** 対応で若干異なる設定の違いについては 「Pacemaker を使用したクラスター作成 - Red Hat Enterprise Linux リリース 6.5 および Red Hat Enterprise Linux リリース 6.6」 で簡単に示します。

A.1. クラスター作成 - RGMANAGER と PACEMAKER

表A.1「gmanager と Pacemaker を使用した場合のクラスター設定に関する比較」 では Red Hat Enterprise Linux release 6.6 で Pacemaker を使用した場合と **rgmanager** を使用した場合のクラスターコンポーネントの設定方法について比較しています。

表A.1 gmanager と Pacemaker を使用した場合のクラスター設定に関する比較

設定コンポーネント	rgmanager	Pacemaker
クラスター設定ファイル	各ノード上のクラスター設定ファイルは **cluster.conf**、目的に応じて直接編集が可能、または**luci** か **ccs** を使用	クラスターおよび Pacemaker 設定ファイルは **cluster.conf** と **cib.xml**、ファイルの編集は **pcs** インターフェースを使用し直接編集は行わない
ネットワーク設定	クラスター設定前に IP アドレスおよび SSH を設定	クラスター設定前に IP アドレスおよび SSH を設定
クラスター設定ツール	**luci**、**ccs** コマンド、手作業で **cluster.conf** ファイルを編集	**pcs**
インストール	**rgmanager** のインストール (**ricci**、**luci**、リソース、フェンスエージェントなどすべての依存パッケージをインストール)、必要に応じて **lvm2-cluster** および **gfs2-utils** をインストール	**pacemaker**、**cman**、**pcs**、リソース、必要なフェンスエージェントなどのインストール、必要に応じて **lvm2-cluster** および **gfs2-utils** をインストール

設定コンポーネント	rgmanager	Pacemaker
クラスターサービスの起動	次の手順でクラスターサービスを起動、有効にする 1. **rgmanager** と **cman**、必要であれば **clvmd** と **gfs2** も起動する 2. **ricci** を起動、**luci** インターフェースを使用している場合は **luci** を起動 3. 必要なサービスに対して **chkconfig on** を実行し各ランタイムで起動するようにする または、**ccs --start** を実行しクラスターサービスの起動と有効化を行う	次の手順でクラスターサービスを起動、有効にする 1. 各ノードで **service pcsd start**、**service pcsd enable** の順で実行しランタイムで **pcsd** が起動できるようにする 2. クラスターの任意のノードで **pcs cluster start --all** を実行し **cman** と **pacemaker** を起動する
設定ツールへのアクセスの制御	**luci** の場合、**luci** にアクセスできるのは root ユーザーまたは **luci** パーミッションを持つユーザーになる、すべてのアクセスにノードの **ricci** パスワードが必要	設定用の GUI はなし
クラスター作成	クラスターの命名、クラスターに含ませるノードの定義などは **luci** または **ccs** で行うか **cluster.conf** ファイルを直接編集	クラスターの命名、クラスターに含ませるノードの定義などは **pcs cluster setup** コマンドで行う
クラスター設定を全ノードに伝える	クラスターを **luci** で設定する場合は設定は自動的に伝わる、**ccs** の場合は **--sync** オプションを使用する、**cman_tool version -r** コマンドの使用も可	クラスターおよび Pacemaker の設定ファイル **cluster.conf** と **cib.xml** はクラスター設定時またはリソースの追加時に自動的に伝わる
グローバルのクラスタープロパティ	**rgmanager** で対応している機能: * クラスターネットワーク内での IP マルチキャスティングに使用するマルチキャストアドレスの選択をシステム側で行うよう設定することが可能 * IP マルチキャスティングが利用できない場合に UDP Unicast トランスポートメカニズムが使用可能 * RRP プロトコルを使用したクラスター設定が可能	Pacemaker ではクラスターの次の機能に対応: * クラスターに **no-quorum-policy** を設定しクラスターが定足数を持たない場合のシステムの動作を指定できる * 設定可能な他のクラスタープロパティは 表10.1「クラスターのプロパティ」 を参照
ログ機能	グローバルおよびデーモン固有のログ記録設定が可能	ログ記録を手作業で設定する方法については **/etc/sysconfig/pacemaker** ファイルを参照
クラスターの検証	**luci** および **ccs** ではクラススキーマを使った自動検証、クラスターは起動時に自動的に検証される	クラスターは起動時に自動的に検証される、または **pcs cluster verify** を使っての検証も可

設定コンポーネント	rgmanager	Pacemaker
2 ノード構成のクラスターでの定足数	2 ノードのクラスターの場合、システムの定足数を確定する方法の設定が可能: * 定足数ディスクの設定 * **ccs** を使用するか **cluster.conf** ファイルを編集、**two_node=1** と **expected_votes=1** を設定し単一ノードによる定足数の維持を許可	**pcs** により自動的に 2 ノードクラスターに必要なオプションを **cman** に追加
クラスターの状態	**luci** ではクラスターの現在の状態がインターフェースの各種コンポーネントで表示され更新が可能、**ccs** コマンドの **--gethost** オプションを使用して現在の設定ファイルの表示が可能、**clustat** コマンドを使ったクラスターの状態表示が可能	**pcs status** でクラスター状態の表示が可能
リソース	定義したタイプのリソースの追加およびリソース固有のプロパティの設定は **luci** または **ccs** コマンドを使って行うか **cluster.conf** 設定ファイルを編集して行う	定義したタイプのリソースの追加およびリソース固有のプロパティの設定は **pcs resource create** で行う、Pacemaker でのクラスターリソースの設定に関する一般的な詳細は 5章 *クラスターリソースの設定* を参照
リソースの動作、グループ化、起動と停止の順序	リソースの通信方法の設定にクラスターの **サービス** を定義	Pacemaker では一緒に配置し、順番に起動と停止を行う必要があるリソースセットを定義する簡単な方法としてリソースグループを使用、またリソースの動作法は次の手順で行う: * リソース動作の一部はリソースオプションとして設定 * 場所の制約を使ってリソースを実行させるノードを指定 * 順序の制約を使ってリソースの実行順序を指定 コロケーション制約を使って任意のリソースの場所が別のリソースの場所に依存することを指定 詳細については 5章 *クラスターリソースの設定* を参照
リソース管理: リソースの移動、起動、停止	**luci** でクラスター、クラスターの個別ノード、およびクラスターサービスの管理が可能、**ccs** ではクラスターの管理が可能、クラスターサービスの管理には **clusvadm** を使用	**pcs cluster standby** コマンドを使ってノードを一時的に無効にしリソースをホストできないようにしてからリソースを移行させる、リソースの停止は **pcs resource disable** で行う

設定コンポーネント	rgmanager	Pacemaker
クラスターの設定を完全に削除	**luci** でクラスター内の全ノード削除を選択しクラスター全体を削除、クラスター内の各ノードの **cluster.conf** を削除することも可能	任意のノードのクラスター設定の削除は **pcs cluster destroy** コマンドで行う
複数のノードで実行中のリソース、複数モードで複数のノード上で実行中のリソース	該当なし	Pacemaker ではリソースのクローンを作成し複数のノードで実行させることが可能、作成したリソースのクローンをマスターリソースとスレーブリソースとして定義し複数のモードで実行させることが可能、リソースのクローンおよびマスターとスレーブリソースについては 8章*高度なリソースタイプ*を参照
フェンス機能 -- 1 ノードにつき 1 フェンス	フェンスデバイスをグローバルまたはローカルに作成しノードに追加、クラスター全体に **post-fail delay** と **post-join delay** の値を定義することが可能	**pcs stonith create** コマンドで各ノードにフェンスデバイスを作成、複数ノードの排他処理が可能なデバイスの場合は各ノード別々ではなく一度のみの設定、また一つのコマンドで **pcmk_host_map** を指定して全ノードに対するフェンスデバイスを設定することが可能、**pcmk_host_map** については 表4.1「フェンスデバイスの汎用プロパティ」 を参照、クラスター全体に対して **stonith-timeout** の値の定義が可能
1 ノードごとに複数の (バックアップ) フェンスデバイス	バックアップデバイスの定義は **luci** または **ccs** コマンドを使用するか **cluster.conf** ファイルを直接編集	フェンスレベルを設定

A.2. PACEMAKER を使用したクラスター作成 - RED HAT ENTERPRISE LINUX リリース 6.5 および RED HAT ENTERPRISE LINUX リリース 6.6

Red Hat Enterprise Linux 6.5 で Pacemaker クラスターを作成する場合はクラスター内の各ノードでクラスターの作成とクラスターサービスの起動を行う必要があります。例えば、**z1.example.com** と **z2.example.com** の各ノードで構成される **my_cluster** というクラスターを作成して各ノードでクラスターサービスを起動するには次のコマンドを **z1.example.com** と **z2.example.com** の両方で実行します。

```
[root@rhel6.5]# pcs cluster setup --name my_cluster z1.example.com z2.example.com
[root@rhel6.5]# pcs cluster start
```

Red Hat Enterprise Linux 6.6 では、クラスター作成コマンドの実行はクラスター内の一つのノードからのみ行います。次のコマンドを任意のノードから実行すると、**z1.example.com** と **z2.example.com** というノードで構成される **my_cluster** というクラスターが作成され、クラスターサービスが両方のノードで起動されます。

```
[root@rhel6.6]# pcs cluster setup --start --name my_cluster z1.example.com
z2.example.com
```

付録B PCS コマンドの使用例

Red Hat Enterprise Linux 6.6 以降のリリースに **pcs** コマンドを使用して 2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターを設定する手順を説明していきます。また、このクラスターで Apache web サーバーを設定する手順についても解説しています。

本章で説明しているクラスターを設定する場合には次のコンポーネントが必要になります。

- 2 ノード、クラスターを構成させるノードです。ここでは **z1.example.com** と **z2.example.com** という名前にしています。
- プライベートネットワーク用のネットワークスイッチ、クラスター同士の通信およびネットワーク電源スイッチやファイバーチャンネルスイッチなどのクラスターハードウェアとの通信に必要になります。
- 各ノード用の電源フェンスデバイス、ここでは APC 電源スイッチの 2 ポートを使用しています。ホスト名は **zapc.example.com** にしています。

B.1. システムの初期セットアップ

クラスター作成に使用するシステムの初期設定について説明していきます。

B.1.1. クラスターソフトウェアをインストールする

次の手順に従ってクラスターソフトウェアをインストールします。

1. 必ず **pacemaker**、**cman**、**pcs** をインストールします。

```
yum install -y pacemaker cman pcs
```

2. インストールしたら **corosync** が **cman** がない場合は起動しないようクラスターの全ノードで次のコマンドを実行します。

```
# chkconfig corosync off
```

3. 定足数がなくても必ず **cman** を起動させたい場合でクラスターにノードが 2 つ以上ある場合には次のコマンドを実行します。

```
# sed -i.sed "s/.*CMAN_QUORUM_TIMEOUT=.*/CMAN_QUORUM_TIMEOUT=0/g"
/etc/sysconfig/cman
```

B.1.2. クラスターを作成して起動する

最初のクラスターを作成する手順を説明していきます。この後このクラスターにクラスターリソースを設定していくことになります。

1. **pcs** を使ってクラスターの設定やノード間の通信を行うため、**pcs** の管理アカウントとなるユーザー ID **hacluster** のパスワードを各ノードに設定しなければなりません。ユーザー **hacluster** のパスワードは各ノードで同じパスワードにすることを推奨しています。

```
# passwd hacluster
Changing password for user hacluster.
New password:
```

```
Retype new password:
passwd: all authentication tokens updated successfully.
```

2. クラスターを設定する前に **pcsd** デーモンを起動する必要があります。このデーモンは **pcs** と連動してクラスター内のノード全体の設定を管理します。

   クラスターの各ノードで次のコマンドを実行し **pcsd** サービスを起動、またシステムの起動時に **pcsd** が有効になるよう設定します。

```
# service pcsd start
# service pcsd enable
```

3. **pcs** を実行するノードでクラスター内の各ノードの **pcs** ユーザー **hacluster** を認証します。

   次のコマンドでは、**z1.example.com** と **z2.example.com** の 2 ノードで構成されるクラスターの両ノードに対してユーザー **hacluster** の認証を **z1.example.com** 上で行っています。

```
root@z1 ~]# pcs cluster auth z1.example.com z2.example.com
Username: hacluster
Password:
z1.example.com: Authorized
z2.example.com: Authorized
```

4. 次のコマンドを **z1.example.com** で実行し **z1.example.com** と **z2.example.com** で構成される 2 ノードクラスターの **mycluster** を作成します。これによりクラスター設定ファイルがクラスター内の全ノードに伝搬されます。コマンドに **--start** オプションを含ませることでクラスター内の全ノードでクラスターサービスが起動されます。

```
[root@z1 ~]# pcs cluster setup --start --name my_cluster ¥
z1.example.com z2.example.com
z1.example.com: Succeeded
z1.example.com: Starting Cluster...
z2.example.com: Succeeded
z2.example.com: Starting Cluster...
```

5. オプションでクラスターの各ノードが起動するときにクラスターサービスを実行するよう設定することもできます。

注記

使用環境によってクラスターサービスを無効にしておきたい場合などはこの手順を省いて構いません。この手順を行うことで、ノードがダウンした場合にクラスターやリソース関連の問題をすべて解決してからそのノードをクラスターに戻すことができます。クラスターサービスを無効にしている場合には、ノードを再起動する時に **pcs cluster start** コマンドを使って手作業でサービスを起動しなければならないので注意してください。

```
# pcs cluster enable --all
```

pcs cluster status コマンドを使用するとクラスターの現在の状態を表示できます。

```
[root@z1 ~]# pcs cluster status
Cluster Status:
```

```
Last updated: Thu Jul 25 13:01:26 2013
Last change: Thu Jul 25 13:04:45 2013 via crmd on z2.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
0 Resources configured
```

B.2. 排他処理の設定

クラスター内の各ノードに対してフェンスデバイスを設定する必要があります。フェンスデバイス設定に関する全般については *4章 フェンス機能: STONITH の設定* をご覧ください。

注記

フェンスデバイスの設定をする際、そのフェンスデバイスで管理を行うノードと電源が共有されていないことを必ず確認してください。

ここでは **zapc.example.com** というホスト名の APC 電源スイッチを使ってノードの排他処理を行います。このスイッチは **fence_apc_snmp** フェンスエージェントを使用します。ノードはすべて同じフェンスエージェントで排他処理されるため、**pcmk_host_map** と **pcmk_host_list** のオプションを使ってすべてのフェンスデバイスを一つのリソースとして設定できます。

pcs stonith create コマンドを使って **stonith** リソースとしてデバイスを設定することでフェンスデバイスを作成します。次のコマンドでは **myapc** と言う名前の **stonith** リソースを設定し、**z1.example.com** ノードと **z2.example.com** ノードに対して **fence_apc_snmp** フェンスエージェントを使用します。**pcmk_host_map** オプションで **z1.example.com** をポート 1 に、**z2.example.com** をポート 2 にマッピングしています。APC デバイスのログイン値とパスワードはいずれも **apc** です。このデバイスは各ノードに対し 60 秒のモニタリング間隔をデフォルトで使用します。

ノードのホスト名を指定する場合、IP アドレスを使用することができます。

```
[root@z1 ~]# pcs stonith create myapc fence_apc_snmp params ¥
ipaddr="zapc.example.com" pcmk_host_map="z1.example.com:1;z2.example.com:2" ¥
pcmk_host_check="static-list" pcmk_host_list="z1.example.com,z2.example.com" ¥
login="apc" passwd="apc"
```

注記

fence_apc_snmp stonith デバイスを作成するときに次のような警告メッセージが表示されることがありますがこのメッセージは無視して構いません。

```
Warning: missing required option(s): 'port, action' for resource type:
stonith:fence_apc_snmp
```

次のコマンドを使うと既存の STONITH デバイスのパラメーターが表示されます。

```
[root@rh7-1 ~]# pcs stonith show myapc
 Resource: myapc (class=stonith type=fence_apc_snmp)
  Attributes: ipaddr=zapc.example.com pcmk_host_map=z1.example.com:1;z2.example.com:2
```

```
pcmk_host_check=static-list pcmk_host_list=z1.example.com,z2.example.com login=apc
passwd=apc
  Operations: monitor interval=60s (myapc-monitor-interval-60s)
```

B.3. RED HAT HIGH AVAILABILITY CLUSTER に PCS コマンドを使用して APACHE WEB サーバーを設定する

本セクションでは **pcs** を使って 2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターに Apache web サーバーをクラスターリソースとして設定する方法を説明していきます。使用する事例では、クライアントはフローティング IP アドレスを使用して Apache web サーバーにアクセスします。web サーバーはクラスター内の 2 ノードいずれかで稼働します。web サーバーが稼働しているノードが正常に動作しなくなった場合、サービスの中断を最小限に抑えながらクラスターの 2 番目のノードでの再起動が行われます。

ここではシステムに次のようなコンポーネントが必要になります。

- 2 ノードの Red Hat High Availability クラスター (電源フェンスが各ノードに設定済み)、ここでは 「クラスターを作成して起動する」 で作成したクラスターを使用します。
- パブリック仮想 IP アドレス、Apache web サーバーに必要になります。
- クラスター内のノードで使用する共有ストレージ、iSCSI または Fibre チャンネルを使用します。

web サーバーで必要とされる LVM リソース、ファイルシステムリソース、IP アドレスリソース、web サーバーリソースなどのクラスターコンポーネントを含ませた Apache リソースグループでクラスターが設定されます。このリソースグループはクラスター内の一つのノードから別のノードへのフェールオーバーが可能なため、いずれのノードでも web サーバーを稼働することができます。クラスターにリソースグループを作成する前に次の手順を行います。

1. 「LVM ボリュームを ext4 ファイルシステムで設定する」 の説明に従い **my_lv** 論理ボリュームに **ext4** ファイルシステムを設定します。
2. 「Web サーバーの設定」 の説明に従い web サーバーを設定します。
3. 「ボリュームグループの作動をクラスター内に限定する」 の説明に従い、**my_lv** を含むボリュームグループの作動はクラスターでしか行えないよう限定し、またボリュームグループが起動時にクラスター以外の場所で作動しないようにします。

上記の手順をすべて完了したら、「pcs コマンドを使用してリソースとリソースグループを作成する」の説明に従いリソースグループおよびそのグループに含ませるリソースを作成します。

B.3.1. LVM ボリュームを ext4 ファイルシステムで設定する

ここで説明している例ではクラスターのノード間で共有させるストレージに LVM 論理ボリュームを作成する必要があります。

次の手順に従い LVM 論理ボリュームを作成しその論理ボリューム上に **ext4** ファイルシステムを作成します。ここでは **/dev/sdb1** 共有パーティションを使って LVM 論理ボリュームの作成元となる LVM 物理ボリュームを格納します。

注記

LVM ボリューム、該当パーティション、クラスターノードで使用するデバイスなどはクラスターノード以外には接続しないでください。

/dev/sdb1 パーティションは共有させるストレージとなるため、この手順は一つのノードでのみ行います。

1. LVM 物理ボリュームを **/dev/sdb1** パーティション上に作成します。

```
# pvcreate /dev/sdb1
  Physical volume ”/dev/sdb1” successfully created
```

2. **/dev/sdb1** 物理ボリュームで構成される **my_vg** ボリュームグループを作成します。

```
# vgcreate my_vg /dev/sdb1
  Volume group ”my_vg” successfully created
```

3. **my_vg** ボリュームグループを使用する論理ボリュームを作成します。

```
# lvcreate -L450 -n my_lv my_vg
  Rounding up size to full physical extent 452.00 MiB
  Logical volume ”my_lv” created
```

lvs コマンドを使って論理ボリュームを表示してみます。

```
# lvs
  LV      VG      Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
  my_lv   my_vg   -wi-a---- 452.00m
  ...
```

4. **ext4** ファイルシステムを **my_lv** 論理ボリューム上に作成します。

```
# mkfs.ext4 /dev/my_vg/my_lv
mke2fs 1.42.7 (21-Jan-2013)
Filesystem label=
OS type: Linux
...
```

B.3.2. Web サーバーの設定

次の手順に従って Apache web サーバーを設定します。

1. クラスター内の各ノードに Apache HTTPD サーバーがインストールされているか確認します。また、 Apache web サーバーの状態をチェックするためクラスターに **wget** ツールもインストールしておく必要があります。

   各ノードで次のコマンドを実行します。

```
# yum install -y httpd wget
```

2. Apache リソースエージェントで Apache web サーバーの状態を取得できるようクラスター内の各ノード上の **/etc/httpd/conf/httpd.conf** ファイルに次のテキストがあるか、テキストがコメントアウトされていなか確認します。ファイルにこのテキストがなかった場合はファイルの末尾に追加します。

```

<Location /server-status>
```

```
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
</Location>
```

3. Apache で提供する web ページを作成します。クラスター内のいずれかのノードに 「LVM ボリュームを ext4 ファイルシステムで設定する」 で作成したファイルシステムをマウントし、そのファイルシステム上で **index.html** ファイルを作成したら再びファイルシステムをアンマウントします。

```
# mount /dev/my_vg/my_lv /var/www/
# mkdir /var/www/html
# mkdir /var/www/cgi-bin
# mkdir /var/www/error
# restorecon -R /var/www
# cat <<-END >/var/www/html/index.html
<html>
<body>Hello</body>
</html>
END
# umount /var/www
```

B.3.3. ボリュームグループの作動をクラスター内に限定する

次の手順でボリュームグループを設定すると、クラスターでしかボリュームグループを作動することができなくなり、またボリュームグループは起動時にクラスター以外の場所では作動しなくなります。クラスター以外のシステムでボリュームグループが作動されるとボリュームグループのメタデータが破損する恐れがあります。

この手順では **/etc/lvm/lvm.conf** 設定ファイル内の **volume_list** のエントリーを編集します。**volume_list** のエントリーに記載されているボリュームグループはクラスターマネージャーの管轄外となるローカルノードでの自動作動が許可されます。ノードのローカルな root ディレクトリやホームディレクトリに関連するボリュームグループはこのリストに含ませてください。クラスターマネージャーで管理するボリュームグループは **volume_list** のエントリーには入れないでください。ここでの手順に **clvmd** を使用する必要はありません。

クラスター内の各ノードで以下の手順を行います。

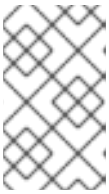
1. 次のコマンドでローカルストレージに現在設定されているボリュームグループを確認します。現在設定されているボリュームグループ一覧が出力されます。このノード上に root ディレクトリ用のボリュームグループとホームディレクトリ用のボリュームグループを別々に用意している場合は各ボリュームが以下のように出力されます。

```
# vgs --noheadings -o vg_name
  my_vg
  rhel_home
  rhel_root
```

2. **/etc/lvm/lvm.conf** 設定ファイルの **volume_list** のエントリーとして **my_vg** (クラスター用として定義したボリュームグループ) 以外のボリュームグループを追加します。例えば、root ディレクトリ用のボリュームグループ、ホームディレクトリ用のボリュームグループを別々に用意し

ている場合は、**lvm.conf** ファイルの **volume_list** の行のコメントを外して以下のように root ディレクトリ用、ホームディレクトリ用の各ボリュームグループを **volume_list** のエントリーとして追加します。

```
volume_list = [ ”rhel_root”, ”rhel_home” ]
```



注記

クラスターマネージャーの管轄外で作動させるローカルボリュームグループがノードにない場合でも **volume_list** のエントリーは **volume_list = []** と指定して初期化する必要があります。

3. 起動イメージがクラスターで制御しているボリュームグループを作動させないよう **initramfs** 起動イメージを再構築します。次のコマンドで **initramfs** デバイスを更新します。このコマンドは完了に 1 分ほどかかる場合があります。

```
# dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r)
```

4. ノードを再起動します。

注記

起動イメージを作成したノードに新しい Linux カーネルをインストールした場合、**initrd** イメージはそれを作成したときに実行していたカーネル用であってノードを再起動したら実行される新しいカーネル用ではありません。再起動の前後で **uname -r** コマンドを使って実行しているカーネルリリースを確認し必ず正しい **initrd** デバイスを使用するよう注意してください。リリースが異なる場合は新しいカーネルで再起動した後、**initrd** ファイルを更新しノードをもう一度再起動します。

5. ノードが再起動したらそのノードで **pcs cluster status** コマンドを実行しクラスターサービスが起動しているかどうか確認します。**Error: cluster is not currently running on this node** というメッセージが出力される場合は次のコマンドを実行します。

```
# pcs cluster start
```

または、クラスター内の各ノードの再起動が完了するのを待ってから次のコマンドで各ノードでのクラスターサービスの起動を行います。

```
# pcs cluster start --all
```

B.3.4. pcs コマンドを使用してリソースとリソースグループを作成する

この事例の場合、クラスターリソースを 4 つ作成する必要があります。すべてのリソースが必ず同じノードで実行されるよう **apachegroup** というリソースグループの一部として構成させます。作成するリソースを起動する順序で以下に示します。

1. **my_lvm** という名前の **LVM** リソース、「LVM ボリュームを ext4 ファイルシステムで設定する」の手順で作成した LVM ボリュームグループを使用します。

2. **my_fs** と言う名前の **Filesystem** リソース、「LVM ボリュームを ext4 ファイルシステムで設定する」 の手順で作成した **/dev/my_vg/my_lv** ファイルシステムデバイスを使用します。

3. **IPaddr2** リソース、**apachegroup** リソースグループのフローティング IP アドレスになります。物理ノードにすでに関連付けした IP アドレスは使用しないでください。**IPaddr2** リソースの NIC デバイスを指定しない場合、フローティング IP はクラスターノードで使用される静的割り当て IP アドレスと同じネットワークにしなければなりません。ネットワークが異なるとフローティング IP アドレスを割り当てる NIC デバイスを正しく検出することができません。

4. **Website** と言う名前の **apache** リソース、「Web サーバーの設定」 の手順で定義した **index.html** ファイルと Apache 設定を使用します。

次の手順で **apachegroup** リソースグループとこのグループに含ませるリソースを作成します。リソースはグループに追加した順序で起動し、またその逆順で停止します。次の手順はクラスター内いずれか一つのノードだけで行います。

1. 次のコマンドでは **my_lvm** LVM リソースを作成しています。LVM 論理ボリュームの作動がクラスター以外では行えないよう **exclusive=true** パラメーターを指定しています。この時点で **apachegroup** リソースグループはまだ存在していないため、このコマンドにより作成されることになります。

```
[root@z1 ~]# pcs resource create my_lvm LVM volgrpname=my_vg ¥
exclusive=true --group apachegroup
```

   リソースを作成するとそのリソースは自動的に起動されます。次のコマンドを使ってリソースが確かに作成、起動されたことを確認します。

```
# pcs resource show
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM): Started
```

   pcs resource disable と **pcs resource enable** のコマンドを使用すると手作業によるリソースの停止と起動をリソースごと個別に行うことができます。

2. 次のコマンドでは構成に必要な残りのリソースを作成し、**apachegroup** リソースグループに追加しています。

```
[root@z1 ~]# pcs resource create my_fs Filesystem ¥
device="/dev/my_vg/my_lv" directory="/var/www" fstype="ext4" --group ¥
apachegroup

[root@z1 ~]# pcs resource create VirtualIP IPaddr2 ip=198.51.100.3 ¥
cidr_netmask=24 --group apachegroup

[root@z1 ~]# pcs resource create Website apache ¥
configfile="/etc/httpd/conf/httpd.conf" ¥
statusurl="http://127.0.0.1/server-status" --group apachegroup
```

3. リソースおよびそのリソースを含ませるリソースグループの作成が完了したらクラスターの状態を確認します。4 つのリソースすべてが同じノードで実行していることを確認してください。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 16:38:51 2013
Last change: Wed Jul 31 16:42:14 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
```

```
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured


Online: [ z1.example.com z2.example.com ]

Full list of resources:
 myapc (stonith:fence_apc_snmp): Started z1.example.com
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM): Started z1.example.com
     my_fs (ocf::heartbeat:Filesystem): Started z1.example.com
     VirtualIP (ocf::heartbeat:IPaddr2): Started z1.example.com
     Website (ocf::heartbeat:apache): Started z1.example.com
```

「排他処理の設定」 の手順でクラスターにフェンスデバイスを設定していないとリソースはデフォルトでは起動しないので注意してください。

4. クラスターが起動し稼働し始めたら、 ブラウザで **IPaddr2** リソースとして定義した IP アドレスをポイントし "Hello" のテキストで構成されるサンプル表示が正しく表示されるか確認します。

```
Hello
```

設定したリソースが実行していない場合には **pcs resource debug-start *resource*** コマンドを実行してリソースの設定をテストしてみます。**pcs resource debug-start** コマンドの詳細については 『High Availability Add-On Reference (High Availability Add-On リファレンス)』 のガイドを参照してください。

B.3.5. リソース設定のテスト

「pcs コマンドを使用してリソースとリソースグループを作成する」 で示すようにクラスターの状態表示では全リソースが **z1.example.com** ノードで実行しています。次の手順に従い 1 番目のノードを **standby** モードにしてリソースグループが **z2.example.com** ノードにフェールオーバーするかどうかテストします。1 番目のノードをスタンバイモードにするとこのノードではリソースをホストできなくなります。

1. 次のコマンドでは **z1.example.com** を **standby** モードにしています。

```
root@z1 ~]# pcs cluster standby z1.example.com
```

2. **z1** をスタンバイモードにしたらクラスターの状態を確認します。リソースがすべて **z2** で実行しているはずです。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 17:16:17 2013
Last change: Wed Jul 31 17:18:34 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured
```

```
Node z1.example.com (1): standby
Online: [ z2.example.com ]

Full list of resources:

 myapc (stonith:fence_apc_snmp): Started z1.example.com
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM): Started z2.example.com
     my_fs (ocf::heartbeat:Filesystem): Started z2.example.com
     VirtualIP (ocf::heartbeat:IPaddr2): Started z2.example.com
     Website (ocf::heartbeat:apache): Started z2.example.com
```

定義している IP アドレスの web サイトは中断されることなく表示されているはずです。

3. 次のコマンドを実行して **z1** を **standby** モードから外します。

```
root@z1 ~]# pcs cluster unstandby z1.example.com
```

注記

ノードを **standby** モードから外してもリソースが再びそのノードにフェールバックして戻ってくるわけではありません。リソースの実行を許可するノードの制御については 『Red Hat High Availability Add-On Reference (Red Hat High Availability Add-On リファレンス)』 のクラスターリソースの設定に関する章を参照してください。

付録C 改訂履歴

改訂 2.0-7.2 **Thu Apr 23 2015** **Noriko Mizumoto**
翻訳および査読完了

改訂 2.0-7.1 **Thu Apr 23 2015** **Noriko Mizumoto**
翻訳ファイルを XML ソースバージョン 2.0-7 と同期

改訂 2.0-7 **Tue Dec 16 2014** **Steven Levine**
RHEL 6 スプラッシュページを更新し、sort_order を実装

改訂 2.0-5 **Thu Oct 9 2014** **Steven Levine**
6.6 GA リリース向けバージョン

改訂 2.0-4 **Wed Oct 8 2014** **Steven Levine**
解決済み: #1131544
ACL に関する記載を追記

改訂 2.0-2 **Wed Aug 7 2014** **Steven Levine**
6.6 Beta リリース向けバージョン

改訂 2.0-1 **Wed Jul 23 2014** **Steven Levine**
6.6 Beta ドラフト版

解決済み: #1126896、#1126018、#986462、#1045406、#1122145、#1079340
若干の編集と技術的な修正

解決済み: #1081225、#1081248、#1092720
ノード間通信機能およびノード全体での同期設定に関するサポートを含めた記載に更新

改訂 1.1-2 **Wed Nov 20 2013** **Steven Levine**
6.5 GA リリース向けバージョン

改訂 0.1-4 **Wed Oct 2 2013** **Steven Levine**
6.5 beta 初版ドラフト