Red Hat Enterprise Linux 8

高可用性クラスターの設定および管理

Red Hat High Availability Add-On の設定および管理に関するガイド

Last Updated: 2019-05-11

Red Hat Enterprise Linux 8 高可用性クラスターの設定および管理

Red Hat High Availability Add-On の設定および管理に関するガイド

法律上の通知

概要

本書は、Red Hat Enterprise Linux 8 に Red Hat High Availability Add-On をインストール、設定、および管理する方法を説明します。

目次

RED HAT ドキュメントへのフィードバック

ドキュメントの改善に関するご意見やご要望をお聞かせください。

- 特定の文章に簡単なコメントを記入する場合は、ドキュメントが Multi-page HTML 形式になっているのを確認してください。コメントを追加する部分を強調表示し、そのテキストの下に表示される **Add Feedback** ポップアップをクリックし、表示された手順に従ってください。
- より詳細なフィードバックを行う場合は、Bugzilla のチケットを作成します。
    1. Bugzilla の Web サイトにアクセスします。
    2. Component で **Documentation** を選択します。
    3. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも記入してください。
    4. **Submit Bug** をクリックします。

第1章 HIGH AVAILABILITY ADD-ON の概要

High Availability Add-On は、基幹実稼働サービスに、信頼性、スケーラビリティー、および可用性を提供するクラスター化されたシステムです。

クラスターは、連携してタスクを実行する 2 つ以上のコンピューター (**ノード** および **メンバー** と呼ばれています) で、可用性の高いサービスまたはリソースを提供するのに使用できます。複数のマシンの冗長性は、さまざまな障害から守るために使用されます。

高可用性クラスターは、単一障害点を排除し、ノードが稼働しなくなった場合に、あるクラスターノードから別のクラスターノードにサービスをフェイルオーバーすることにより、可用性が高いサービスを提供します。通常、高可用性クラスターのサービスは、データの読み取りや書き込みを行います (read-write でマウントされたファイルシステム経由)。したがって、1 つのクラスターノードが別のクラスターノードからサービスの制御を引き継ぐ際に、高可能性クラスターはデータ整合性を維持する必要があります。高可用性クラスター内のノードの障害は、クラスター外にあるクライアントからは見えません (高可用性クラスターはフェイルオーバークラスターと呼ばれることがあります)。High Availability Add-On は、高可性サービス管理コンポーネントの **Pacemaker** を介して、高可用性クラスタリングを提供します。

1.1. HIGH AVAILABILITY ADD-ON コンポーネント

High Availability Add-On は、以下の主要なコンポーネントで構成されています。

- クラスターインフラストラクチャー - クラスターとして連携するように、ノード群に基本的な機能 (設定ファイル管理、メンバーシップ管理、ロック管理、およびフェンシング) を提供します。
- 高可用性サービス管理 - 1 つのクラスターノードが動作不能になった場合は、そのクラスターノードから別のノードにサービスのフェイルオーバーを提供します。
- クラスター管理ツール - High Availability Add-On のセットアップ、設定、および管理を行うツール。このツールは、クラスターインフラストラクチャーのコンポーネント、高可用性およびサービス管理のコンポーネント、およびストレージで使用されます。

以下のコンポーネントで、High Availability Add-On を補完できます。

- Red Hat GFS2 (Global File System 2) - Resilient Storage Add-On に同梱され、High Availability Add-On で使用するクラスターファイルシステムを提供します。GFS2 により、ストレージがローカルで各クラスターノードに接続されているかのように、ブロックレベルで、複数ノードでストレージを共有できるようになります。GFS2 クラスターファイルシステムを使用する場合は、クラスターインフラストラクチャーが必要になります。
- LVM ロッキングデーモン (**lvmlockd**) - Resilient Storage Add-On に同梱され、クラスターストレージのボリューム管理を提供します。**lvmlockd** サポートには、クラスターインフラストラクチャーも必要になります。
- Load Balancer Add-On - レイヤー 4 (TCP) およびレイヤー 7 (HTTP および HTTPS) サービスで高可用性負荷分散とフェイルオーバーを提供するルーティングソフトウェアです。Load Balancer Add-On は、負荷アルゴリズムを使用して、クライアント要求を実サーバーに分散する冗長な仮想ルーターのクラスターで実行し、まとまって仮想サーバーとして機能します。Load Balancer Add-On は、Pacemaker と併用する必要はありません。

1.2. PACEMAKER の概要

Pacemaker は、クラスターリソースマネージャーです。クラスターインフラストラクチャーのメッセージング機能およびメンバーシップ機能を使用して、ノードおよびリソースレベルの障害を防ぎ、障害から復旧することで、クラスターサービスおよびリソースの可用性を最大化します。

1.2.1. Pacemaker アーキテクチャーコンポーネント

Pacemaker で設定されたクラスターは、クラスターメンバーシップを監視する個別のコンポーネントデーモン、サービスを管理するスクリプト、および異なるリソースを監視するリソース管理サブシステムで構成されます。

Pacemaker アーキテクチャーを形成するコンポーネントは、以下のとおりです。

Cluster Information Base (CIB)

XML を内部的に使用して、Designated Coordinator (DC) (CIB を介してクラスターのステータスと動作を格納および分散するために、Pacemaker により割り当てられたノード) から、他のすべてのクラスターノードに対して現在の設定とステータス情報を分散し、同期する Pacemaker 情報デーモン。

Cluster Resource Management Daemon (CRMd)

Pacemaker クラスターリソースの動作は、このデーモンを介してルーティングされます。CRMd により管理されるリソースは、必要に応じてクライアントシステムが問い合わせたり、これを移動したり、インスタンス化したり、変更したりできます。
各クラスターノードには、CRMd とリソースとの間のインターフェースとして動作する Local Resource Manager daemon (LRMd) も含まれます。LRMd は、起動、停止、ステータス情報のリレーなどのコマンドを、CRMd からエージェントに渡します。

Shoot the Other Node in the Head (STONITH)

STONITH は Pacemaker フェンシングの実装です。STONITH は、フェンス要求を処理する Pacemaker のクラスターリソースとして動作し、強制的にノードの電源をオフにし、クラスターからノードを削除してデータの整合性を確保します。STONITH は CIB で設定し、通常のクラスターリソースとして監視できます。フェンシングの概要は「フェンシングの概要」 を参照してください。

corosync

corosync は、コアメンバーシップと、高可用性クラスターのメンバー間の通信ニーズに対応するコンポーネントで、デーモンも同じ名前です。これは、High Availability Add-On が機能するのに必要です。
corosync は、これらのメンバーシップとメッセージング機能のほかに、以下も実行します。

- クォーラムのルールおよび決定を管理します。
- クラスターの複数のメンバー間の調整機能、またはメンバー間で動作するメッセージング機能を提供します。そのため、インスタンス間で、ステートフルな情報またはその他の情報を通信できる必要があります。
- **kronosnet** ライブラリーをネットワークトランスポートとして使用し、複数の冗長なリンクおよび自動フェイルオーバーを提供します。

1.2.2. 設定および管理ツール

High Availability Add-On には、クラスターのデプロイメント、監視、および管理に使用する 2 つの設定ツールが含まれます。

pcs

pcs コマンドラインインターフェースは、Pacemaker および **corosync** ハートビートデーモンを制御し、設定します。コマンドラインベースのプログラムである **pcs** は、以下のクラスター管理タスクを実行できます。

- Pacemaker/Corosync クラスターの作成および設定
- 実行中のクラスターの設定変更
- Pacemaker と Corosync の両方のリモートでの設定、起動、停止、およびクラスターのステータス情報の表示

pcsd Web UI

Pacemaker/Corosync クラスターを作成および設定するグラフィカルユーザーインターフェースです。

1.2.3. クラスターおよび Pacemaker の設定ファイル

Red Hat High Availability Add-On の設定ファイルは、**corosync.conf** および **cib.xml** です。

corosync.conf ファイルは、Pacemaker を構築するクラスターマネージャー (**corosync**) が使用するクラスターパラメーターを提供します。通常、直接 **corosync.conf** を編集するのではなく、**pcs** または **pcsd** インターフェースを使用します。ただし、このファイルを直接編集することが必要になる場合もあります。

cib.xml ファイルは、クラスターの設定、およびクラスターの全リソースの現在の状態を表す XML ファイルです。このファイルは、Pacemaker のクラスター情報ベース (CIB) により使用されます。CIB の内容は、自動的にクラスター全体に同期されます。**cib.xml** ファイルは直接編集せず、代わりに **pcs** または **pcsd** インターフェースを使用してください。

1.3. フェンシングの概要

クラスター内のノードの 1 つと通信が失敗した場合に、障害が発生したクラスターノードがアクセスする可能性があるリソースへのアクセスを、その他のノードが制限したり、解放したりできるようにする必要があります。クラスターノードが応答しない可能性があるため、そのクラスターノードと通信しても成功しません。代わりに、フェンスエージェントによる、フェンシングと呼ばれる外部メソッドを指定する必要があります。フェンスデバイスは、クラスターが使用する外部デバイスのことで、このデバイスを使用して、不安定なノードによる共有リソースへのアクセスを制限したり、クラスタノードでハードリブートを実行します。

フェンスデバイスが設定されていないと、切断されているクラスターノードが、以前使用されていたリソースが解放されていることを把握することができず、他のクラスターノードでサービスを実行できなくなる可能性があります。また、クラスターノードがそのリソースを解放したとシステムが誤って想定する場合もあり、そのためデータの破損やデータの損失が発生する可能性があります。フェンスデバイスが設定されていないと、データの整合性は保証できず、クラスター設定はサポートされません。

フェンシングの進行中は、他のクラスター操作を実行できません。クラスターノードの再起動後にフェンシングが完了するか、クラスターノードがクラスターに再参加するまで、クラスターの通常の動作を再開することはできません。

フェンシングの詳細は「RHEL 高可用性クラスターでフェンシングが重要なのはなぜですか?」を参照してください。

1.4. クォーラムの概要

クラスターの整合性と可用性を維持するために、クラスターシステムは、**クォーラム** と呼ばれる概念を

使用してデータの破損および損失を防ぎます。クラスターノードの過半数がオンラインになると、クラスターでクォーラムが確立されます。障害によるデータ破損の可能性を小さくするために、Pacemaker はデフォルトですべてのリソースを停止します (クラスターでクォーラムが確立されない場合)。

クォーラムは、投票システムを使用して確立されます。クラスターノードが通常どおり機能しない場合や、クラスターの他の部分との通信が失われた場合に、動作している過半数のノードが、そのノードを分離するように投票し、必要に応じて、接続を切断して修復します。

たとえば、6 ノードクラスターで、4 つ以上のクラスターノードが動作している場合にクォーラムが確立されます。過半数のノードがオフラインまたは利用できない状態になると、クラスターでクォーラムが確立されず、Pacemaker がクラスター化サービスを停止します。

Pacemaker におけるクォーラム機能も、**スプレットブレイン** と呼ばれ、クラスターは通信から分離されていますが、各部分は、潜在的に同じデータを書き込み、おそらく破壊または損失を引き起こし、別のクラスターとして動作し続ける現象を防ぎます。スプリットブレイン状態の詳細と、一般的なクォーラムの概念は「Exploring Concepts of RHEL High Availability Clusters - Quorum」 を参照してください。

Red Hat High Availability Add-On クラスターは、スプリットブレインの状況を回避するために、フェンシングと共に **votequorum** サービスを使用します。クラスターの各システムに多くの投票が割り当てられ、投票が過半数に達した場合に限り、クラスターの操作が続行します。

1.5. リソースの概要

クラスターリソース は、クラスターサービスで管理するプログラム、データ、またはアプリケーションのインスタンスです。このようなリソースは、クラスター環境でリソースを管理する標準的なインターフェースを提供する **エージェント** により抽象化されます。

リソースを健全な状態に保つために、リソースの定義に監視操作を追加できます。リソースの監視操作を指定しない場合は、デフォルトで監視操作が追加されます。

クラスター内のリソースの動作は、**制約** を設定して指定します。以下の制約のカテゴリーを設定できます。

- 場所の制約 - 場所の制約により、リソースを実行できるノードが決定します。
- 順序の制約 - 順序の制約により、リソースを実行する順序が決定します。
- コロケーションの制約 - コロケーションの制約により、他のリソースに対して相対的なリソースの配置先が決定します。

クラスターの最も一般的な構成要素の 1 つがリソースセットです。リソースセットは一緒に配置し、順番に起動し、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** という概念がサポートされます。

/ モジュールは、次のアセンブリーに含まれます。

1.6. RED HAT HIGH AVAILABILITY クラスターの LVM 論理ボリューム

Red Hat High Availability Add-On は、2 つの異なるクラスター設定で LVM ボリュームをサポートします。

- アクティブ/パッシブフェイルオーバー設定の HA-LVM (High Availability LVM) ボリューム。クラスターでストレージにアクセスするノードは 1 つだけになります。

- アクティブ/アクティブ設定でストレージデバイスを管理する **lvmlockd** を使用する LVM ボリューム。クラスターで、1つ以上のクラスターが同時にストレージにアクセスする必要があります。**lvmlockd** デーモンは、Resilient Storage Add-On で提供されます。

1.6.1. HA-LVM または共有ボリュームの選択

HA-LVM、または **lvmlockd** デーモンが管理する共有論理ボリュームを使用するタイミングは、デプロイされるアプリケーションまたはサービスのニーズに基づいて決定する必要があります。

- クラスターの複数のノードが、アクティブ/アクティブシステムで LVM ボリュームへの同時読み取りまたは書き込みを必要とする場合に、**lvmlockd** デーモンを使用して、ボリュームを共有ボリュームとして設定します。**lvmlockd** デーモンは、クラスターのノード全体で、LVM ボリュームのアクティベーションおよび変更を同時に調整するシステムを提供します。**lvmlockd** デーモンのロックサービスは、クラスターのさまざまなノードがボリュームと対話し、レイアウトに変更を加え、LVM メタデータを保護します。この保護は、複数のクラスタノードで同時にアクティブにされるボリュームグループを共有ボリュームとして構成することにより決まります。

- 指定した LVM ボリュームに同時にアクセスするメンバーは1つだけにする必要があり、アクティブ/パッシブに共有リソースを管理するように HA クラスターを設定した場合は、**lvmlockd** ロックサービスを使用せずに HA-LVM を使用できます。

ほとんどのアプリケーションは、その他のインスタンスと同時に実行するように設計または最適化されていないため、アクティブ/パッシブ設定での実行により適しています。共有論理ボリュームで、クラスターに対応していないアプリケーションを実行すると、パフォーマンスが低下することがあります。これは、論理ボリューム自体にクラスター通信のオーバーヘッドが発生するためです。クラスター対応のアプリケーションは、クラスターファイルシステムとクラスター対応の論理ボリュームにより発生するパフォーマンスの低下を上回るパフォーマンスの向上を実現する必要があります。実現が容易かどうかは、アプリケーションやワークロードによって異なります。クラスターの要件を判断し、アクティブ/アクティブのクラスターを最適化する努力に価値があるかどうかを判断して、2 種類の LVM のいずれかを選択します。ほとんどの場合は、HA-LVM を使用すると HA を最適化できます。

HA-LVM および **lvmlockd** を使用する共有論理ボリュームは、複数のマシンが変更を重複して行うと発生する、LVM メタデータとその論理ボリュームの破損を防ぐという点で似ています。HA-LVM では、論理ボリュームは、アクティベートする場合は排他的に行う制限があるため、一度に1つのマシンでしかアクティブになりません。そのため、ストレージドライバーのローカル (非クラスター) 実装のみが使用されます。このようにクラスターの調整オーバーヘッドが発生しないようにすると、パフォーマンスが向上します。**lvmlockd** を使用する共有ボリュームにはこのような制限はなく、ユーザーは、クラスターのすべてのマシンで論理ボリュームをアクティベートできます。これにより、クラスター対応のストレージドライバーの使用が強制され、クラスター対応のファイルシステムとアプリケーションが優先されます。

1.6.2. クラスター内での LVM ボリュームの設定

Red Hat Enterprise Linux 8 では、クラスターは Pacemaker で管理されます。HA-LVM および共有論理ボリュームは、Pacemaker クラスターと併用される場合のみサポートされ、クラスターリソースとして設定する必要があります。

- HA-LVM ボリュームを Pacemaker クラスターの一部として設定する手順は「Red Hat High Availability クラスターでのアクティブ/パッシブ Apache HTTP サーバーの設定」および「Red Hat High Availability クラスターのアクティブ/パッシブな NFS サーバーの設定」を参照してください。

この手順には、以下の手順が含まれます。

+ ** クラスターのみがボリュームグループをアクティベートできるようにする

+ ** LVM 論理ボリュームを設定する

+ ** LVM ボリュームをクラスターリソースとして設定する

- クラスターで CLVM ボリュームを設定する手順は、xref:proc_configuring-gfs2-in-a-cluster.adoc-configuring-gfs2-cluster を参照してください。

第2章 PACEMAKER の使用の開始

以下の手順では、Pacemaker クラスターを作成するのに使用するツールとプロセスの概要を説明します。ここでは、作業用のクラスターを設定することなくクラスターソフトウェアの概要およびその管理方法に関心のあるユーザーを対象としています。

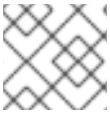
注記

ここで説明する手順では、2 つ以上のノードとフェンシングデバイスの設定が必要となるサポート対象の Red Hat クラスターは作成されません。

2.1. PACEMAKER の使用方法

この例では、RHEL 8 を実行している単一ノードと、IP アドレスが静的に割り当てられたそのノードのいずれかと同じネットワークにあるフローティング IP アドレスが必要です。

- この例で使用されているノードは、**z1.example.com** です。
- この例で使用されているフローティング IP アドレスは、192.168.122.120 です。



注記

/etc/hosts ファイルに、実行しているノードの名前があることを確認してください。

ここでは、Pacemaker を使用してクラスターを設定する方法、クラスターのステータスを表示する方法、およびクラスターサービスを設定する方法を学習します。この例では、Apache HTTP サーバーをクラスターリソースとして作成し、リソースに障害が発生した場合のクラスターの応答方法を表示します。

1. High Availability チャンネルから Red Hat High Availability Add-On ソフトウェアパッケージをインストールし、**pcsd** サービスを起動し、これを有効にします。

```
# yum install pcs pacemaker fence-agents-all
...
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

firewalld デーモンを実行している場合は、Red Hat High Availability Add-On で必要なポートを有効にします。

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --reload
```

2. クラスター内の各ノードにユーザー **hacluster** のパスワードを設定し、**pcs** コマンドを実行するノードで、クラスター内の各ノードに、**hacluster** ユーザーの認証を行います。この例では、コマンドを実行するノードだけを使用していますが、このステップは、サポート対象の Red Hat High Availability マルチノードクラスターを設定する際に必要となるため、手順に含まれています。

```
# passwd hacluster
...
# pcs host auth z1.example.com
```

3. メンバーを 1 つ含む **my_cluster** という名前のクラスターを作成し、クラスターのステータスを確認します。このコマンドで、クラスターが作成され、起動します。

```
# pcs cluster setup my_cluster --start z1.example.com
...
# pcs cluster status
Cluster Status:
 Stack: corosync
 Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
 Last updated: Thu Oct 11 16:11:18 2018
 Last change: Thu Oct 11 16:11:00 2018 by hacluster via crmd on z1.example.com
 1 node configured
 0 resources configured

PCSD Status:
  z1.example.com: Online
```

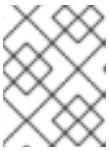
4. Red Hat High Availability クラスターでは、クラスターのフェンシングを設定することが必要になります (理由は「RHEL 高可用性クラスターでフェンシングが重要なのはなぜですか?」を参照)。ただし、ここでは基本的な Pacemaker コマンドの使用方法を説明することを目的としているため、**stonith-enabled** クラスターのオプションを **false** に設定し、フェンシングを無効にします。

警告

stonith-enabled=false に設定すると、障害のあるノードが安全にフェンスされているようにクラスターが振る舞うため、この設定は実稼働のクラスターには適していません。

```
# pcs property set stonith-enabled=false
```

5. システムに Web ブラウザーを設定し、Web ページを作成して簡単なテキストメッセージを表示します。**firewalld** デーモンを実行している場合は、**httpd** で必要なポートを有効にします。



注記

systemctl enable では、システムの起動時に、クラスターが管理するサービスを有効にすることはできません。

```
# yum install -y httpd wget
...
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload

# cat <<-END >/var/www/html/index.html
<html>
<body>My Test Site - $(hostname)</body>
</html>
END
```

- Apache リソースエージェントが Apache のステータスを取得できるようにするため、既存の設定に以下の内容を追加して、ステータスサーバーの URL を有効にします。

```
# cat <<-END > /etc/httpd/conf.d/status.conf
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
Allow from 127.0.0.1
Allow from ::1
</Location>
END
```

6. クラスターが管理する **IPaddr2** および **apache** を作成します。「IPaddr2」リソースはフローティング IP であるため、物理ノードに関連付けられている IP アドレスは使用できません。「IPaddr2」リソースの NIC デバイスを指定しない場合は、そのノードで使用される、静的に割り当てられた IP アドレスと同じネットワークにフローティング IP が存在する必要があります。
利用可能なリソースタイプの一覧を表示するには、**pcs resource list** コマンドを使用します。指定したリソースタイプに設定できるパラメーターを表示するには、**pcs resource describe resourcetype** コマンドを使用します。たとえば、以下のコマンドは、**apache** というタイプのリソースに設定できるパラメーターを表示します。

```
# pcs resource describe apache
...
```

この例では、IP アドレスリソースおよび apache リソースの両方が、**apachegroup** という名前のグループに含まれるように設定します。これにより、両リソースが一緒に保存され、作業用のマルチノードクラスターを設定する際に、同じノードで実行できます。

```
# pcs resource create ClusterIP ocf:heartbeat:IPaddr2 ip=192.168.122.120 --group
apachegroup

# pcs resource create WebSite ocf:heartbeat:apache
configfile=/etc/httpd/conf/httpd.conf statusurl="http://localhost/server-status" --group
apachegroup

# pcs status
Cluster name: my_cluster
Stack: corosync
Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
Last updated: Fri Oct 12 09:54:33 2018
Last change: Fri Oct 12 09:54:30 2018 by root via cibadmin on z1.example.com

1 node configured
2 resources configured

Online: [ z1.example.com ]

Full list of resources:

Resource Group: apachegroup
    ClusterIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
    WebSite    (ocf::heartbeat:apache):        Started z1.example.com
```

```
PCSD Status:
  z1.example.com: Online
...
```

クラスターリソースを設定したら、**pcs resource config** コマンドを使用して、そのリソースに設定したオプションを表示します。

```
# pcs resource config WebSite
Resource: WebSite (class=ocf provider=heartbeat type=apache)
 Attributes: configfile=/etc/httpd/conf/httpd.conf statusurl=http://localhost/server-status
 Operations: start interval=0s timeout=40s (WebSite-start-interval-0s)
             stop interval=0s timeout=60s (WebSite-stop-interval-0s)
             monitor interval=1min (WebSite-monitor-interval-1min)
```

7. ブラウザーで、設定済みのフローティング IP アドレスを使用して作成した Web サイトを開くように指定します。定義したテキストメッセージが表示されるはずです。

8. Apache Web サービスを停止し、クラスターのステータスを確認します。**killall -9** を使用すると、アプリケーションレベルのクラッシュをシミュレートします。

```
# killall -9 httpd
```

クラスターのステータスを確認します。Web サービスは停止したためアクションは失敗しますが、クラスターソフトウェアがサービスを再起動したため、Web サイトに引き続きアクセスできることが確認できるはずです。

```
# pcs status
Cluster name: my_cluster
...
Current DC: z1.example.com (version 1.1.13-10.el7-44eb2dd) - partition with quorum
1 node and 2 resources configured

Online: [ z1.example.com ]

Full list of resources:

Resource Group: apachegroup
    ClusterIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
    WebSite    (ocf::heartbeat:apache):        Started z1.example.com

Failed Resource Actions:
* WebSite_monitor_60000 on z1.example.com 'not running' (7): call=13, status=complete,
exitreason='none',
    last-rc-change='Thu Oct 11 23:45:50 2016', queued=0ms, exec=0ms

PCSD Status:
    z1.example.com: Online
```

サービスが再開すると、障害が発生したリソースの障害 (failure) ステータスが削除されるため、クラスターステータスの閲覧時に、障害が発生したアクションの通知が表示されなくなります。

```
# pcs resource cleanup WebSite
```

9. クラスターおよびクラスターのステータスを確認したら、ノードでクラスターサービスを停止します。この手順の確認のためにサービスを起動したノードが 1 つだけであっても、**--all** パラメーターを追加します。これにより実際のマルチノードクラスターの全ノードでクラスターサービスが停止します。

```
# pcs cluster stop --all
```

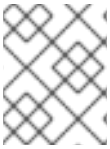
2.2. フェイルオーバーの設定方法

以下の手順では、サービスを実行する Pacemaker クラスターの作成方法を紹介します。このサービスは、サービスを実行しているノードが利用できなくなると、現在のノードから別のノードにフェイルオーバーします。この手順を行って、2 ノードクラスターでサービスを作成する方法を学び、サービスを実行しているノードが失敗するとどうなるかを確認します。

この手順では、Apache HTTP サーバーを実行する 2 ノード Pacemaker クラスターを設定します。その後、1 つのノードで Apache サービスを停止し、サービスが利用可能な状態がどのように維持されるかを確認できます。

この手順では、相互に通信が可能な Red Hat Enterprise Linux 8 を実行するノードを 2 つ用意するという要件を満たし、フローティング IP アドレスが、IP アドレスが静的に割り当てられているノードのいずれかと同じネットワークにあることが必要です。

- この例で使用されるノードは、**z1.example.com** および **z2.example.com** です。
- この例で使用されているフローティング IP アドレスは、192.168.122.120 です。



注記

ノードの **/etc/hosts** ファイルに、使用しているノード名が登録されていることを確認します。

1. 両方のノードで、High Availability チャンネルから Red Hat High Availability Add-On ソフトウェアパッケージをインストールし、 **pcsd** サービスを起動し、これを有効にします。

```
# yum install pcs pacemaker fence-agents-all
...
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

firewalld デーモンを実行している場合は、Red Hat High Availability Add-On で必要なポートを有効にします。

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --reload
```

2. クラスター内の両方のノードに、ユーザー **hacluster** のパスワードを設定します。

```
# passwd hacluster
```

3. **pcs** コマンドを実行するノードで、クラスター内の各ノードのユーザー **hacluster** の認証を行います。

```
# pcs host auth z1.example.com z2.example.com
```

4. 両方のノードがクラスターメンバーとなる **my_cluster** という名前のクラスターを作成します。このコマンドを実行すると、クラスターが作成され、起動します。**pcs** 設定コマンドはクラスター全体に適用されるため、このコマンドは、クラスター内のいずれかのノードで実行してください。
   クラスター内のいずれかのノードで、以下のコマンドを実行します。

```
# pcs cluster setup my_cluster --start z1.example.com z2.example.com
```

5. Red Hat High Availability クラスターでは、クラスターのフェンシングを設定することが必要になります (理由は「RHEL 高可用性クラスターでフェンシングが重要なのはなぜですか?」を参照)。ただし、ここでは、この設定でフェイルオーバーがどのように機能するかを説明することを目的としているため、**stonith-enabled** クラスターのオプションを **false** に設定し、フェンシングを無効にします。

警告

stonith-enabled=false に設定すると、障害のあるノードが安全にフェンスされているようにクラスターが振る舞うため、この設定は実稼働のクラスターには適していません。

```
# pcs property set stonith-enabled=false
```

6. クラスターを作成し、フェンシングを無効にしたら、クラスターのステータスを確認します。

注記

pcs cluster status コマンドを実行したときの出力は、一時的に、システムコンポーネントの起動時の例とは若干異なる場合があります。

```
# pcs cluster status
Cluster Status:
 Stack: corosync
 Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
 Last updated: Thu Oct 11 16:11:18 2018
 Last change: Thu Oct 11 16:11:00 2018 by hacluster via crmd on z1.example.com
 2 nodes configured
 0 resources configured

PCSD Status:
  z1.example.com: Online
  z2.example.com: Online
```

7. 両方のノードに Web ブラウザーを設定し、Web ページを作成して簡単なテキストメッセージを表示します。**firewalld** デーモンを実行している場合は、**httpd** で必要なポートを有効にします。

注記

systemctl enable では、システムの起動時に、クラスターが管理するサービスを有効にすることはできません。

```
# yum install -y httpd wget
...
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload

# cat <<-END >/var/www/html/index.html
<html>
<body>My Test Site - $(hostname)</body>
</html>
END
```

Apache リソースエージェントが、クラスター内の各ノードで Apache のステータスを取得できるようにするため、既存の設定に以下の内容を追加して、ステータスサーバーの URL を有効にします。

```
# cat <<-END > /etc/httpd/conf.d/status.conf
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
Allow from 127.0.0.1
Allow from ::1
</Location>
END
```

8. クラスターが管理する **IPaddr2** および **apache** を作成します。「IPaddr2」リソースはフローティング IP であるため、物理ノードに関連付けられている IP アドレスは使用できません。「IPaddr2」リソースの NIC デバイスを指定しない場合は、そのノードで使用される、静的に割り当てられた IP アドレスと同じネットワークにフローティング IP が存在する必要があります。
利用可能なリソースタイプの一覧を表示するには、**pcs resource list** コマンドを使用します。指定したリソースタイプに設定できるパラメーターを表示するには、**pcs resource describe resourcetype** コマンドを使用します。たとえば、以下のコマンドは、**apache** というタイプのリソースに設定できるパラメーターを表示します。

```
# pcs resource describe apache
...
```

この例では、IP アドレスリソースおよび apache リソースの両方が、**apachegroup** という名前のグループに含まれるように設定します。これにより、両リソースが一緒に保存され、同じノードで実行できます。

クラスター内のいずれかのノードで、以下のコマンドを実行します。

```
# pcs resource create ClusterIP ocf:heartbeat:IPaddr2 ip=192.168.122.120 --group
apachegroup

# pcs resource create WebSite ocf:heartbeat:apache
configfile=/etc/httpd/conf/httpd.conf statusurl="http://localhost/server-status" --group
```

```
apachegroup

# pcs status
Cluster name: my_cluster
Stack: corosync
Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
Last updated: Fri Oct 12 09:54:33 2018
Last change: Fri Oct 12 09:54:30 2018 by root via cibadmin on z1.example.com

2 nodes configured
2 resources configured

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

Full list of resources:

Resource Group: apachegroup
    ClusterIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
    WebSite    (ocf::heartbeat:apache):        Started z1.example.com

PCSD Status:
  z1.example.com: Online
  z2.example.com: Online
...
```

このインスタンスでは、**apachegroup** サービスが z1.example.com ノードで実行していることに注意してください。

9. 作成した Web サイトにアクセスし、サービスを実行しているノードでそのサービスを停止し、2 番目のノードにサービスがフェイルオーバーする方法を確認してください。

   a. ブラウザーで、設定済みのフローティング IP アドレスを使用して作成した Web サイトを開くように指定します。定義したテキストメッセージが表示され、Web サイトを実行しているノードの名前が表示されるはずです。

   b. Apache Web サービスを停止します。**killall -9** を使用して、アプリケーションレベルのクラッシュをシミュレートします。

      ```
      # killall -9 httpd
      ```

      クラスターのステータスを確認します。Web サービスを停止したためにアクションが失敗したものの、サービスが実行していたノードでクラスターソフトウェアがサービスを再起動し、Web サイトに引き続きアクセスできることを確認できるはずです。

      ```
      # pcs status
      Cluster name: my_cluster
      Stack: corosync
      Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
      Last updated: Fri Oct 12 09:54:33 2018
      Last change: Fri Oct 12 09:54:30 2018 by root via cibadmin on z1.example.com

      2 nodes configured
      2 resources configured

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

```
Full list of resources:

Resource Group: apachegroup
    ClusterIP  (ocf::heartbeat:IPaddr2):       Started z1.example.com
    WebSite    (ocf::heartbeat:apache):        Started z1.example.com

Failed Resource Actions:
* WebSite_monitor_60000 on z1.example.com 'not running' (7): call=31,
status=complete, exitreason='none',
    last-rc-change='Fri Feb  5 21:01:41 2016', queued=0ms, exec=0ms
```

サービスが再開したら、障害 (failure) ステータスをクリアします。

```
# pcs resource cleanup WebSite
```

c. サービスを実行しているノードをスタンバイモードにします。フェンシングは無効にしているため、ノードレベルの障害（電源ケーブルを引き抜くなど）を効果的にシミュレートできません。クラスターがこのような状態から復旧するにはフェンシングが必要になるためです。

```
# pcs node standby z1.example.com
```

d. クラスターのステータスを確認し、サービスを実行している場所をメモします。

```
# pcs status
Cluster name: my_cluster
Stack: corosync
Current DC: z1.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
Last updated: Fri Oct 12 09:54:33 2018
Last change: Fri Oct 12 09:54:30 2018 by root via cibadmin on z1.example.com

2 nodes configured
2 resources configured

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

Full list of resources:

Resource Group: apachegroup
    ClusterIP  (ocf::heartbeat:IPaddr2):       Started z2.example.com
    WebSite    (ocf::heartbeat:apache):        Started z2.example.com
```

e. Web サイトにアクセスします。表示メッセージは、現在サービスを実行しているノードを示しますが、サービスの切断はありません。

10. クラスターサービスを最初のノードに復元するには、そのノードをスタンドバイモードから回復します。ただし、必ずしもそのサービスが最初のノードに戻るわけではありません。

```
# pcs cluster unstandby z1.example.com
```

11. 最終的なクリーンナップを行うために、両方のノードでクラスターサービスを停止します。

```
# pcs cluster stop --all
```

第3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を使用すると、**corosync**、**pacemaker**、**booth**、**sbd** などのクラスターサービスを制御し、設定を簡単に行うことができます。

cib.xml 設定ファイルは直接編集しないでください。ほとんどの場合、Pacemaker は、直接編集した **cib.xml** ファイルを受け付けません。

3.1. PCS HELP DISPLAY

pcs コマンドで **-h** オプションを使用すると、**pcs** コマンドのパラメーターと、その説明が表示されます。たとえば、以下のコマンドは、**pcs resource** コマンドのパラメーターを表示します。出力の一部だけが表示されます。

```
# pcs resource -h
```

3.2. RAW 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せず、**pcs cluster cib** コマンドを使用して、raw クラスター設定を表示します。

pcs cluster cib filename コマンドで、指定したファイルに、raw クラスター設定を保存できます。クラスターを事前に設定している場合はすでにアクティブな CIB が存在しますが、以下のコマンドを実行して、raw の xml ファイルを保存します。

```
pcs cluster cib filename
```

たとえば、次のコマンドを使用すると、**testfile** という名前のファイルに CIB の raw xml が保存されます。

```
pcs cluster cib testfile
```

3.3. 作業ファイルへの設定変更の保存

クラスターを設定すると、アクティブな CIB に影響を及ぼさずに、指定したファイルに設定変更を保存できます。これにより、個々の更新を使用して実行中のクラスター設定を直ちに更新することなく、設定の更新を指定できます。

CIB をファイルに保存する方法は、「raw クラスター設定の表示」 を参照してください。そのファイルを作成したら、**pcs** コマンドの **-f** オプションを使用したアクティブな CIB ではなく、ファイルに設定変更を保存できます。変更を完了し、アクティブな CIB ファイルへの更新が用意できたら、**pcs cluster cib-push** コマンドでファイルの更新をプッシュできます。

以下は、CIB のファイルに変更をプッシュするのに推奨される手順です。この手順は、元の保存された CIB ファイルのコピーを作成し、そのコピーを変更します。アクティブな CIB にその変更をプッシュする場合、この手順は、**pcs cluster cib-push** コマンドの **diff-against** オプションを指定して、元のファイルと、更新されたファイルとの間の変更だけが CIB にプッシュされるようにします。これにより、ユーザーが互いを上書きしないように、並列に変更を加えることができます。ここでは、構成ファイル全体を解析する必要はないため、Pacemaker への負荷が減ります。

1. ファイルへのアクティブな CIB を保存します。この例では、**original.xml** という名前のファイルに CIB が保存されます。

```
# pcs cluster cib original.xml
```

2. 設定の更新に使用する作業ファイルに、保存したファイルをコピーします。

```
# cp original.xml updated.xml
```

3. 必要に応じて設定を更新します。以下のコマンドは、**updated.xml** ファイルにリソースを作成しますが、現在実行しているクラスター設定にはそのリソースを追加しません。

```
# pcs -f updated.xml resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.120
op monitor interval=30s
```

4. アクティブな CIB に更新されたファイルをプッシュします。元のファイルに加えた変更のみをプッシュすることを指定します。

```
# pcs cluster cib-push updated.xml diff-against=original.xml
```

もしくは、次のコマンドを使用して、CIB ファイルの現在のコンテンツ全体をプッシュできます。

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

CIB ファイル全体をプッシュすると、Pacemaker はバージョンを確認し、クラスターにあるものよりも古い場合は CIB ファイルをプッシュしません。クラスターにあるものよりも古いバージョンで CIB ファイル全体を更新する必要がある場合は、**pcs cluster cib-push** コマンドの **--config** オプションを使用します。

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

3.4. クラスターのステータス表示

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ステータスを表示します。

```
pcs status
```

pcs status コマンドの commands パラメーター (**resources**、**cluster**、**nodes**、または **pcsd**) を指定すると、特定のクラスターコンポーネントのステータスを表示できます。

```
pcs status commands
```

たとえば、次のコマンドは、クラスターリソースのステータスを表示します。

```
pcs status resources
```

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

```
pcs cluster status
```

3.5. クラスターの全設定の表示

現在のクラスター設定をすべて表示する場合は、次のコマンドを実行します。

```
pcs config
```

第4章 PACEMAKER を使用した RED HAT HIGH AVAILABILITY クラスターの作成

以下の手順では、**pcs** を使用して Red Hat High Availability 2 ノードクラスターを作成します。クラスターを作成したら、必要なリソースおよびリソースグループを設定できます。

この例では、クラスターを設定するために、システムに以下のコンポーネントを追加する必要があります。

- クラスターを作成するのに使用する 2 つのノード。この例では、使用されるノードは **z1.example.com** および **z2.example.com** です。
- プライベートネットワーク用のネットワークスイッチ。クラスターノード同士の通信、およびその他のクラスターハードウェア（ネットワーク電源スイッチやファイバーチャネルスイッチなど）との通信にプライベートネットワークを使用することが推奨されますが、必須ではありません。
- クラスターの各ノード用のフェンスデバイス。この例では、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** です。

4.1. クラスターソフトウェアのインストール

以下は、クラスターのインストールおよび設定の手順です。

1. クラスターの各ノードに、Red Hat High Availability Add-On ソフトウェアパッケージと使用可能なすべての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 pacemaker fence-agents-all
```

   または、以下のコマンドを実行して、Red Hat High Availability Add-On ソフトウェアパッケージと必要なフェンスエージェントのみをインストールすることもできます。

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

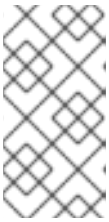
   以下のコマンドは、利用できるフェンスエージェントの一覧を表示します。

```
# rpm -q -a | grep fence
fence-agents-rhevm-4.0.2-3.el7.x86_64
fence-agents-ilo-mp-4.0.2-3.el7.x86_64
fence-agents-ipmilan-4.0.2-3.el7.x86_64
...
```

警告

Red Hat High Availability Add-On パッケージをインストールしたら、自動的に何もインストールされないように、ソフトウェア更新設定を行う必要があります。実行中のクラスターへのインストールで、予期しない動作が発生する可能性があります。クラスターへのソフトウェア更新を実行する方法は「RHEL 高可用性またはレジリエントストレージクラスターにソフトウェアアップデートを適用するのに推奨されるプラクティス」を参照してください。

2. **firewalld** デーモンを実行している場合は、以下のコマンドを実行して、Red Hat High Availability Add-On に必要なポートを有効にします。



注記

firewalld デーモンがシステムにインストールされているかどうかを確認するには、**rpm -q firewalld** コマンドを実行します。**firewalld** デーモンをインストールしている場合は、**firewall-cmd --state** コマンドを使用して、実行しているかどうかを確認できます。

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --add-service=high-availability
```

注記

クラスターコンポーネントの理想的なファイアウォールの設定は、ローカル環境に応じて異なります。ローカル環境では、ノードに複数のネットワークインターフェースがあるかどうか、またはオフホストのファイアウォールがあるかどうかを考慮する必要があります。この例では、 通常 Pacemaker クラスターで必要となるポートを開きますが、これはローカルの条件に合わせて変更する必要があります。「High Availability Add-On のポートの有効化」 では、Red Hat High Availability Add-On に対して有効にする各ポートと、その目的を説明しています。

3. **pcs** を使用し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する必要があります。**hacluster** ユーザーのパスワードは、各ノードで同じにすることが推奨されます。

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

4. クラスターを設定する前に、各ノードでブート時に起動するように、**pcsd** デーモンを起動して有効にしておく必要があります。このデーモンは、**pcs** コマンドで動作し、クラスターのノード全体で設定を管理します。
クラスターの各ノードで、次のコマンドを実行して **pcsd** サービスが起動し、システムの起動時に **pcsd** が有効になるよう設定します。

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

5. **pcs** を実行するノードで、クラスター内の各ノードに対して、**pcs** ユーザー **hacluster** を認証します。
以下のコマンドでは、**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、**z1.example.com** の **hacluster** ユーザーの認証を行います。

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

4.2. 高可用性クラスターの作成

この手順では、**z1.example.com** ノードおよび **z2.example.com** ノードで構成される Red Hat High Availability Add-On を作成します。

1. **z1.example.com** で以下のコマンドを実行し、2 つのノード **z1.example.com** と **z2.example.com** で構成される 2 ノードクラスターの **my_cluster** 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両方のノードに伝搬されます。このコマンドには **--start** オプションが含まれます。このオプションを使用すると、クラスター内の両方のノードでクラスターサービスが起動します。

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

2. クラスターサービスを有効にし、ノードの起動時にクラスターの各ノードでクラスターサービスが実行するようにします。

注記

使用している環境でクラスターサービスを無効にしておきたい場合などは、この手順を省略できます。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、ノードで **pcs cluster start** コマンドを実行し、サービスを手動で起動する必要があります。

```
[root@z1 ~]# 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ステータスを表示できます。**pcs cluster setup** コマンドで **--start** オプションを使用してクラスターサービスを起動した場合は、クラスターが稼働するまでに少し時間がかかる可能性があるため、クラスターとその設定で後続の動作を実行する前に、クラスターが稼働していることを確認する必要があります。

```
[root@z1 ~]# pcs cluster status
Cluster Status:
 Stack: corosync
 Current DC: z2.example.com (version 2.0.0-10.el8-b67d8d0de9) - partition with quorum
 Last updated: Thu Oct 11 16:11:18 2018
 Last change: Thu Oct 11 16:11:00 2018 by hacluster via crmd on z2.example.com
```

```
 2 Nodes configured
 0 Resources configured

 ...
```

4.3. 複数のリンクを使用した高可用性クラスターの作成

pcs cluster setup コマンドを使用して、各ノードにリンクをすべて指定することで、複数のリンクを持つ Red Hat High Availability クラスターを作成できます。

2 つのリンクを持つ 2 ノードクラスターを作成するコマンドの形式は、以下のとおりです。

```
pcs cluster setup cluster_name node1_name addr=node1_link0_address
addr=node1_link1_address node2_name addr=node2_link0_address
addr=node2_link1_address
```

複数のリンクを持つクラスタを作成する場合は、次に示す内容を検討してください。

- **addr=address** パラメーターの順番は重要です。ノード名の後に指定する最初のアドレスは **link0** 用で、2 番目のアドレスは **link1** 用となります。
- デフォルトのトランスポートプロトコルである knet トランスポートプロトコルを使用して、リンクを 8 つまで指定できます。
- **addr=** パラメーターの数は、すべてのノードで同じでなければなりません。
- 現時点では、**pcs** コマンドを使用して既存のクラスターにリンクを追加、削除、または変更することはできません。
- シングルリンククラスターでは、片方のリンクは IPv4 で実行し、もう片方のリンクが IPv6 で実行していたとしても、1 つのリンクに IPv4 アドレスと IPv6 アドレスを混在させないでください。
- シングルリンククラスターでは、IPv4 アドレスおよび IPv6 アドレスを 1 つのリンクに混在させて、名前が IPv4 アドレスまたは IPv6 アドレスを解決できなことがない限り、アドレスを IP または名前として指定できます。

この例では、名前が **my_twolink_cluster** で、**rh80-node1** と **rh80-node2** の 2 つのノードを持つ 2 ノードクラスターを作成します。**rh80-node1** には、IP アドレス 192.168.122.201 が **link0**、192.168.123.201 が **link1** の、2 つのインターフェースがあります。**rh80-node2** には、IP アドレス 192.168.122.202 が **link0**、192.168.123.202 が **link1** の、2 つのインターフェースがあります。

```
# pcs cluster setup my_twolink_cluster rh80-node1 addr=192.168.122.201
addr=192.168.123.201 rh80-node2 addr=192.168.122.202 addr=192.168.123.202
```

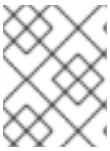
複数のリンクを持つクラスターにノードを追加する場合は、すべてのリンクにアドレスを指定する必要があります。次の例では、ノード **rh80-node3** をクラスターに追加し、IP アドレス 192.168.122.203 を link0 に、192.168.123.203 を link1 に指定します。

```
# pcs cluster node add rh80-node3 addr=192.168.122.203 addr=192.168.123.203
```

4.4. フェンシングの設定

クラスターの各ノードにフェンスデバイスを設定する必要があります。フェンスデバイスの設定コマンドおよびオプションに関する情報は「Red Hat High Availability クラスターでのフェンシングの設定」 を参照してください。

フェンシングの概要と、Red Hat High Availability クラスターにおけるフェンシングの重要性は「RHEL 高可用性クラスターでフェンシングが重要なのはなぜですか?」を参照してください。



注記

フェンスデバイスを設定する際に、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、ホスト名が **zapc.example.com** の APC 電源スイッチを使用してノードをフェンシングし、**fence_apc_snmp** フェンスエージェントを使用します。どちらのノードも同じフェンスエージェントでフェンシングされるため、**pcmk_host_map** オプションと **pcmk_host_list** オプションを使用して、両方のフェンスデバイスを 1 つのリソースとして設定できます。

pcs stonith create コマンドを使用して、**stonith** リソースとしてデバイスを設定し、フェンスデバイスを作成します。以下のコマンドは、**z1.example.com** ノードおよび **z2.example.com** ノードの **fence_apc_snmp** フェンスエージェントを使用する、**myapc** という名前の **stonith** リソースを設定します。**pcmk_host_map** オプションは、**z1.example.com** をポート 1 にマップし、**z2.example.com** をポート 2 にマップします。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。デフォルトでは、このデバイスは各ノードに対して、60 秒間隔で監視を行います。

ノードのホスト名を指定する際に IP アドレスを使用できることに注意してください。

```
[root@z1 ~]# pcs stonith create myapc fence_apc_snmp \
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"
```

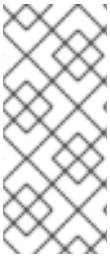
以下のコマンドは、既存の STONITH デバイスのパラメーターを表示します。

```
[root@rh7-1 ~]# pcs stonith config 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)
```

フェンスデバイスの設定後に、デバイスをテストする必要があります。フェンスデバイスのテストに関する情報は「フェンスデバイスのテスト」を参照してください。

注記

ネットワークインターフェースを無効にしてフェンスデバイスのテストを実行しないでください。フェンシングが適切にテストされなくなるためです。



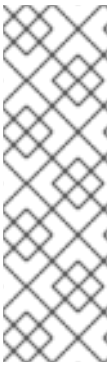
注記

フェンシングが設定され、クラスターが起動されると、ネットワークの再起動により、タイムアウトを超えない場合でもネットワークを再起動するノードのフェンシングがトリガーされます。このため、クラスターサービスの実行中にネットワークサービスを再起動しないでください。ノードで意図しないフェンシングがトリガーされてしまうためです。

4.5. クラスター設定のバックアップおよび復元

次のコマンドを使用して、クラスター設定のバックアップを tarball に作成できます。ファイル名を指定しないと、標準出力が使用されます。

```
pcs config backup filename
```



注記

pcs config backup コマンドは、CIB に設定したようにクラスターの設定だけをバックアップします。リソースデーモンの設定は、このコマンドに含まれません。たとえば、クラスターで Apache リソースを設定すると、(CIB にある) リソース設定のバックアップが作成されますが、(`/etc/httpd` に設定したとおり) Apache デーモン設定と、そこで使用されるファイルのバックアップは作成されません。同様に、クラスターに設定されているデータベースリソースがある場合は、データベースそのものはバックアップが作成されません。ただし、データベースのリソース設定のバックアップ (CIB) は作成されます。

以下のコマンドを使用して、バックアップからすべてのノードのクラスター設定ファイルを復元します。ファイル名を指定しないと、標準入力が使用されます。**--local** オプションは、現在のノードにあるファイルだけを復元します。

```
pcs config restore [--local] [filename]
```

4.6. HIGH AVAILABILITY ADD-ON のポートの有効化

クラスターコンポーネントの理想的なファイアウォール設定は、ローカル環境によって異なります。ここでは、ノードに複数のネットワークインターフェースがあるかどうか、またはオフホストのファイアウォールがあるかどうかを考慮する必要があります。

firewalld デーモンを実行している場合は、以下のコマンドを実行して Red Hat High Availability Add-On が必要とするポートを有効にします。ローカルの状況に合わせて開くポートを変更することが必要になる場合があります。

注記

firewalld デーモンがシステムにインストールされているかどうかを確認するには、**rpm -q firewalld** コマンドを実行します。**firewalld** デーモンをインストールしている場合は、**firewall-cmd --state** コマンドを使用して、実行しているかどうかを確認できます。

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --add-service=high-availability
```

表4.1「High Availability Add-On で有効にするポート」 は、Red Hat High Availability Add-On で有効にするポートを示し、ポートの使用目的を説明しています。

表4.1 High Availability Add-On で有効にするポート

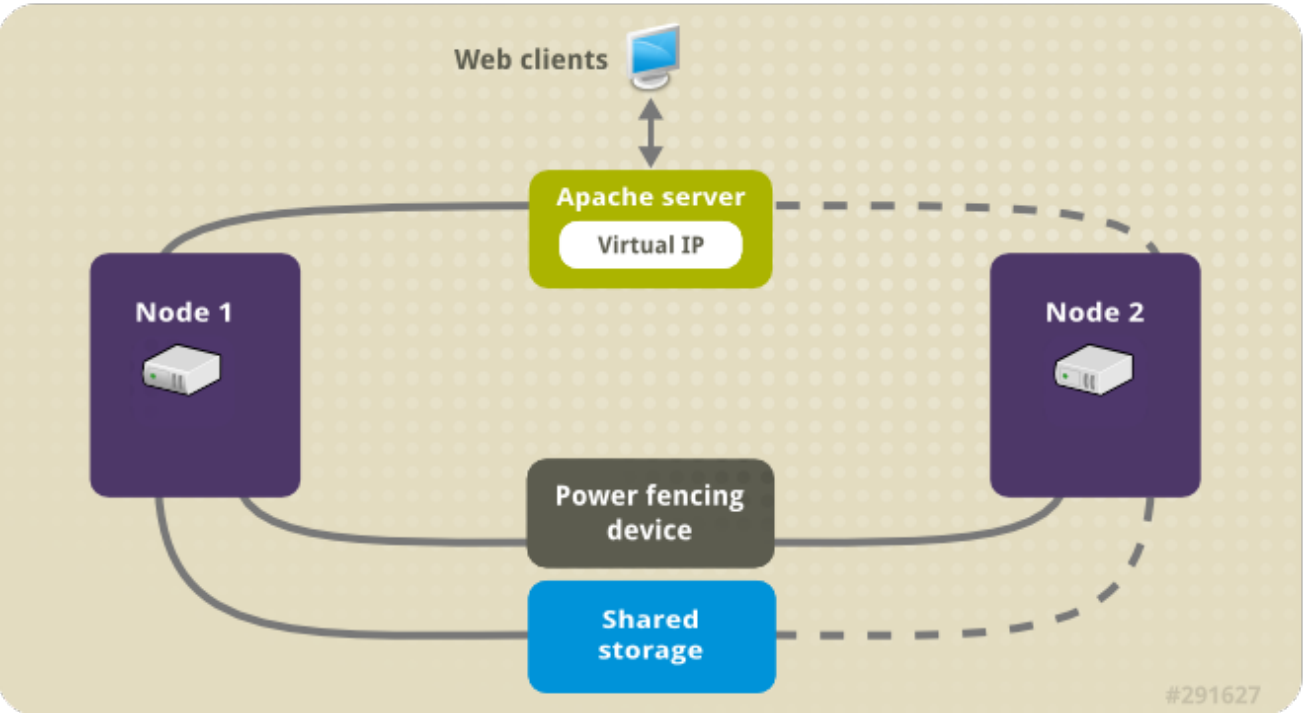
ポート	必要になる場合
TCP 2224	すべてのノードに必要なデフォルトの **pcsd** ポート (pcsd Web UI およびノード間通信に必要)。**/etc/sysconfig/pcsd** ファイルの **PCSD_PORT** パラメーターを使用して **pcsd** を設定できます。 任意のノードの **pcs** が、それ自体も含め、クラスター内のすべてのノードに通信できる方法でポート 2224 を開くことは重要です。Booth クラスターチケットマネージャーまたはクォーラムデバイスを使用する場合は、Booth Arbiter、またはクォーラムデバイスなどのすべての関連ホストで、ポート 2224 を開く必要があります。
TCP 3121	クラスターに Pacemaker リモートノードがある場合に、すべてのノードで必須です。 完全なクラスターノードの Pacemaker の **pacemaker-based** デーモンは、ポート 3121 で Pacemaker リモートノードの **pacemaker_remoted** デーモンに通信できます。クラスターの通信に別個のインターフェースが使用される場合、ポートは単にそのインターフェースで開いておく必要があります。最低限でも、ポートは完全なクラスターノードに対して Pacemaker リモートノードで開いている必要があります。ユーザーは完全なノードとリモートノード間でホストを変換する可能性があるか、またはホストのネットワークを使用してコンテナー内でリモートノードを実行する可能性があるため、すべてのノードに対してポートを開くことは役に立ちます。ノード以外のホストに対してポートを開く必要はありません。
TCP 5403	**corosync-qnetd** で、クォーラムデバイスを使用する場合にクォーラムデバイスで必須。デフォルト値は、**corosync-qnetd** コマンドの **-p** オプションを使用して変更できます。
UDP 5404-5412	ノード間の通信を容易にするために corosync ノードで必須です。任意のノードの **corosync** が、それ自体も含め、すべてのノードと通信できるようにポート 5404-5412 を開くことが重要です。
TCP 21064	クラスターに DLM を必要とするリソースが含まれる場合にすべてのノードで必須です (例: **GFS2**)。

第5章 RED HAT HIGH AVAILABILITY クラスターでのアクティブ/パッシブ APACHE HTTP サーバーの設定

以下の手順では、**pcs** を使用して、2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターでアクティブ/パッシブな Apache HTTP サーバーを設定し、クラスターリソースを設定します。このユースケースでは、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 HTTP 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 つのノードの 1 つで実行されます。Web サーバーが実行しているノードが正常に動作しなくなると、Web サーバーはクラスターの 2 つ目のノードで再起動し、サービスの中断は最小限に抑えられます。

図5.1「2 ノード Red Hat High Availability クラスターの Apache」 は、クラスターのおおまかな概要を示しています。クラスターはネットワーク電源スイッチおよび共有ストレージと共に設定される 2 ノードの Red Hat High Availability クラスターです。クライアントは仮想 IP を使用して Apache HTTP サーバーにアクセスするため、クラスターノードはパブリックネットワークに接続されます。Apache サーバーは、ノード 1 またはノード 2 のいずれかで実行されます。いずれのノードも、Apache のデータが保持されるストレージにアクセスできます。

図5.1 2 ノード Red Hat High Availability クラスターの Apache



このユースケースでは、システムに以下のコンポーネントが必要です。

- 各ノードに電源フェンスが設定されている 2 ノードの Red Hat High Availability クラスター。プライベートネットワークが推奨されますが、必須ではありません。この手順では「Pacemaker を使用した Red Hat High Availability クラスターの作成」で説明されているサンプルのクラスターを使用します。
- Apache に必要なパブリック仮想 IP アドレス。
- iSCSI またはファイバーチャネルを使用する、クラスターのノードに対する共有ストレージ。

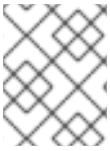
Web サーバーで必要とされる LVM リソース、ファイルシステムリソース、IP アドレスリソース、Web サーバーリソースなどのクラスターコンポーネントを含む Apache リソースグループで、クラスターが設定されます。このリソースグループは、クラスター内の 1 つのノードから別のノードへのフェールオーバーが可能なため、いずれのノードでも Web サーバーを実行できます。このクラスターのリソースグループを作成する前に、以下の手順を実行します。

1. 論理ボリューム **my_lv** に、 **ext4** ファイルシステムを設定します。
2. Web サーバーを設定します。

上記の手順をすべて完了したら、リソースグループと、そのグループに追加するリソースを作成します。

5.1. PACEMAKER クラスターで EXT4 ファイルシステムを持つ LVM ボリュームの設定

このユースケースでは、クラスターのノード間で共有されるストレージに、LVM 論理ボリュームを作成する必要があります。



注記

LVM ボリュームと、クラスターノードで使用するパーティションおよびデバイスは、クラスターノード以外には接続しないでください。

以下の手順では、LVM 論理ボリュームを作成してから Pacemaker クラスターで使用できるように、ボリュームに **ext4** ファイルシステムを作成します。この例では、LVM 論理ボリュームの作成元である LVM 物理ボリュームを保管するために、共有パーティション **/dev/sdb1** が使用されます。

1. クラスターの両方のノードで以下のステップを実行し、LVM システム ID の値を、システムの **uname** の値に設定します。LVM システム ID を使用すると、クラスターのみがボリュームグループをアクティブにできるようにできます。
   a. **/etc/lvm/lvm.conf** 設定ファイルの **system_id_source** 設定オプションを **uname** に設定します。

   ```
   # Configuration option global/system_id_source.
   system_id_source = "uname"
   ```

   b. ノードの LVM システム ID が、ノードの **uname** に一致することを確認します。

   ```
   # lvm systemid
     system ID: z1.example.com
   # uname -n
     z1.example.com
   ```

2. LVM ボリュームを作成し、そのボリュームに **ext4** ファイルシステムを作成します。**/dev/sdb1** パーティションは共有されるストレージであるため、この手順のこの部分は、いずれかのノードで実行してください。
   a. パーティション **/dev/sdb1** に LVM 物理ボリュームを作成します。

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

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

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

c. 新規ボリュームグループには、実行中のノードで、かつボリュームグループの作成元であるノードのシステム ID があることを確認します。

```
# vgs -o+systemid
  VG    #PV #LV #SN Attr   VSize  VFree  System ID
  my_vg   1   0   0 wz--n- <1.82t <1.82t z1.example.com
```

d. ボリュームグループ **my_vg** を使用して、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 コマンドを実行すると、論理ボリュームを表示できます。

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

e. 論理ボリューム **my_lv** に、**ext4** ファイルシステムを作成します。

```
# mkfs.ext4 /dev/my_vg/my_lv
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 462848 1k blocks and 115824 inodes
...
```

5.2. APACHE HTTP サーバーの設定

以下の手順で、Apache HTTP サーバーを設定します。

1. クラスター内の各ノードに、Apache HTTP サーバーがインストールされていることを確認します。Apache HTTP サーバーのステータスをチェックするには、クラスターに **wget** ツールがインストールされている必要があります。
   各ノードで、以下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が Apache HTTP サーバーのステータスを取得できるようにするには、クラスター内の各ノードの **/etc/httpd/conf/httpd.conf** ファイルに、以下のテキストが含まれ、コメントアウトされていないことを確認してください。これが記載されていない場合は、ファイルの末尾に追加します。

```
<Location /server-status>
   SetHandler server-status
   Require local
</Location>
```

3. **apache** リソースエージェントを使用して Apache を管理する場合は **systemd** が使用されません。このため、Apache で提供される **logrotate** スクリプトを編集して、**systemctl** を使用して Apache をリロードしないようにする必要があります。
   クラスター内の各ノードで、**/etc/logrotate.d/httpd** ファイルから以下の行を削除します。

```
/bin/systemctl reload httpd.service > /dev/null 2>/dev/null || true
```

削除した行を以下の行に置き換えます。

```
/usr/sbin/httpd -f /etc/httpd/conf/httpd.conf -c "PidFile /var/run/httpd.pid" -k graceful > /dev/null
2>/dev/null || true
```

4. 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 ボリュームと ext4 ファイルシステムの設定」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で **index.html** ファイルを作成し、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
```

5.3. リソースおよびリソースグループの作成

このユースケースでは、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するよう、このリソースは **apachegroup** というリソースグループに追加します。作成するリソースは、以下の順に開始します。

1. 「LVM ボリュームと ext4 ファイルシステムの設定」で作成した LVM ボリュームグループを使用する、**my_lvm** という名前の **LVM** リソース。
2. 「LVM ボリュームと ext4 ファイルシステムの設定」で作成したファイルシステムデバイス **/dev/my_vg/my_lv** を使用する、**my_fs** という名前の **Filesystem** リソース。
3. **apachegroup** リソースグループのフローティング IP アドレスである **IPaddr2** リソース。物理ノードに関連付けられている IP アドレスは使用できません。**IPaddr2** リソースの NIC デバイスが指定されていない場合は、クラスターノードで使用される、IP アドレス静的に割り当てられたノードのいずれかと同じネットワークにフローティング IP が存在しないと、フローティング IP アドレスを割り当てる NIC デバイスが適切に検出されません。
4. **Website** という名前の **apache** リソースで、**index.html** ファイルと「Apache HTTP サーバーの設定」の手順で定義した Apache 設定を使用します。

以下の手順で、**apachegroup** リソースグループとこのグループに含め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。以下の手順は、クラスター内のいずれかのノードで実行してください。

1. 次のコマンドは、 **LVM が有効な** リソース **my_lvm** を作成します。リソースグループ **apachegroup** は存在しないため、このコマンドによりリソースグループが作成されます。

注記

データ破損のリスクとなるため、アクティブ/パッシブの HA 設定で、同じ LVM ボリュームグループを使用する **LVM が有効な** リソースを 1 つ以上設定しないでください。

```
[root@z1 ~]# pcs resource create my_lvm ocf:heartbeat:LVM-activate vgname=my_vg
vg_access_mode=system_id --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します。以下のコマンドを使用してリソースが作成され、起動していることを確認できます。

```
# pcs resource status
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM-activate): Started
```

pcs resource disable コマンドおよび **pcs resource enable** コマンドを使用すると、各リソースを個別に停止および起動できます。

2. 以下のコマンドでは、設定に必要な残りのリソースを作成し、それらを既存の **apachegroup** リソースグループに追加します。

```
[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. リソースと、そのリソースを含むリソースグループの作成が完了したら、クラスターのステータスを確認します。4 つのリソースがすべて同じノードで実行していることに注意してください。

```
[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
```

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ことに注意してください。

4. クラスターが稼働したら、ブラウザーで、**IPaddr2** リソースとして定義した IP アドレスを指定して、「Hello」と単語が表示されるサンプル表示を確認します。

```
Hello
```

設定したリソースが実行していない場合は、**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ます。

5.4. リソース設定のテスト

「リソースおよびリソースグループの作成」で説明するクラスターのステータス表示では、すべてのリソースが **z1.example.com** ノードで実行します。以下の手順に従い、1 つ目のノードを **スタンバイ** モードにし、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をテストします。1 つ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. 以下のコマンドは、**z1.example.com** ノードを **スタンバイ** モードにします。

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

2. **z1** をスタンバイモードにしたら、クラスターのステータスを確認します。リソースはすべて **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、中断せず表示されているはずです。

3. **スタンバイ** モードから **z1** を削除するには、以下のコマンドを実行します。

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

注記

ノードを **スタンバイ** モードから削除しても、リソースはそのノードにフェイルオーバーしません。

第6章 RED HAT HIGH AVAILABILITY クラスターのアクティブ/パッシブな NFS サーバーの設定

以下の手順では、共有ストレージを使用して、2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターで、高可用性アクティブ/パッシブ NFS サーバーを設定します。この手順では、**pcs** を使用して Pacemaker クラスターリソースを設定します。このユースケースでは、クライアントが、フローティング IP アドレスから NFS ファイルシステムにアクセスします。NFS サービスは、クラスターにある 2 つのノードのいずれかで実行します。NFS サーバーが実行しているノードが正常に動作しなくなると、NFS サーバーはクラスターの 2 つ目のノードで再起動し、サービスの中断は最小限に抑えられます。

このユースケースでは、システムに以下のコンポーネントが必要です。

- 各ノードに電源フェンスが設定されている 2 ノードの Red Hat High Availability クラスター。プライベートネットワークが推奨されますが、必須ではありません。この手順では「Pacemaker を使用した Red Hat High Availability クラスターの作成」で説明されているサンプルのクラスターを使用します。
- NFS サーバーに必要なパブリック仮想 IP アドレス。
- iSCSI またはファイバーチャネルを使用する、クラスターのノードに対する共有ストレージ。

既存の 2 ノード Red Hat Enterprise Linux High Availability クラスターで高可用性アクティブ/パッシブ NFS サーバーを設定するには、以下のステップを実行する必要があります。

1. クラスターのノードに共有ストレージの LVM 論理ボリューム **my_lv** で **ext4** ファイルシステムを設定する。
2. 共有ストレージのLVM 論理ボリュームで NFS 共有を設定する。
3. クラスターリソースを作成する。
4. 設定した NFS サーバーをテストする。

6.1. PACEMAKER クラスターで EXT4 ファイルシステムを持つ LVM ボリュームの設定

このユースケースでは、クラスターのノード間で共有されるストレージに、LVM 論理ボリュームを作成する必要があります。

注記

LVM ボリュームと、クラスターノードで使用するパーティションおよびデバイスは、クラスターノード以外には接続しないでください。

以下の手順では、LVM 論理ボリュームを作成してから Pacemaker クラスターで使用できるように、ボリュームに **ext4** ファイルシステムを作成します。この例では、LVM 論理ボリュームの作成元である LVM 物理ボリュームを保管するために、共有パーティション **/dev/sdb1** が使用されます。

1. クラスターの両方のノードで以下のステップを実行し、LVM システム ID の値を、システムの **uname** の値に設定します。LVM システム ID を使用すると、クラスターのみがボリュームグループをアクティブにできるようにできます。

a. **/etc/lvm/lvm.conf** 設定ファイルの **system_id_source** 設定オプションを **uname** に設定します。

```
# Configuration option global/system_id_source.
system_id_source = "uname"
```

b. ノードの LVM システム ID が、ノードの **uname** に一致することを確認します。

```
# lvm systemid
  system ID: z1.example.com
# uname -n
  z1.example.com
```

2. LVM ボリュームを作成し、そのボリュームに **ext4** ファイルシステムを作成します。**/dev/sdb1** パーティションは共有されるストレージであるため、この手順のこの部分は、いずれかのノードで実行してください。

a. パーティション **/dev/sdb1** に LVM 物理ボリュームを作成します。

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

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

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

c. 新規ボリュームグループには、実行中のノードで、かつボリュームグループの作成元であるノードのシステム ID があることを確認します。

```
# vgs -o+systemid
  VG    #PV #LV #SN Attr   VSize  VFree  System ID
  my_vg   1   0   0 wz--n- <1.82t <1.82t z1.example.com
```

d. ボリュームグループ **my_vg** を使用して、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 コマンドを実行すると、論理ボリュームを表示できます。

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

e. 論理ボリューム **my_lv** に、**ext4** ファイルシステムを作成します。

```
# mkfs.ext4 /dev/my_vg/my_lv
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 462848 1k blocks and 115824 inodes
...
```

6.2. NFS 共有の設定

次の手順では、NFS デーモンのフェールオーバーに、NFS 共有を設定します。

1. クラスターの両方のノードに、**/nfsshare** ディレクトリーを作成します。

```
# mkdir /nfsshare
```

2. クラスター内の 1 ノードで、以下の手順を行います。

   a. 「LVM ボリュームと ext4 ファイルシステムの設定」で作成した **ext4** ファイルシステムを、**/nfsshare** ディレクトリーにマウントします。

```
[root@z1 ~]# mount /dev/my_vg/my_lv /nfsshare
```

   b. **/nfsshare** ディレクトリーに、**exports** ディレクトリーツリーを作成します。

```
[root@z1 ~]# mkdir -p /nfsshare/exports
[root@z1 ~]# mkdir -p /nfsshare/exports/export1
[root@z1 ~]# mkdir -p /nfsshare/exports/export2
```

   c. NFS クライアントがアクセスするファイルを、**exports** ディレクトリーに置きます。この例では、**clientdatafile1** および **clientdatafile2** という名前のテストファイルを作成します。

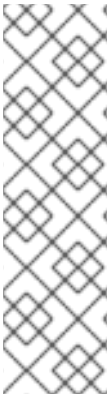
```
[root@z1 ~]# touch /nfsshare/exports/export1/clientdatafile1
[root@z1 ~]# touch /nfsshare/exports/export2/clientdatafile2
```

   d. ext4 ファイルシステムをアンマウントし、LVM ボリュームグループを非アクティブにします。

```
[root@z1 ~]# umount /dev/my_vg/my_lv
[root@z1 ~]# vgchange -an my_vg
```

6.3. クラスター内の NFS サーバーへリソースおよびリソースグループを設定

このセクションでは、このユースケースで、クラスターリソースを設定する手順を説明します。



注記

pcs resource create を使用してクラスターリソースを作成する場合は、作成直後に**pcs status** コマンドを実行して、リソースが実行していることを確認することが推奨されます。クラスターのフェンスデバイスを設定していないと、リソースがデフォルトで起動しません。

設定したリソースが実行していない場合は、**pcs resource debug-start resource** コマンドを実行してリソース設定をテストできます。このコマンドは、クラスターの制御や認識の範囲外でサービスを起動します。設定したリソースが再稼働したら、**pcs resource cleanup resource** を実行してクラスターが更新を認識するようにします。

以下の手順では、システムリソースを設定します。これらのリソースがすべて同じノードで実行するよ

うに、これらのリソースはリソースグループ **nfsgroup** に含まれます。リソースは、グループに追加された順序で起動し、その逆の順序で停止します。この手順は、クラスター内のいずれかのノードで実行してください。

1. 次のコマンドは、LVM が有効なリソース (**my_lvm**) を作成します。リソースグループ **my_lvm** は存在しないため、このコマンドによりリソースグループが作成されます。

注記

データ破損のリスクとなるため、アクティブ/パッシブの HA 設定で、同じ LVM ボリュームグループを使用する **LVM が有効な** リソースを 1 つ以上設定しないでください。

```
[root@z1 ~]# pcs resource create my_lvm ocf:heartbeat:LVM-activate vgname=my_vg
vg_access_mode=system_id --group nfsgroup
```

クラスターのステータスを確認し、リソースが実行していることを確認します。

```
root@z1 ~]#  pcs status
Cluster name: my_cluster
Last updated: Thu Jan  8 11:13:17 2015
Last change: Thu Jan  8 11:13:08 2015
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
3 Resources configured

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

Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com

PCSD Status:
  z1.example.com: Online
  z2.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

2. クラスターの **Filesystem** リソースを設定します。

注記

options=options パラメーターを使用すると、**Filesystem** リソースのリソース設定にマウントオプションを指定できます。完全な設定オプションを表示するには、**pcs resource describe Filesystem** コマンドを実行します。

以下のコマンドは、**nfsshare** という名前の ext4 **Filesystem** リソースを **nfsgroup** リソースグループに追加します。このファイルシステムは、「LVM ボリュームと ext4 ファイルシステム

の設定」で作成した、LVM ボリュームグループと ext4 ファイルシステムを使用します。このファイルシステムは、「NFS 共有の設定」で作成した **/nfsshare** ディレクトリーにマウントされます。

```
[root@z1 ~]# pcs resource create nfsshare Filesystem \
device=/dev/my_vg/my_lv directory=/nfsshare \
fstype=ext4 --group nfsgroup
```

my_lvm リソースおよび **nfsshare** リソースが実行していることを確認します。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
...
```

3. **nfsgroup** リソースグループに、**nfs-daemon** という名前の **nfsserver** リソースを作成します。

注記

nfsserver リソースを使用して、**nfs_shared_infodir** パラメーターを指定できます。これは、NFS デーモンが、NFS 関連のステートフル情報を保管するのに使用するディレクトリーです。この属性は、このエクスポートのコレクションで作成した **Filesystem** リソースのいずれかのサブディレクトリーに設定することが推奨されます。これにより、NFS デーモンは、このリソースグループを再度移動する必要が生じた場合に、別のノードで利用可能になるステートフル情報をデバイスに保存します。この例では、**/nfsshare** は **Filesystem** リソースで管理される共有ストレージディレクトリーで、**/nfsshare/exports/export1** および **/nfsshare/exports/export2** はエクスポートディレクトリーです。**/nfsshare/nfsinfo** は、**nfsserver** リソースの共有情報ディレクトリーです。

```
[root@z1 ~]# pcs resource create nfs-daemon nfsserver \
nfs_shared_infodir=/nfsshare/nfsinfo nfs_no_notify=true \
--group nfsgroup
[root@z1 ~]# pcs status
...
```

4. **exportfs** リソースを追加して、**/nfsshare/exports** ディレクトリーをエクスポートします。このリソースは、**nfsgroup** リソースグループに含まれます。これにより、NFSv4 クライアントの仮想ディレクトリーが構築されます。このエクスポートには、NFSv3 クライアントもアクセスできます。

```
[root@z1 ~]# pcs resource create nfs-root exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash \
directory=/nfsshare/exports \
fsid=0 --group nfsgroup

[root@z1 ~]# # pcs resource create nfs-export1 exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash directory=/nfsshare/exports/export1 \
```

```
fsid=1 --group nfsgroup

[root@z1 ~]# # pcs resource create nfs-export2 exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash directory=/nfsshare/exports/export2 \
fsid=2 --group nfsgroup
```

5. NFS 共有にアクセスするために、NFS クライアントが使用するフローティング IP アドレスリソースを追加します。指定するフローティング IP アドレスには DNS リバースルックアップが必要になりますが、クラスターの全ノードの **/etc/hosts** に、フローティング IP アドレスを指定することもできます。このリソースは、**nfsgroup** リソースグループに含まれます。このデプロイ例では、192.168.122.200 をフローティング IP アドレスとして使用します。

```
[root@z1 ~]# pcs resource create nfs_ip IPaddr2 \
ip=192.168.122.200 cidr_netmask=24 --group nfsgroup
```

6. NFS デプロイメント全体が初期化されたら、NFSv3 の再起動通知を送信する **nfsnotify** リソースを追加します。このリソースは、リソースグループ **nfsgroup** に含まれます。

注記

NFS の通知が適切に処理されるようにするには、フローティング IP アドレスにホスト名が関連付けられおり、それが NFS サーバーと NFS クライアントで同じである必要があります。

```
[root@z1 ~]# pcs resource create nfs-notify nfsnotify \
source_host=192.168.122.200 --group nfsgroup
```

リソースとリソースの制約を作成したら、クラスターのステータスをチェックできます。すべてのリソースが同じノードで実行していることに注意してください。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z1.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z1.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z1.example.com
...
```

6.4. NFS リソース設定のテスト

以下の手順を使用すると、システムの設定を検証できます。NFSv3 または NFSv4 のいずれかで、エクスポートされたファイルシステムをマウントできるはずです。

1. デプロイメントと同じネットワークにあるクラスター外部のノードで NFS 共有をマウントすると、NFS 共有が表示されることを確認します。この例では、192.168.122.0/24 ネットワークを使用します。

```
# showmount -e 192.168.122.200
Export list for 192.168.122.200:
/nfsshare/exports/export1 192.168.122.0/255.255.255.0
/nfsshare/exports         192.168.122.0/255.255.255.0
/nfsshare/exports/export2 192.168.122.0/255.255.255.0
```

2. NFSv4 で NFS 共有をマウントできることを確認するには、クライアントノードのディレクトリーに NFS 共有をマウントします。マウントしたら、エクスポートディレクトリーの内容が表示されることを確認します。テスト後に共有をアンマウントします。

```
# mkdir nfsshare
# mount -o "vers=4" 192.168.122.200:export1 nfsshare
# ls nfsshare
clientdatafile1
# umount nfsshare
```

3. NFSv3 で NFS 共有をマウントできることを確認します。マウント後、テストファイル **clientdatafile1** が表示されることを確認します。NFSv4 とは異なり、NFSv3 は仮想ファイルシステムを使用しないため、特定のエクスポートをマウントする必要があります。テストが終わったら共有をアンマウントします。

```
# mkdir nfsshare
# mount -o "vers=3" 192.168.122.200:/nfsshare/exports/export2 nfsshare
# ls nfsshare
   clientdatafile2
# umount nfsshare
```

4. フェイルオーバーをテストするには、以下のステップを実行します。

   a. クラスター外のノードで、NFS 共有をマウントし、「NFS 共有の設定」で作成した **clientdatafile1** へのアクセスを確認します。

```
# mkdir nfsshare
# mount -o "vers=4" 192.168.122.200:export1 nfsshare
# ls nfsshare
clientdatafile1
```

   b. クラスター内のノードから、**nfsgroup** を実行しているノードを判別します。この例では、**nfsgroup** が **z1.example.com** で実行しています。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z1.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z1.example.com
```

```
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z1.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z1.example.com
...
```

c. クラスター内のノードから、**nfsgroup** を実行しているノードをスタンバイモードにします。

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

d. **nfsgroup** が、別のクラスターノードで正常に起動することを確認します。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z2.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z2.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z2.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z2.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z2.example.com
...
```

e. NFS 共有をマウントしたクラスターの外部のノードより、この外部ノードが NFS マウント内のテストファイルに引き続きアクセスできることを確認します。

```
# ls nfsshare
clientdatafile1
```

フェイルオーバー時に、クライアントに対するサービスは一時的に失われますが、クライアントはユーザーが介入しなくても回復します。デフォルトでは、NFSv4 を使用するクライアントの場合は、マウントのリカバリーに最大 90 秒かかることがあります。この 90 秒は、起動時にサーバーにより確認される NFSv4 ファイルのリースの猶予期間です。NFSv3 クライアントでは、数秒でマウントへのアクセスが回復します。

f. クラスター内のノードから、最初に **nfsgroup** を実行していたノードをスタンバイモードから削除します。しかし、スタンバイモードから回復しただけでは、クラスターリソースはこのノードに戻りません。

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

第7章 クラスター内の GFS2 ファイルシステム

本セクションは、以下を提供します。

- GFS2 ファイルシステムを含む Pacemaker クラスターを設定する手順
- RHEL 8 クラスターに GFS2 ファイルシステムが含まれている RHEL 7 の論理ボリュームを移行する手順

7.1. クラスターに GFS2 ファイルシステムを設定

この手順では、GFS2 ファイルシステムを含む Pacemaker クラスターをセットアップするのに必要なステップの概要を示します。この例では、3 つの論理ボリュームに 3 つの GFS2 ファイルシステムを作成します。

この手順の前提条件として、すべてのノードで、クラスターソフトウェアをインストールおよび起動し、基本的な 2 ノードクラスターを作成する必要があります。また、クラスターにフェンシングを設定することも必要です。Pacemaker クラスターの作成およびクラスターのフェンシングの設定情報は「Pacemaker を使用した Red Hat High Availability クラスターの作成」 を参照してください。

1. クラスターの両方のノードで、**lvm2-lockd** パッケージ、**gfs2-utils** パッケージ、および **dlm** パッケージをインストールします。**lvm2-lockd** パッケージは AppStream チャンネルで提供され、**gfs2-utils** パッケージおよび **dlm** パッケージは、Resilient Storage チャンネルで提供されます。

```
# yum install lvm2-lockd gfs2-utils dlm
```

2. **dlm** リソースをセットアップします。これは、クラスター内で GFS2 ファイルシステムを設定するために必要な依存関係です。この例では、**dlm** リソースを作成し、**locking** という名前のリソースグループに追加します。

```
[root@z1 ~]# pcs resource create dlm --group locking ocf:pacemaker:controld op
monitor interval=30s on-fail=fence
```

3. リソースグループがクラスターの両方のノードでアクティブになるように、**locking** リソースグループのクローンを作成します。

```
[root@z1 ~]# pcs resource clone locking interleave=true
```

4. **lvmlockd** リソースを、**locking** グループに所属するように設定します。

```
[root@z1 ~]# pcs resource create lvmlockd --group locking ocf:heartbeat:lvmlockd op
monitor interval=30s on-fail=fence
```

5. クラスターのステータスを確認し、クラスターの両方のノードで、**locking** リソースグループが起動していることを確認します。

```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...]

Online: [ z1.example.com (1) z2.example.com (2) ]

Full list of resources:
```

```
 smoke-apc      (stonith:fence_apc):    Started z1.example.com
 Clone Set: locking-clone [locking]
     Resource Group: locking:0
         dlm    (ocf::pacemaker:controld):      Started z1.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
     Resource Group: locking:1
         dlm    (ocf::pacemaker:controld):      Started z2.example.com
         lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
     Started: [ z1.example.com z2.example.com ]
```

6. **lvmlockd** デーモンが、クラスターの両方のノードで実行していることを確認します。

```
[root@z1 ~]# ps -ef | grep lvmlockd
root     12257     1  0 17:45 ?        00:00:00 lvmlockd -p /run/lvmlockd.pid -A 1 -g dlm
[root@z2 ~]# ps -ef | grep lvmlockd
root     12270     1  0 17:45 ?        00:00:00 lvmlockd -p /run/lvmlockd.pid -A 1 -g dlm
```

7. クラスターの1つのノードで、2 つの共有ボリュームグループを作成します。一方のボリュームグループには GFS2 ファイルシステムが 2 つ含まれ、もう一方のボリュームグループには GFS2 ファイルシステムが 1 つ含まれます。
以下のコマンドは、共有ボリュームグループ **shared_vg1** を **/dev/vdb** に作成します。

```
[root@z1 ~]# vgcreate --shared shared_vg1 /dev/vdb
  Physical volume "/dev/vdb" successfully created.
  Volume group "shared_vg1" successfully created
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

以下のコマンドは、共有ボリュームグループ **shared_vg2** を **/dev/vdc** に作成します。

```
[root@z1 ~]# vgcreate --shared shared_vg2 /dev/vdc
  Physical volume "/dev/vdc" successfully created.
  Volume group "shared_vg2" successfully created
  VG shared_vg2 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

8. クラスター内の 2 番目のノードで、共有ボリュームグループごとにロックマネージャーを起動します。

```
[root@z2 ~]# vgchange --lock-start shared_vg1
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
[root@z2 ~]# vgchange --lock-start shared_vg2
  VG shared_vg2 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

9. クラスター内の1つのノードで、共有論理ボリュームを作成し、ボリュームを GFS2 ファイルシステムでフォーマットします。必ずクラスター内の各ノードに十分なジャーナルを作成してください。

```
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg1
  Logical volume "shared_lv1" created.
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv2 shared_vg1
```

```
  Logical volume "shared_lv2" created.
[root@z1 ~]# lvcreate --activate sy -L5G -n shared_lv1 shared_vg2
  Logical volume "shared_lv1" created.

[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo1
/dev/shared_vg1/shared_lv1
[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo2
/dev/shared_vg1/shared_lv2
[root@z1 ~]# mkfs.gfs2 -j2 -p lock_dlm -t my_cluster:gfs2-demo3
/dev/shared_vg2/shared_lv1
```

10. すべてのノードで論理ボリュームを自動的にアクティブにするために、各論理ボリュームで **LVM が有効な** リソースを作成します。

    a. ボリュームグループ **shared_vg1** の論理ボリューム **shared_lv1** に、**LVM が有効な** リソース (**sharedlv1**) を作成します。このコマンドは、リソースを含むリソースグループ **shared_vg1** も作成します。この例のリソースグループの名前は、論理ボリュームを含む共有ボリュームグループと同じになります。

    ```
    [root@z1 ~]# pcs resource create sharedlv1 --group shared_vg1 ocf:heartbeat:LVM-
    activate lvname=shared_lv1 vgname=shared_vg1 activation_mode=shared
    vg_access_mode=lvmlockd
    ```

    b. ボリュームグループ **shared_vg1** の論理ボリューム **shared_lv2** に、**LVM が有効な** リソース (**sharedlv2**) を作成します。このリソースは、リソースグループ **shared_vg1** に含まれます。

    ```
    [root@z1 ~]# pcs resource create sharedlv2 --group shared_vg1 ocf:heartbeat:LVM-
    activate lvname=shared_lv2 vgname=shared_vg1 activation_mode=shared
    vg_access_mode=lvmlockd
    ```

    c. ボリュームグループ **shared_vg2** の論理ボリューム **shared_lv1** に、**LVM が有効な** リソース (**sharedlv3**) を作成します。このコマンドは、リソースを含むリソースグループ **shared_vg2** も作成します。

    ```
    [root@z1 ~]# pcs resource create sharedlv3 --group shared_vg2 ocf:heartbeat:LVM-
    activate lvname=shared_lv1 vgname=shared_vg2 activation_mode=shared
    vg_access_mode=lvmlockd
    ```

11. リソースグループのクローンを新たに 2 つ作成します。

    ```
    [root@z1 ~]# pcs resource clone shared_vg1 interleave=true
    [root@z1 ~]# pcs resource clone shared_vg2 interleave=true
    ```

12. **dlm** および **lvmlockd** リソースを含む **locking** リソースグループが最初に起動するように、順序の制約を設定します。

    ```
    [root@z1 ~]# pcs constraint order start locking-clone then shared_vg1-clone
    Adding locking-clone shared_vg1-clone (kind: Mandatory) (Options: first-action=start then-
    action=start)
    [root@z1 ~]# pcs constraint order start locking-clone then shared_vg2-clone
    Adding locking-clone shared_vg2-clone (kind: Mandatory) (Options: first-action=start then-
    action=start)
    ```

13. クラスター内の両方のノードで、論理ボリュームがアクティブであることを確認します。数秒の遅れが生じる可能性があります。

```
[root@z1 ~]# lvs
  LV         VG         Attr       LSize
  shared_lv1 shared_vg1 -wi-a----- 5.00g
  shared_lv2 shared_vg1 -wi-a----- 5.00g
  shared_lv1 shared_vg2 -wi-a----- 5.00g

[root@z2 ~]# lvs
  LV         VG         Attr       LSize
  shared_lv1 shared_vg1 -wi-a----- 5.00g
  shared_lv2 shared_vg1 -wi-a----- 5.00g
  shared_lv1 shared_vg2 -wi-a----- 5.00g
```

14. ファイルシステムリソースを作成し、各 GFS2 ファイルシステムをすべてのノードに自動的にマウントします。
このファイルシステムは Pacemaker のクラスターリソースとして管理されるため、 **/etc/fstab** ファイルには追加しないでください。マウントオプションは、**options=options** を使用して、リソース設定に指定できます。すべての設定オプションを確認する場合は、**pcs resource describe Filesystem** コマンドを実行します。

    以下のコマンドは、ファイルシステムのリソースを作成します。これらのコマンドは各リソースを、そのファイルシステムの論理ボリュームを含むリソースグループに追加します。

```
[root@z1 ~]# pcs resource create sharedfs1 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/shared_vg1/shared_lv1" directory="/mnt/gfs1"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
[root@z1 ~]# pcs resource create sharedfs2 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/shared_vg1/shared_lv2" directory="/mnt/gfs2"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
[root@z1 ~]# pcs resource create sharedfs3 --group shared_vg2
ocf:heartbeat:Filesystem device="/dev/shared_vg2/shared_lv1" directory="/mnt/gfs3"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
```

15. GFS2 ファイルシステムが、クラスターの両方のノードにマウントされていることを確認します。

```
[root@z1 ~]# mount | grep gfs2
/dev/mapper/shared_vg1-shared_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg1-shared_lv2 on /mnt/gfs2 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg2-shared_lv1 on /mnt/gfs3 type gfs2 (rw,noatime,seclabel)

[root@z2 ~]# mount | grep gfs2
/dev/mapper/shared_vg1-shared_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg1-shared_lv2 on /mnt/gfs2 type gfs2 (rw,noatime,seclabel)
/dev/mapper/shared_vg2-shared_lv1 on /mnt/gfs3 type gfs2 (rw,noatime,seclabel)
```

16. クラスターのステータスを確認します。

```
[root@z1 ~]# pcs status --full
Cluster name: my_cluster
[...1

Full list of resources:
```

```
smoke-apc      (stonith:fence_apc):    Started z1.example.com
Clone Set: locking-clone [locking]
    Resource Group: locking:0
        dlm    (ocf::pacemaker:controld):      Started z2.example.com
        lvmlockd       (ocf::heartbeat:lvmlockd):      Started z2.example.com
    Resource Group: locking:1
        dlm    (ocf::pacemaker:controld):      Started z1.example.com
        lvmlockd       (ocf::heartbeat:lvmlockd):      Started z1.example.com
    Started: [ z1.example.com z2.example.com ]
Clone Set: shared_vg1-clone [shared_vg1]
    Resource Group: shared_vg1:0
        sharedlv1      (ocf::heartbeat:LVM-activate):  Started z2.example.com
        sharedlv2      (ocf::heartbeat:LVM-activate):  Started z2.example.com
        sharedfs1      (ocf::heartbeat:Filesystem):    Started z2.example.com
        sharedfs2      (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg1:1
        sharedlv1      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedlv2      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedfs1      (ocf::heartbeat:Filesystem):    Started z1.example.com
        sharedfs2      (ocf::heartbeat:Filesystem):    Started example.co
    Started: [ z1.example.com z2.example.com ]
Clone Set: shared_vg2-clone [shared_vg2]
    Resource Group: shared_vg2:0
        sharedlv3      (ocf::heartbeat:LVM-activate):  Started z2.example.com
        sharedfs3      (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg2:1
        sharedlv3      (ocf::heartbeat:LVM-activate):  Started z1.example.com
        sharedfs3      (ocf::heartbeat:Filesystem):    Started z1.example.com
    Started: [ z1.example.com z2.example.com ]

...
```

7.2. RHEL7 から RHEL8 へ GFS2 ファイルシステムの移行

Red Hat Enterprise Linux 8 では、LVM は、**clvmd** の代わりに LVM ロックデーモン **lvmlockd** を使用して、アクティブ/アクティブクラスターで共有ストレージデバイスを管理します。これにより、アクティブ/アクティブクラスターに共有論理ボリュームとして使用する論理ボリュームを設定する必要があります。また、これにより、**LVM が有効な** リソースを使用して、LVM ボリュームを管理し、**lvmlockd** リソースエージェントを使用して **lvmlockd** デーモンを管理する必要があります。共有論理ボリュームを使用して、GFS2 ファイルシステムを含む Pacemaker クラスターを設定する手順は「クラスターでの GFS2 ファイルシステムの設定」を参照してください。

GFS2 ファイルシステムを含む RHEL8 クラスターを設定する際に、既存の Red Hat Enterprise Linux 7 論理ボリュームを使用するには、RHEL8 クラスターから以下の手順を実行します。この例では、クラスター化された RHEL 7 論理ボリュームが、ボリュームグループ **upgrade_gfs_vg** に含まれます。

注記

既存のファイルシステムを有効にするために、RHEL8 クラスターの名前は、GFS2 ファイルシステムに含まれるRHEL7 クラスターと同じになります。

1. GFS2 ファイルシステムを含む論理ボリュームが現在非アクティブであることを確認してください。すべてのノードがボリュームグループを使用して停止した場合にのみ、この手順は安全です。

2. クラスター内の1つのノードから、強制的にボリュームグループをローカルに変更します。

```
[root@rhel8-01 ~]# vgchange --lock-type none --lock-opt force upgrade_gfs_vg
Forcibly change VG lock type to none? [y/n]: y
  Volume group "upgrade_gfs_vg" successfully changed
```

3. クラスター内の1つのノードから、ローカルボリュームグループを共有ボリュームグループに変更します。

```
[root@rhel8-01 ~]# vgchange --lock-type dlm upgrade_gfs_vg
   Volume group "upgrade_gfs_vg" successfully changed
```

4. クラスター内の各ノードで、ボリュームグループのロックを開始します。

```
[root@rhel8-01 ~]# vgchange --lock-start upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
[root@rhel8-02 ~]# vgchange --lock-start upgrade_gfs_vg
  VG upgrade_gfs_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

この手順を実行すると、各論理ボリュームに、**LVM が有効な** リソースを作成できます。

第8章 RED HAT HIGH AVAILABILITY クラスターでのフェンシングの設定

応答しないノードがデータへのアクセスを続けている可能性があります。データが安全であることを確認するには、STONITH を使用してノードをフェンシングすることが唯一の方法になります。STONITH は「Shoot The Other Node In The Head」の頭字語で、不安定なノードや同時アクセスによるデータの破損を防ぐことができます。STONITH を使用すると、別のノードからデータをアクセスする前に、そのノードが実際にオフラインであることを確認できます。

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。この場合、クラスターが STONITH を使用してノード全体を強制的にオフラインにし、その後サービスを別の場所で開始すると安全です。

フェンシングの概要と、Red Hat High Availability クラスターにおけるフェンシングの重要性は「RHEL 高可用性クラスターでフェンシングが重要なのはなぜですか?」を参照してください。

クラスターのノードにフェンスデバイスを設定して、Pacemaker クラスターに STONITH を実装します。

8.1. 利用可能なフェンスエージェントと、そのオプションの表示

以下のコマンドは、利用可能な STONITH エージェントを一覧表示します。フィルターを指定すると、フィルターに一致する STONITH エージェントのみが表示されます。

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

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

```
pcs stonith describe stonith_agent
```

以下のコマンドでは、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
```

警告

method オプションを提供するフェンスエージェントでは **cycle** がサポートされず、データの破損が生じる可能性があるため、この値は指定できません。

8.2. フェンスデバイスの作成

以下のコマンドは、stonith デバイスを作成します。利用可能なその他のオプションは **pcs stonith -h** の出力を参照してください。

```
pcs stonith create stonith_id stonith_device_type [stonith_device_options] [op
operation_action operation_options
```

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

各ノードのポートを使用して、複数のノードに対して 1 つのフェンスデバイスを使用する場合は、各ノードに個別にデバイスを作成する必要はありません。**pcmk_host_map** オプションを使用すると、ノードとポートのマッピングを定義できます。たとえば次のコマンドは、APC 電源スイッチ **west-apc** を使用し、**west-13** ノードに 15 番ポートを使用するフェンスデバイス **myapc-west-13** を 1 つ作成します。

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

以下の例は、15 番ポートを使用するフェンスノード **west-13**、17 番ポートを使用するフェンスノード **west-14**、18 番ポートを使用するフェンスノード **west-15**、および 19 番ポートを使用するフェンスノード **west-16** に、APC 電源スイッチ **west-apc** を使用します。

```
# 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"
```

フェンスデバイスを設定したら、デバイスをテストして正しく機能していることを確認してください。フェンスデバイスのテストは「フェンスデバイスのテスト」を参照してください。

8.3. フェンスデバイスの一般的なプロパティー

クラスターノードは、フェンスリソースが開始しているかどうかに関わらず、フェンスデバイスでその他のクラスターノードをフェンスできます。以下の例外を除き、リソースが開始しているかどうかは、デバイスの定期的なモニターのみを制御するものとなり、使用可能かどうかは制御しません。

- フェンスデバイスは、**pcs stonith disable stonith_id** コマンドを実行して無効にできます。これにより、ノードがそのデバイスを使用できないように設定できます。
- 特定のノードがフェンスデバイスを使用できないようにするには、**pcs constraint location ... avoids** コマンドで、フェンスリソースの場所制約を設定できます。
- **stonith-enabled=false** を設定すると、フェンシングがすべて無効になります。ただし、実稼働環境でフェンシングを無効にすることは適していないため、フェンシングが無効になっている時のクラスターは、Red Hat ではサポートされないことに注意してください。

表8.1「フェンスデバイスの一般的なプロパティー」 は、フェンスデバイスに設定できる一般的なプロパティーを説明します。

表8.1 フェンスデバイスの一般的なプロパティー

フィールド	タイプ	デフォルト	説明
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。たとえば、**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 を使用するように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 (**pcmk_host_check=static-list** 以外は任意)。
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。使用できる値は、**dynamic-list** (デバイスに問い合わせる)、**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) です。

8.4. 高度な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プロパティーは 表8.2「フェンスデバイスの高度なプロパティー」 にまとめられています。これらのオプションは高度な設定を行う場合にのみ使用されます。

表8.2 フェンスデバイスの高度なプロパティー

フィールド	タイプ	デフォルト	説明

フィールド	タイプ	デフォルト	説明
pcmk_host_argument	文字列	port	port の代替パラメーターです。デバイスによっては、標準の port パラメーターに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このパラメーターで、フェンシングするマシンを示す、デバイス固有の代替パラメーターを指定します。クラスターが追加パラメーターを提供しないようにするには、**none** 値を使用します。
pcmk_reboot_action	文字列	reboot	**reboot** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このパラメーターを使用して、再起動を実行するデバイス固有のコマンドを指定します。
pcmk_reboot_timeout	時間	60s	**stonith-timeout** の代替コマンドで、再起動にタイムアウトを指定します。デバイスにより、再起動が完了するまでの時間が通常より長く、または短くなります。このパラメーターを使用して、再起動にデバイス固有のタイムアウトを指定します。
pcmk_reboot_retries	整数	2	タイムアウト期間内に、**reboot** コマンドを再試行する回数の上限です。デバイスによっては、複数の接続に対応しておらず、別のタスクでビジー状態になるとデバイスが操作に失敗する場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker による再起動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_off_action	文字列	off	**off** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このような場合は、このパラメーターを使用して、オフ操作を実行するデバイス固有のコマンドを指定します。
pcmk_off_timeout	時間	60s	**stonith-timeout** の代替コマンドで、オフ操作にタイムアウトを指定します。デバイスにより、オフ操作にかかる時間が通常より長く、または短くなります。このパラメーターを使用して、オフ操作にデバイス固有のタイムアウトを指定します。
pcmk_off_retries	整数	2	タイムアウト期間内に、off コマンドを再試行する回数の上限です。デバイスによっては、複数の接続に対応しておらず、別のタスクでビジー状態になるとデバイスが操作に失敗する場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker によるオフ操作の再試行回数を変更する場合に使用します。
pcmk_list_action	文字列	list	**list** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このような場合は、このパラメーターを使用して、list 操作を実行するデバイス固有のコマンドを指定します。

フィールド	タイプ	デフォルト	説明
pcmk_list_timeout	時間	60s	list 操作にタイムアウトを指定します。デバイスにより、list が完了するまでの時間が通常より長く、または短くなります。このパラメーターを使用して、list 操作にデバイス固有のタイムアウトを指定します。
pcmk_list_retries	整数	2	タイムアウト期間内に、**list** コマンドを再試行する回数の上限です。デバイスによっては、複数の接続に対応しておらず、別のタスクでビジー状態になるとデバイスが操作に失敗する場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker による list 操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** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このような場合は、このパラメーターを使用して、監視操作を実行するデバイス固有のコマンドを指定します。
pcmk_monitor_timeout	時間	60s	**stonith-timeout** の代替コマンドで、監視にタイムアウトを指定します。デバイスによっては、監視にかかる時間が通常より長く、または短くなります。このパラメーターを使用して、監視操作にデバイス固有のタイムアウトを指定します。

フィールド	タイプ	デフォルト	説明
pcmk_monitor_retries	整数	2	タイムアウト期間内に、**monitor** コマンドを再試行する回数の上限です。デバイスによっては、複数接続に対応しておらず、別のタスクでビジー状態になるとデバイスが操作に失敗する場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker による監視の再試行回数を変更する場合に使用します。
pcmk_status_action	文字列	status	**status** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このような場合は、このパラメーターを使用して、status 操作を実行するデバイス固有のコマンドを指定します。
pcmk_status_timeout	時間	60s	**stonith-timeout** の代替コマンドで、status 操作にタイムアウトを指定します。デバイスによっては、status にかかる時間が通常より長く、または短くなります。このパラメーターを使用して、status 操作にデバイス固有のタイムアウトを指定します。
pcmk_status_retries	整数	2	タイムアウト期間内に、status コマンドを再試行する回数の上限です。デバイスによっては、複数接続に対応しておらず、別のタスクでビジー状態になるとデバイスが操作に失敗する場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker による status 操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_delay_base	時間	0s	stonith 操作のベース遅延を有効にし、ベース遅延の値を指定します。これにより、複数の異なる遅延がノードに設定されている場合に、ダブルフェンシングが発生しなくなります。このオプションを使用して、stonith 操作の静的な遅延を有効にします。全体の遅延は、合計が最大遅延を下回るように、ランダムな遅延値に静的遅延を加算します。
pcmk_delay_max	時間	0s	stonith 動作のランダムな遅延を有効にし、ランダムな遅延の最大値を指定します。これにより、SBD などの低速デバイスを使用している場合に、ダブルフェンシングが発生しなくなります。このオプションを使用して、stonith 操作のランダムな遅延を有効にします。全体の遅延は、合計が最大遅延を下回るように、このランダムな遅延値に静的遅延を加算します。
pcmk_action_limit	整数	1	このデバイスで並行して実行できる操作の上限です。最初に、クラスタープロパティーの **concurrent-fencing=true** を設定する必要があります。値を -1 にすると無制限になります。
pcmk_on_action	文字列	on	高度な使用 - **on** の代替コマンドです。デバイスによっては、標準コマンドに対応していない場合や、そのデバイス固有のパラメーターも提供している場合があります。このような場合は、このパラメーターを使用して、**on** 操作を実行するデバイス固有のコマンドを指定します。

フィールド	タイプ	デフォルト	説明
pcmk_on_timeout	時間	60s	高度な使用 - **stonith-timeout** の代替コマンドで、**on** 操作にタイムアウトを指定します。デバイスにより、この操作が完了するのにかかる時間が通常より長く、または短くなります。このパラメーターを使用して、**on** 操作にデバイス固有のタイムアウトを指定します。
pcmk_on_retries	整数	2	高度な使用 - タイムアウト期間内に、**on** コマンドを再試行する回数の上限です。デバイスによっては、複数接続に対応しておらず、別のタスクでビジー状態になるとデバイス操作が **fail** になる場合があるため、タイムアウトに達していなければ、Pacemaker が操作を自動的に再試行します。このオプションは、Pacemaker による **on** 操作の再試行回数を変更する場合に使用します。

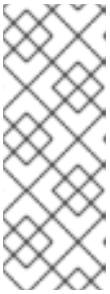
8.5. フェンスデバイスのテスト

フェンシングは、Red Hat Cluster インフラストラクチャーの基本的な部分を構成しているため、フェンシングが適切に機能していることを確認またはテストすることは重要です。

以下の手順で、フェンスデバイスをテストします。

1. デバイスへの接続に使用する ssh、telnet、HTTP などのリモートプロトコルを使用して、手動でログインしてフェンスデバイスをテストしたり、出力される内容を確認します。たとえば、IPMI 対応デバイスのフェンシングを設定する場合は、 **ipmitool** を使用してリモートでのログインを試行します。手動でログインする際に使用するオプションに注意してください。これらのオプションは、フェンスエージェントを使用する際に必要になる場合があります。
   フェンスデバイスにログインできない場合は、そのデバイスが ping 可能であること、ファイアウォール設定フェンスデバイスへのアクセスを妨げていないこと、フェンスデバイスでリモートアクセスが有効になっていること、認証情報が正しいことなどを確認します。

2. フェンスエージェントスクリプトを使用して、フェンスエージェントを手動で実行します。フェンスエージェントを実行するのに、クラスターサービスが実行している必要はないため、デバイスをクラスターに設定する前にこのステップを完了できます。これにより、先に進む前に、フェンスデバイスが適切に応答することを確認できます。



注記

本セクションの例では、iLO デバイスの **fence_ipmilan** フェンスエージェントスクリプトを使用します。実際に使用するフェンスエージェントと、そのエージェントを呼び出すコマンドは、お使いのサーバーハードウェアによって異なります。指定するオプションを確認するには、フェンスエージェントの man ページを参照してください。通常は、フェンスデバイスのログインとパスワードなどの情報を把握しておく必要があります。

以下の例は、**-o status** パラメーターを指定して **fence_ipmilan** フェンスエージェントスクリプトを実行する場合に使用する形式になります。このコマンドを実行すると、フェンシングを実行せずに、別のノードのフェンスデバイスインターフェースのステータスを確認します。ノードの再起動を試行する前にデバイスをテストして、機能させることができます。このコマンドを実行する際に、iLO デバイスの電源をオン/オフにするパーミッションを持つ iLO ユーザーの名前およびパスワードを指定します。

```
# fence_ipmilan -a ipaddress -l username -p password -o status
```

以下の例は、**-o reboot** パラメーターを指定して **fence_ipmilan** フェンスエージェントスクリプトを実行するのに使用する形式になります。このコマンドをノードで実行すると、この iLO デバイスで管理するノードが再起動します。

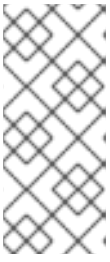
```
# fence_ipmilan -a ipaddress -l username -p password -o reboot
```

フェンスエージェントがステータス、オフ、オン、または再起動の動作を適切に実行しない場合は、ハードウェア、フェンスデバイスの設定、およびコマンドの構文を確認する必要があります。さらに、デバッグ出力を有効にした状態で、フェンスエージェントスクリプトを実行できます。デバッグ出力は、一部のフェンスエージェントで、フェンスデバイスにログインする際に、フェンスエージェントスクリプトに問題が発生しているイベントシーケンスの場所を確認するのに役に立ちます。

```
# fence_ipmilan -a ipaddress -l username -p password -o status -D /tmp/$(hostname)-fence_agent.debug
```

発生した障害を診断する際に、フェンスデバイスに手動でログインする際に指定したオプションが、フェンスエージェントスクリプトでフェンスエージェントに渡した内容と同一であることを確認する必要があります。

フェンスエージェントが、暗号化した接続に対応する場合は、証明書の検証で障害が生じているためにエラーが出力される場合があり、ホストを信頼することや、フェンスエージェントの **ssl-insecure** パラメーターを使用することが求められます。同様に、ターゲットデバイスで SSL/TLS を無効にした場合は、フェンスエージェントの SSL パラメーターを設定する際に、これを考慮しないといけない場合があります。



注記

テストしているフェンスエージェントが **fence_drac** または **fence_ilo** の場合、もしくはその他の、継続して失敗しているシステム管理デバイスのフェンスエージェントの場合は、フォールバックして **fence_ipmilan** を試行します。多くの場合、システム管理カードは IPMI リモートログインに対応しており、フェンスエージェントとしては **fence_ipmilan** だけに対応しています。

3. フェンスデバイスを、手動で機能したオプションと同じオプションでクラスターに設定し、クラスターを起動したら、以下の例にあるように、任意のノードから **pcs stonith fence** コマン

ドを実行してフェンシングをテストします (または複数のノードから複数回実行します)。**pcs stonith fence** コマンドは、クラスター設定を CIB から読み取り、フェンス動作を実行するように設定したフェンスエージェントを呼び出します。これにより、クラスター設定が正確であることが確認できます。

```
# pcs stonith fence node_name
```

pcs stonith fence コマンドに成功した場合は、フェンスイベントの発生時に、クラスターのフェンシング設定が機能します。このコマンドが失敗した場合は、クラスター管理が検出した設定でフェンスデバイスを起動することができません。以下の問題を確認し、必要に応じてクラスター設定を更新します。

- フェンス設定を確認します。たとえば、ホストマップを使用したことがある場合は、システムが指定したホスト名を使用して、システムがノードを見つけられるようにする必要があります。
- デバイスのパスワードおよびユーザー名に、bash シェルが誤って解釈する可能性がある特殊文字が含まれるかどうかを確認します。パスワードおよびユーザー名は引用符で囲み、この問題に対応してください。
- **pcs stonith** コマンドで IP アドレスまたはホスト名を使用してデバイスに接続できるかどうかを確認してください。たとえば、stonith コマンドでホスト名を指定し、IP アドレスを使用して行ったテストは有効ではありません。
- フェンスデバイスが使用するプロトコルにアクセスできる場合は、そのプロトコルを使用してデバイスへの接続を試行します。たとえば、多くのエージェントのが ssh または telnet を使用します。デバイスへの接続は、デバイスの設定時に指定した認証情報を使用して試行する必要があります。これにより、有効なプロンプトを取得し、そのデバイスにログインできるかどうかを確認できます。
  すべてのパラメーターが適切であることが確認できたものの、フェンスデバイスには接続できない時に、フェンスデバイスでログ機能が使用できる場合は、ログを確認できます。これにより、ユーザーが接続したかどうかと、ユーザーが実行したコマンドが表示されます。**/var/log/messages** ファイルで stonith やエラーを確認すれば、発生している問題のヒントが得られる可能性もあります。また、エージェントによっては、より詳細な情報が得られる場合があります。

4. フェンスデバイステストに成功し、クラスターが稼働したら、実際の障害をテストします。このテストでは、クラスターで、トークンの損失を生じさせる動作を実行します。
   - ネットワークを停止します。ネットワークの利用方法は、設定により異なりますが、多くの場合は、ネットワークケーブルまたは電源ケーブルをホストから物理的に抜くことができます。ネットワーク障害をシミュレートする方法は「What is the proper way to simulate a network failure on a RHEL Cluster?」を参照してください。

     注記

     ネットワークや電源ケーブルを物理的に切断せずに、ローカルホストのネットワークインターフェースを無効にすることは、フェンシングのテストとしては推奨されません。実際に発生する障害を正確にシミュレートしていないためです。

   - ローカルのファイアウォールを使用して、corosync の受信トラフィックおよび送信トラフィックをブロックします。
     以下の例では corosync をブロックします。ここでは、デフォルトの corosync ポートと、ローカルのファイアウォールとして **firewalld** が使用されていることと、corosync が使用

するネットワークインターフェースがデフォルトのファイアウォールゾーンにあることが前提となっています。

```
# firewall-cmd --direct --add-rule ipv4 filter OUTPUT 2 -p udp --dport=5405 -j DROP
# firewall-cmd --add-rich-rule='rule family="ipv4" port port="5405" protocol="udp" drop'
```

- **sysrq-trigger** でクラッシュをシミュレートし、マシンをクラッシュします。ただし、カーネルパニックを発生させると、データが損失する可能性があることに注意してください。クラッシュする前に、クラスターリソースを無効にすることが推奨されます。

```
# echo c > /proc/sysrq-trigger
```

8.6. フェンスレベルの設定

Pacemaker は、フェンストポロジーと呼ばれる機能を用いて、複数デバイスでのノードのフェンシングをサポートします。トポロジーを実装するには、通常の方法で各デバイスを作成し、設定のフェンストポロジーセクションでフェンスレベルを 1 つ以上定義します。

- レベルは、1 から昇順で試行されていきます。
- デバイスに障害が発生すると、現行レベルの処理は終了し、同レベルのデバイスへの試行は行われなくなり、次のレベルが試行されます。
- すべてのデバイスのフェンシング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なくなります。
- いずれかのレベルで成功するか、すべてのレベルが試行され失敗すると、操作は終了します。

ノードにフェンスレベルを追加する場合は、次のコマンドを使用します。デバイスは、stonith ID をコンマで区切って指定します。指定したデバイスが、指定したレベルで試行します。

```
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フェンスレベルが一覧表示されます。

```
pcs stonith level
```

以下の例では、ノード **rh7-2** に、2 つのフェンスデバイス (il0 フェンスデバイス **my_ilo**と、apc フェンスデバイス **my_apc**) が設定されています。このコマンドはフェンスレベルを設定し、デバイス **my_ilo** が失敗し、ノードがフェンスできない場合に、Pacemaker がデバイス **my_apc** の使用を試行できるようにします。この例では、レベルを設定した後の **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、指定したノードおよびデバイス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ないと、指定したフェンスレベルがすべてのノードから削除されます。

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

以下のコマンドを使用すると、指定したノードや stonith id のフェンスレベルが削除されます。ノードや stonith id を指定しないと、すべてのフェンスレベルが削除されます。

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

複数の stonith ID を指定する場合は、コンマで区切ります。空白文字は使用しません。以下に例を示します。

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

以下のコマンドを使用して、フェンスレベルに指定したフェンスデバイスとノードがすべて、実際に存在することを確認します。

```
pcs stonith level verify
```

フェンストポロジーのノードは、ノード名に適用する正規表現と、ノードの属性 (およびその値) で指定できます。たとえば、次のコマンドでは、ノード **node1**、**node2**、および **node3** で、フェンスデバイス **apc1** および **apc2** を使用するように設定し、ノード **node4**、**node5**、および **node6** には、フェンスデバイス **apc3** および **apc4** を使用するように設定します。

```
pcs stonith level add 1 "regexp%node[1-3]" apc1,apc2
pcs stonith level add 1 "regexp%node[4-6]" apc3,apc4
```

次のコマンドでは、ノード属性のマッチングを使用して、同じように設定します。

```
pcs node attribute node1 rack=1
pcs node attribute node2 rack=1
pcs node attribute node3 rack=1
pcs node attribute node4 rack=2
pcs node attribute node5 rack=2
pcs node attribute node6 rack=2
pcs stonith level add 1 attrib%rack=1 apc1,apc2
pcs stonith level add 1 attrib%rack=2 apc3,apc4
```

8.7. 冗長電源のフェンシング設定

冗長電源にフェンシングを設定する場合は、ホストを再起動するときに、クラスターが、最初に両方の電源をオフにしてから、いずれかの電源をオンにするようにする必要があります。

ノードの電源が完全にオフにならないと、ノードがリソースを解放しない場合があります。この場合、解放できなかったリソースに複数のノードが同時にアクセスして、リソースが破損する可能性があります。

以下の例にあるように、各デバイスを一度だけ定義し、両方のデバイスがノードのフェンスに必要であると指定する必要があります。

```
# pcs stonith create apc1 fence_apc_snmp ipaddr=apc1.example.com login=user
passwd='7a4D#1j!pz864' pcmk_host_map="node1.example.com:1;node2.example.com:2"

# pcs stonith create apc2 fence_apc_snmp ipaddr=apc2.example.com login=user
passwd='7a4D#1j!pz864' pcmk_host_map="node1.example.com:1;node2.example.com:2"
```

```
# pcs stonith level add 1 node1.example.com apc1,apc2
# pcs stonith level add 1 node2.example.com apc1,apc2
```

8.8. 設定済みのフェンスデバイスの表示

以下のコマンドは、現在設定されているフェンスデバイスをすべて表示します。*stonith_id* を指定すると、設定されているその stonith デバイスのオプションだけが表示されます。**--full** オプションを指定すると、設定されている stonith オプションがすべて表示されます。

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

8.9. 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
```

8.10. 手動によるクラスターノードのフェンシング

次のコマンドで、ノードを手動でフェンシングできます。**--off** を指定すると、stonith に対して API 呼び出しの **off** を使用し、ノードをオフにします (再起動はしません)。

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

ノードがアクティブでない場合でも、そのノードを stonith デバイスがフェンスできない状況では、そのノードのリソースをクラスターが復旧できない可能性があります。この場合は、ノードの電源を手動でオフにしたあと、以下のコマンドを実行して、ノードの電源が切れていて、そのリソースが解放される復旧できる状態であることが確認できます。

警告

指定したノードが実際にオフになっていない状態で、クラスターソフトウェア、または通常クラスターが制御するサービスを実行すると、データ破損またはクラスター障害が発生します。

```
pcs stonith confirm node
```

8.11. フェンスデバイスの無効化

フェンスデバイス/リソースを無効にするには、**pcs stonith disable** コマンドを実行します。

以下のコマンドは、フェンスデバイス **myapc** を無効にします。

```
# pcs stonith disable myapc
```

8.12. ノードがフェンスデバイスを使用することを禁止

特定のノードがフェンスデバイスを使用できないようにするには、フェンスリソースの場所の制約を設定します。

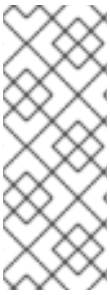
以下の例では、フェンスデバイスの **node1-ipmi** が、**node1** で実行されないようにします。

```
# pcs constraint location node1-ipmi avoids node1
```

8.13. 統合フェンスデバイスで使用する ACPI の設定

クラスターが統合フェンスデバイスを使用する場合は、即時かつ完全なフェンシングを実行できるように、ACPI (Advanced Configuration and Power Interface) を設定する必要があります。

クラスターノードが統合フェンスデバイスでフェンシングされるように設定されている場合は、そのノードの ACPI Soft-Off を無効にします。ACPI Soft-Off を無効にすることにより、統合フェンスデバイスは、クリーンシャットダウンを試行する代わりに、ノードを即時に、かつ完全にオフにできます (例: **shutdown -h now**)。ACPI Soft-Off が有効になっている場合は、統合フェンスデバイスがノードをオフにするのに 4 秒以上かかる可能性があります (以下の注記部分を参照してください)。さらに、ACPI Soft-Off が有効になっていて、ノードがシャットダウン時にパニック状態になるか、フリーズすると、統合フェンスデバイスがノードをオフにできない場合があります。このような状況では、フェンシングは遅延するか、または成功しません。したがって、ノードが統合フェンスデバイスでフェンシングされ、ACPI Soft-Off が有効になっている場合は、クラスターが徐々に復元します。または管理者の介入による復旧が必要になります。



注記

ノードのフェンシングにかかる時間は、使用している統合フェンスデバイスによって異なります。統合フェンスデバイスの中には、電源ボタンを押し続けるのと同じ動作を実行するものもあります。この場合は、ノードがオフになるのに 4 秒から 5 秒かかります。また、電源ボタンを押してすぐ離すのと同等の動作を行い、ノードの電源をオフにする行為をオペレーティングシステムに依存する統合フェンスデバイスもあります。この場合は、ノードがオフになるのにかかる時間は 4~5 秒よりも長くなります。

- ACPI Soft-Off を無効にする場合は、BIOS 設定を「instant-off」、またはこれに類似する設定に変更することが推奨されます。これにより、「BIOS で ACPI Soft-Off を無効化」 で説明しているように、ノードは遅延なくオフになります。

システムによっては、BIOS で ACPI Soft-Off を無効にできません。お使いのクラスターでは、BIOS で ACPI Soft-Off を無効にできない場合に、以下のいずれかの方法で ACPI Soft-Off を無効にできます。

- 「logind.conf ファイルで ACPI Soft-Off を無効化」 で説明しているように、**HandlePowerKey=ignore** を **/etc/systemd/logind.conf** ファイルに設定し、ノードがフェンシングされるとすぐにオフになることを確認します。これが、ACPI Soft-Off を無効にする 1 つ目の代替方法です。

- 「GRUB 2 ファイルを使用した ACPI の完全な無効化」 で説明しているように、カーネル起動

コマンドラインに **acpi=off** を追加します。これは、ACPI Soft-Off を無効にする 2 つ目の代替方法です。この方法の使用が推奨される場合、または 1 つ目の代替方法が利用できない場合に使用してください。

重要

この方法は、ACPI を完全に無効にします。コンピューターの中には、ACPI が完全が無効になってるとシステムが正しく起動しないものもあります。お使いのクラスターに適した方法が他にない場合に **限り**、この方法を使用してください。

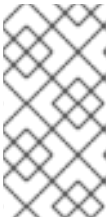
8.13.1. BIOS で ACPI Soft-Off を無効化

以下の手順で、各クラスターノードの BIOS を設定して、ACPI Soft-Off を無効にできます。

注記

BIOS で ACPI Soft-Off を無効にする手順は、サーバーシステムにより異なる場合があります。この手順は、お使いのハードウェアのドキュメントで確認する必要があります。

1. ノードを再起動して **BIOS CMOS Setup Utility** プログラムを起動します。
2. 電源メニュー (または同等の電源管理メニュー) に移動します。
3. 電源メニューで、**Soft-Off by PWR-BTTN** 機能 (または同等の機能) を **Instant-Off** (または、遅延なく電源ボタンでノードをオフにする設定と同等の設定) に設定します。**BIOS CMOS Setup Utility** では、**ACPI Function** が **Enabled** に設定され、**Soft-Off by PWR-BTTN** が **Instant-Off** に設定されている電源メニューを表示します。



注記

ACPI Function、**Soft-Off by PWR-BTTN**、および **Instant-Off** と同等の機能は、コンピューターによっては異なる場合がありますが、この手順では、電源ボタンを使用して、コンピューターを遅延なくオフにするように BIOS を設定することを説明します。

4. **BIOS CMOS Setup Utility** プログラムを終了します。BIOS 設定が保存されます。
5. ノードがフェンシングされるとすぐにオフになることを確認します。フェンスデバイスのテストの詳細は「フェンスデバイスのテスト」を参照してください。

BIOS CMOS Setup Utility

```
`Soft-Off by PWR-BTTN` set to
`Instant-Off`
```

```
+---------------------------------------------|-------------------+
|   ACPI Function             [Enabled]       |   Item Help       |
|   ACPI Suspend Type         [S1(POS)]       |-------------------|
| x Run VGABIOS if S3 Resume   Auto           |   Menu Level   *  |
|   Suspend Mode              [Disabled]      |                   |
|   HDD Power Down            [Disabled]      |                   |
|   Soft-Off by PWR-BTTN      [Instant-Off    |                   |
|   CPU THRM-Throttling       [50.0%]         |                   |
```

```
|   Wake-Up by PCI card       [Enabled]     |                |
|   Power On by Ring          [Enabled]     |                |
|   Wake Up On LAN            [Enabled]     |                |
| x USB KB Wake-Up From S3    Disabled      |                |
|   Resume by Alarm           [Disabled]    |                |
| x  Date(of Month) Alarm       0           |                |
| x  Time(hh:mm:ss) Alarm       0 :  0 :    |                |
|   POWER ON Function         [BUTTON ONLY  |                |
| x KB Power ON Password        Enter       |                |
| x Hot Key Power ON          Ctrl-F1       |                |
|                                           |                |
|                                           |                |
+-------------------------------------------|----------------+
```

この例では、**ACPI Function** が **Enabled** に設定され、**Soft-Off by PWR-BTTN** が **Instant-Off** に設定されていることを示しています。

8.13.2. logind.conf ファイルで ACPI Soft-Off を無効化

/etc/systemd/logind.conf ファイルで電源キーの処理を無効にするには、以下の手順を行います。

1. **/etc/systemd/logind.conf** ファイルに、以下の設定を定義します。

```
HandlePowerKey=ignore
```

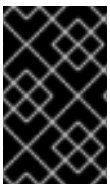
2. **systemd** 設定をリロードします。

```
# systemctl daemon-reload
```

3. ノードがフェンシングされるとすぐにオフになることを確認します。フェンスデバイスのテストの詳細は「フェンスデバイスのテスト」を参照してください。

8.13.3. GRUB 2 ファイルを使用した ACPI の完全な無効化

ACPI Soft-Off は、カーネルの GRUB メニューエントリーに **acpi=off** を追加して無効にできます。



重要

この方法は、ACPI を完全に無効にします。コンピューターの中には、ACPI が完全が無効になってるとシステムが正しく起動しないものもあります。お使いのクラスターに適した方法が他にない場合に **限り**、この方法を使用してください。

以下の手順で、GRUB 2 ファイルで ACPI を無効にします。

1. 以下のように、**grubby** ツールで、**--args** オプションと **--update-kernel** オプションを使用して、各クラスターノードの **grub.cfg** ファイルを変更します。

```
# grubby --args=acpi=off --update-kernel=ALL
```

2. ノードを再起動します。

3. ノードがフェンシングされるとすぐにオフになることを確認します。フェンスデバイスのテストの詳細は「フェンスデバイスのテスト」を参照してください。

第9章 クラスターリソースの設定

クラスターリソースを作成するコマンドの形式は、以下のとおりです。

```
pcs resource create resource_id [standard:[provider:]]type [resource_options] [op
operation_action operation_options [operation_action operation options]...] [meta
meta_options...] [clone [clone_options] | master [master_options] | --group group_name [--before
resource_id | --after resource_id] | [bundle bundle_id] [--disabled] [--wait[=n]]
```

主なクラスターリソースの作成オプションには、以下が含まれます。

- **--group** オプションを指定すると、名前付きのリソースグループにリソースが追加されます。グループが存在しない場合は作成され、そのグループにリソースが追加されます。
- **--before** および **--after** オプションは、リソースグループに含まれるリソースを基準にして、追加するリソースの位置を指定します。
- **--disabled** オプションは、リソースが自動的に起動しないことを示しています。

リソースの制約を設定することで、クラスターにおけるリソースの動作を決定できます。

リソース作成の例

以下のコマンドは、仕様 **ocf**、プロバイダー **heartbeat**、およびタイプ **IPaddr2** リソース **VirtualIP** を作成します。このリソースのフローティングアドレスは 192.168.0.120 であり、システムは、30 秒間隔で、リソースが実行していることを確認します。

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

または、standard フィールドおよび provider フィールドを省略し、以下のようにコマンドを実行できます。デフォルトの **ocf** リソース ID、および **heartbeat** プロバイダーに設定されます。

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

設定済みリソースの削除

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
```

たとえば、以下のコマンドは、リソース ID が **VirtualIP** である既存リソースを削除します。

```
# pcs resource delete VirtualIP
```

9.1. リソースエージェント識別子

リソースに定義する識別子は、リソースに使用するエージェント、そのエージェントを検索する場所、およびそれが準拠する仕様をクラスターに指示します。表9.1「リソースエージェント識別子」 では、これらのプロパティーを説明します。

表9.1 リソースエージェント識別子

フィールド	説明
standard	エージェントが準拠する標準。許可される値とその意味は以下のとおりです。 * **ocf** - 指定した **type** は、Open Cluster Framework Resource Agent API に準拠した実行可能ファイルの名前で、**/usr/lib/ocf/resource.d/provider** の下に置かれます。 * **lsb** - 指定した **type** は、Linux Standard Base Init Script Actions に準拠する実行ファイルの名前です。type で完全パスを指定しないと、システムは **/etc/init.d** ディレクトリーを探します。 * **systemd** - 指定した **type** は、インストール済み **systemd** ユニットの名前です。 * **service** - Pacemaker は、指定した **type** を選択します (まずは **lsb** エージェントとして、次に **systemd** エージェントとして) * **nagios** - 指定した **type** は、Nagios Plugin API に準拠する実行可能ファイルの名前で、**/usr/libexec/nagios/plugins** ディレクトリーに置かれ、OCF スタイルのメタデータが別途 **/usr/share/nagios/plugins-metadata** ディレクトリーに置かれます (一般的なプラグインは **nagios-agents-metadata** パッケージで入手可能)。
type	使用するリソースエージェントの名前 (例: **IPaddr** または **Filesystem**)
provider	OCF 仕様により、複数のベンダーで同じリソースエージェントを指定できます。Red Hat が提供するエージェントのほとんどは、プロバイダーに **heartbeat** を使用しています。

表9.2「リソースプロパティーを表示させるコマンド」 は、利用可能なリソースプロパティーを表示するコマンドの概要を示しています。

表9.2 リソースプロパティーを表示させるコマンド

pcs Display Command	出力
pcs resource list	利用できる全リソースの一覧を表示
pcs resource standards	利用できるリソースエージェントの一覧を表示
pcs resource providers	利用できるリソースエージェントの一覧を表示
pcs resource list string	できます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します。仕様名、プロバイダー名、タイプ名などでフィルターを指定して、リソースを表示できます。

9.2. リソース固有のパラメーターの表示

各リソースで以下のコマンドを使用すると、リソースの説明、そのリソースに設定できるパラメーター、およびそのリソースに設定されるデフォルト値が表示されます。

```
pcs resource describe [standard:[provider:]]type
```

たとえば、以下のコマンドは、タイプ **apache** のリソース情報を表示します。

```
# pcs resource describe ocf:heartbeat:apache
This is the resource agent for the Apache Web server.
This resource agent operates both version 1.x and version 2.x Apache
servers.

...
```

9.3. リソースのメタオプションの設定

リソース固有のパラメーターのほかにも、リソースにリソースオプションを設定できます。このような追加オプションは、クラスターがリソースの動作を決定する際に使用されます。

表9.3「リソースのメタオプション」 は、リソースメタオプションを示しています。

表9.3 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アクティブにできない場合に、クラスターは優先度の低いリソースを停止して、優先度が高いリソースを実行し続けます。
target-role	**Started**	クラスターが維持するリソースのステータスです。以下の値が使用できます。 * **Stopped** - リソースの強制停止 * **Started** - リソースの起動を許可 (クローン昇格でき、適切な場合は、マスターロールに昇格する) * **Master** - リソースの起動を許可し、必要に応じて昇格します。 * **Slave** - リソースが昇格可能な場合に、リソースを開始できますが、スレーブモードでのみ可能
is-managed	**true**	クラスターによるリソースの起動と停止の許可の有無。使用できる値は **true** または **false** です。
resource-stickiness	0	リソースを同じ場所に残すための優先度の値です。

フィールド	デフォルト	説明
requires	Calculated	リソースの起動を許可する条件です。 以下の条件の場合を除き、**fencing** がデフォルトに設定されます。以下の値が使用できます。 * **nothing** - クラスターによるリソースの起動を常に許可します。 * **quorum** - クラスターは、設定されているノードの過半数がアクティブな場合に限りこのリソースを起動できます。**stonith-enabled** が **false** に設定されている場合、またはリソースの **standard** が **stonith** の場合は、このオプションがデフォルト値となります。 * **fencing** -設定されているノードの過半数がアクティブ、**かつ** 障害が発生しているノードや不明なノードがフェンスになっている場合に限り、クラスターはこのリソースの起動を許可できます。 * **unfencing** - 設定されているノードの過半数がアクティブ、**かつ** 障害が発生しているノードや不明なノードがすべてフェンスされている場合に限り、**アンフェンシング** 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します。**provides=unfencing stonith** メタオプションがフェンスデバイスに設定されている場合のデフォルト値です。
migration-threshold	**INFINITY**	指定したリソースが任意のノードで失敗した回数です。この回数を超えると、そのノードは、このリソースのホストとして不適格とするマークがつけられます。値を 0 にするとこの機能は無効になり、ノードに不適格マークがつけられることはありません。**INFINITY** (デフォルト) に設定すると、クラスターは、これを非常に大きい有限数として扱います。このオプションは、失敗した操作に **on-fail=restart** (デフォルト) が設定されていて、かつ失敗した起動操作のクラスタープロパティー **start-failure-is-fatal** が **false** に設定されている場合に限り有効です。

フィールド	デフォルト	説明
failure-timeout	**0** (無効)	**migration-threshold** オプションと併用されます。障害が発生しなかったかのように動作するまで待機する秒数を示します。リソースの実行に失敗したノードで、そのリソースの実行を再度許可する可能性があります。
multiple-active	**stop_start**	その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、クラスターが実行する動作です。以下の値が使用できます。 * **block** - リソースに unmanaged のマークを付けます。 * **stop_only** - 実行しているインスタンスをすべて停止し、以降の動作は行いません。 * **stop_start** - 実行中のインスタンスをすべて停止してから、リソースを 1 カ所でのみ起動します。

9.3.1. リソースオプションのデフォルト値の変更

リソースオプションのデフォルト値を変更する場合は、以下のコマンドを使用します。

```
pcs resource defaults options
```

たとえば、以下のコマンドは、**resource-stickiness** のデフォルト値を 100 にリセットします。

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

9.3.2. 現在設定されているリソースデフォルトの表示

pcs resource defaults で *options* パラメーターを省略すると、現在設定されているリソースオプションのデフォルト値の一覧が表示されます。以下の例は、**resource-stickiness** のデフォルト値を 100 にリセットした後の出力になります。

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

9.3.3. リソース作成でメタオプションの設定

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たかに関わらず、リソースを作成する際に、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できます。以下の形式は、リソースのメタオプションの値を指定する際に使用する **pcs resource create** コマンドです。

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

たとえば、以下のコマンドでは **resource-stickiness** の値を 50 に設定したリソースを作成します。

 # pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=192.168.0.120 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 meta_options**

以下のコマンドは、既存の **dummy_resource** というリソースに、**failure-timeout** メタオプションの値を 20 秒に設定します。これにより、20 秒で同じノードで、リソースの再起動を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、リソースの値を表示して、**failure-timeout=20s** が設定されているかどうかを確認できます。

```
# pcs resource config dummy_resource
 Resource: dummy_resource (class=ocf provider=heartbeat type=Dummy)
  Meta Attrs: failure-timeout=20s
  ...
```

9.4. リソースグループの設定

クラスターの最も一般的な要素の 1 つであるリソースのセットは、共に配置され、順番に起動し、逆順で停止する必要があります。この設定を簡単にするため、 Pacemaker はリソースグループの概念をサポートします。

9.4.1. リソースグループの作成

以下のコマンドを使用してリソースグループを作成し、グループに含めるリソースを指定します。グループが存在しない場合は、このコマンドによりグループが作成されます。グループが存在する場合は、このコマンドにより追加のリソースがグループに追加されます。リソースは、このコマンドで指定された順序で起動し、その逆順で停止します。

pcs resource group add **group_name resource_id** [**resource_id**] ... [**resource_id**] [--before **resource_id** | --after **resource_id**]

このコマンドの **--before** オプションおよび **--after** オプションを使用すると、リソースグループに存在するリソースを基準として、追加するリソースの相対的な位置を指定できます。

また、以下のコマンドを使用すると、リソースの作成時に、そのリソースを既存グループに追加できます。以下のコマンドでは、作成するリソースが **group_name** というグループに追加されます。**group_name** グループが存在しない場合は作成されます。

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

グループに含まれるリソースの数に制限はありません。グループの基本的なプロパティーは以下のとおりです。

- 複数のリソースが、グループ内で同じ場所に置かれています。
- リソースは、指定した順序で起動します。グループ内に実行できないリソースがあると、そのリソースの後に指定したリソースを実行することは許可されません。
- リソースは、指定した順序と逆の順序で停止します。

以下の例では、既存リソースの **IPaddr** と **Email** が含まれる **shortcut** というリソースグループが作成されます。

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

この例では、以下のように設定されています。

- **IPaddr** が最初に起動してから、**Email** が起動します。
- **Email** リソースが停止してから、 **IPAddr** が停止します。
- **IPaddr** を実行できない場合は、**Email** も実行できません。
- **Email** を実行できなくても、**IPaddr** には影響ありません。

9.4.2. リソースグループの削除

以下のコマンドを使用して、グループからリソースを削除します。グループにリソースが残っていないと、このコマンドによりグループ自体が削除されます。

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

9.4.3. リソースグループの表示

以下のコマンドは、現在設定されているリソースグループを一覧表示します。

```
pcs resource group list
```

9.4.4. グループオプション

リソースグループに、**priority** オプション、**target-role**オプション、**is-managed**オプションを設定し、1 つのリソースとして設定されている場合と同じ意味を維持します。リソースのメタオプションの詳細は、「リソースメタオプションの設定」を参照してください。

9.4.5. グループの粘着性

粘着性は、リソースを現在の場所に留ませる優先度の度合いを示し、グループで加算されます。つまり、グループ内でアクティブなリソースが持つ stickness 値の合計が、グループの合計になります。そのため、**resource-stickiness** のデフォルト値が 100 で、グループに 7 つのメンバーがあり、そのメンバーの 5 つがアクティブな場合は、グループ全体のスコアが 500 になります。

9.5. リソース動作の決定

リソースの制約を設定すると、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。以下の制約のカテゴリーを設定できます。

- **location** 制約 - この制約では、リソースを実行するノードを指定します。設定方法は「リソースを実行するノードの決定」を参照してください。
- **order** 制約 - この制約では、制約はリソースが実行する順序を決定します。設定方法は「クラスターリソースの実行順序の決定」を参照してください。
- **colocation** 制約 - この制約では、他のリソースとの対比でリソースの配置先を決定します。設定方法は「クラスターリソースのコロケーション」を参照してください。

複数リソースをまとめて配置して順番に起動する、または逆順で停止する一連の制約を簡単に設定する方法として、Pacemaker ではリソースグループという概念に対応しています。リソースグループの作成後に、個別のリソースの制約を設定するようにグループ自体に制約を設定できます。リソースグループの情報は「リソースグループの設定」を参照してください。

第10章 リソースを実行するノードの決定

場所の制約は、リソースを実行す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はリソースを実行しないなどを決定できます。

10.1. 場所の制約の設定

基本的な場所の制約を設定して、リソースの実行を特定のノードで優先するか、または回避するかを指定できます。オプションの **score** 値を使用して、制約の相対的な優先度を指定できます。

以下のコマンドは、リソースの実行を、指定した 1 つまたは複数のノードで優先するように、場所の制約を作成します。1 回のコマンドで、特定のリソースの制約を複数のノードに対して作成できます。

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

以下のコマンドは、指定したノードでリソースの実行を回避する場所の制約を作成します。

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

表10.1「場所の制約オプション」 では、場所の制約を設定する基本的なオプションを説明します。

表10.1 場所の制約オプション

フィールド	説明
rsc	リソース名
node	ノード名
score	指定のリソースを指定のノードで優先的に実行するか、または実行を回避するかの優先度を示す正の整数値。**score** のデフォルト値は、**INFINITY** です。 リソースを特定ノードで優先的に実行するように設定するコマンドで score の値を **INFINITY** にすると、そのノードが利用可能な場合はそのノードでリソースを優先的に実行しますが、利用できない場合に別のノードでそのリソースを実行しないようにする訳ではありません。リソースがノードを回避するように設定するコマンドで **INFINITY** を設定した場合は、その他のノードが利用できない場合でも、そのリソースはそのノードでは実行されません。 数値スコア (**INFINITY** 以外) は、制約がオプションで、それを上回る他の要因がない限り有効となることを意味します。たとえば、リソースが別のノードに置かれ、その **resource-stickiness** スコアが、場所制約のスコアよりも **優先** される場合、リソースはその場所に残されます。

以下のコマンドは、リソース **Webserver** が、ノード **node1** で優先的に実行するように指定する場所の制約を作成します。

```
pcs constraint location Webserver prefers node1
```

pcs では、コマンドラインの場所の制約に正規表現に対応しています。この制約は、リソース名に一致する正規表現に基づいて、複数のリソースに適用されます。これにより、1 つのコマンドラインで複数の場所の制約を設定できます。

以下のコマンドは、**dummy0** から **dummy9** までのリソースの実行が **node1** に優先されるように指定する場所の制約を作成します。

```
pcs constraint location 'regexp%dummy[0-9]' prefers node1
```

Pacemaker は、http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html#tag_09_04 に説明されているとおり、POSIX 拡張正規表現を使用するため、以下のコマンドを実行しても同じ制約を指定できます。

```
pcs constraint location 'regexp%dummy[[:digit:]]' prefers node1
```

10.2. ノードのサブセットへのリソース検出の制限

Pacemaker がリソースを開始する前に、そのリソースがすでに実行しているかどうかを確認するために、すべてのノードでワンタイムモニター操作 (しばしば「プローブ」と呼ばれています) を実行します。このリソース検出のプロセスは、モニターを実行できないノードではエラーになる場合があります。

ノードに場所の制約を設定する際に、**pcs constraint location** コマンドの **resource-discovery** オプションを使用して、指定のリソース対して Pacemaker がこのノードでリソース検出を実行するかどうかの優先度を指定できます。リソースが物理的に稼働可能なノードのサブセットへのリソース検出を制限すると、ノードが大量に存在する場合にパフォーマンスを大幅に改善できます。**pacemaker_remote** を使用して、ノード数を 100 単位で拡大する場合は、このオプションの使用を検討してください。

以下のコマンドは、**pcs constraint location** コマンドで **resource-discovery** オプションを指定する場合の形式を示しています。このコマンドで **score** を正の値にした場合は、リソースを特定のノードで優先的に実行するように設定する基本的な場所の制約を指定し、**score** を負の値にした場合は、リソースがノードを回避するように設定する基本的な場所の制約に対応します。基本的な場所の制約の場合と同様に、これらの制約でもリソースの正規表現を使用できます。

```
pcs constraint location add id rsc node score [resource-discovery=option]
```

表10.2「リソース検出制約パラメーター」 では、リソース検出の制約を設定する基本パラメーターを説明します。

表10.2 リソース検出制約パラメーター

フィールド	説明
id	制約自体にユーザーが選択した名前
rsc	リソース名
node	ノード名

score	指定のリソースが指定のノードを優先するべきか回避するべきかを示す優先度を示す整数値。スコアが正の値の場合は、ノードを優先するようにリソースを設定する基本的な場所の制約となり、負の場合は、ノードを回避するようにリソースを設定する基本的な場所の制約となります。 score の **INFINITY** の値は、そのノードが利用可能な場合はそのノードでリソースを優先的に実行しますが、利用できない場合に別のノードでそのリソースを実行しないようにする訳ではありません。リソースがノードを回避するように設定するコマンドで **INFINITY** を設定した場合は、その他のノードが利用できない場合でも、そのリソースはそのノードでは実行されません。 数値スコア (**INFINITY** または **-INFINITY** 以外) は、制約がオプションで、それを上回る他の要因がない限り有効となることを意味します。たとえば、リソースが別のノードに置かれ、その **resource-stickiness** スコアが、場所制約のスコアよりも **優先** される場合、リソースはその場所に残されます。
resource-discovery オプション	* **always** - このノードに指定したリソースで、リソース検出を常に実行します。リソースの場所の制約の**resource-discovery** のデフォルト値です。 * **never** - このノードで、指定したリソースに対してリソース検出をまったく行いません。 * **exclusive** - このノード (および **exclusive** と同様にマーク付けされた他のノード) で指定したリソースに対してのみ、リソースの検出を行います。異なるノードにまたがった同じリソースの **exclusive** 検出を使用する複数の場所制約により、**resource-discovery** が排他的なノードのサブセットが作成されます。1つ以上のノードでリソースが **exclusive** 検出に対してマーク付けされた場合、リソースはノードのサブセット内のみに置くことが可能です。

警告

resource-discovery を **never** または **exclusive** に設定すると、Pacemaker が、想定されていない場所で実行している不要なサービスのインスタンスを検出して停止する機能がなくなります。関連するソフトウェアをアンインストールしたままにするなどして、リソース検出なしでサービスがノードでアクティブにならないようにすることは、システム管理者の責任です。

10.3. 場所の制約方法の設定

場所の制約を使用する場合は、リソースをどのノードで実行できるかを指定する一般的な方法を設定できます。

- オプトインクラスター - デフォルトでは、すべてのリソースをいずれのノードでも起動することができません。特定のリソースに対してノードを選択的に許可できるようにクラスターを設定します。
- オプトアウトクラスター - デフォルトでは、すべてのリソースをどのノードでも実行できます。リソースを特定のノードで実行しないように場所の制約を作成できるようにクラスターを設定します。

クラスターでオプトインまたはオプトアウトのどちらを選択するかは、独自に優先する設定やクラスターの構成により異なります。ほとんどのリソースをほとんどのノードで実行できるようにする場合は、オプトアウトを使用した方が設定しやすくなる可能性があります。ほとんどのリソースを、一部のノードでのみ実行する場合は、オプトインを使用した方が設定しやすくなる可能性があります。

10.3.1. 「オプトイン」クラスターの設定

オプトインクラスターを作成するには、クラスタープロパティー **symmetric-cluster** を **false** に設定し、デフォルトでは、リソースのいずれのノードも実行を許可しないようにします。

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

個々のリソースでノードを有効にします。以下のコマンドは、場所の制約を設定し、**Webserver** リソースは **example-1** ノードを優先し、**Database** リソースは **example-2** ノードを優先するようにし、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できるようにします。オプトインクラスターに場所の制約を設定する場合は、スコアをゼロに設定すると、リソースに対してノードの優先や回避を指定せずに、リソースをノードで実行できます。

```
# 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
```

10.3.2. 「オプトアウト」クラスターの設定

オプトアウトクラスターを作成するには、クラスタープロパティー **symmetric-cluster** を **true** に設定し、デフォルトで、すべてのノードでリソースの実行を許可します。これは、**symmetric-cluster** が設定されていない場合のデフォルト設定です。

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

以下のコマンドを実行すると、「「オプトイン」クラスターの設定」 の例と同じ設定になります。すべてのノードのスコアは暗黙的に 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、スコアに INFINITY を指定する必要はないことに注意してください。INFINITY は、スコアのデフォルト値です。

第11章 クラスターリソースの実行順序の決定

リソースが実行する順序を指定するために、順序の制約を設定できます。

以下は、順序の制約を設定するコマンドの形式です。

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

表11.1「順序の制約のプロパティー」 では順序の制約を設定する場合のプロパティーとオプションを簡単に説明します。

表11.1 順序の制約のプロパティー

フィールド	説明
resource_id	動作が実行するリソースの名前。
action	リソースで実行する動作。*action* プロパティーでは、以下の値が使用できます。 * **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 動作を指定しない場合のデフォルトの動作は **start** です。
kind オプション	制約の実施方法。**kind** オプションでは、以下の値が使用できます。 * **Optional** - 両方のリソースが指定の動作を実行する場合のみ適用されます。オプションの順序の詳細は「勧告的な順序付けの設定」を参照してください。 * **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は、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。この強制的な順序付けの詳細は「強制的な順序付けの設定」を参照してください。 * **Serialize** - 指定するリソースに対して、複数の停止または起動のアクションが同時に発生しないようにします。指定した1つ目および 2 つ目のリソースは、いずれの順序で起動できますが、片方が完了しないともう片方が開始しません。典型的なユースケースは、ホストでリソースの起動により負荷が高くなることです。

フィールド	説明
symmetrical オプション	true にすると、逆の制約は、逆のアクションに適用されます (たとえば A が開始した後に B が開始し、**kind** が **Serialize** の順序制約が対象になる前に B が停止します)。 **kind** が **Serialize** である順序制約を対称にすることはできません。**Mandatory** および **Ordering** のデフォルト値は **true** で、**Serialize** の場合は **false** になります。

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が削除されます。

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

11.1. 強制的な順序付けの設定

必須順序制約は、最初のリソースに対する最初のアクションが正常に完了しない限り、2 番目のリソースの 2 番目のアクションが開始しないことを示しています。命令できるアクションが、**stop** または **start** で、昇格可能なクローンが **demote** および **promote** とします。たとえば、「A then B」(「start A then start B」と同等) は、A が適切に開始しないと、B が開始しないことを示しています。順序の制約は、この制約の **kind** オプションが **Mandatory** となっているか、デフォルトのままに設定されている場合は必須になります。

symmetrical オプションを **true** またはデフォルトのままにすると、逆の作用は逆に命令されます。**start** と **stop** のアクションは対象的になり、**demote** と **promote** は対照的になります。たとえば、対照的に、「promote A then start B」順序は「stop B then demote A」(B が正常に停止するまで A が降格しない) ことを示しています。対象順序は、A' の状態を変更すると、B に予定されているアクションが発生します。たとえば、「A then B」と設定すると、失敗により A が再起動すると、B が最初に停止してから、A が停止し、それにより A が開始し、それにより B が開始することを示します。

クラスターは、それぞれの状態変化に対応することに注意してください。 2 番目のリソースで停止操作を開始する前に 1 番目のリソースが再起動し、起動状態にあると、2 番目のリソースを再起動する必要はありません。

11.2. 勧告的な順序付けの設定

順序の制約に **kind=Optional** オプションを指定すると、制約はオプションと見なされ、両方のリソースが指定の動作を実行する場合にのみ適用されます。1 番目に指定するリソースの状態の変更は、2 番目に指定するリソースには影響しません。

次のコマンドは、**VirtualIP** と **dummy_resource** という名前のリソースに、勧告的な順序の制約を設定します。

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

11.3. リソースセットへの順序の設定

一般的に、管理者は、複数のリソースを作成する際に順序を設定します (例: リソース A が開始してからリソース B を開始し、その後にリソース C を開始)。複数のリソースを作成して同じ場所に配置し (コロケーションを指定)、起動の順序を設定する必要がある場合は、「リソースグループの設定」に従って、これらのリソースが含まれるリソースグループを設定できます。

ただし、特定の順序で起動する必要があるリソースをリソースグループとして設定することが適切ではない場合があります。

- リソースを順番に起動するように設定する必要があるものの、リソースは必ずしも同じ場所に配置しない場合。
- リソース C の前にリソース A または B のいずれかが起動する必要があるものの、A と B の間には関係が設定されていない場合。
- リソース C およびリソース D の前にリソース A およびリソース B の両方が起動している必要があるものの、A と B、または C と D の間には関係が設定されていない場合。

このような状況では、**pcs constraint order set** コマンドを使用して、リソースの 1 つまたは複数のセットに対して順序の制約を作成できます。

pcs constraint order set コマンドを使用して、リソースセットに以下のオプションを設定できます。

- **sequential** - リソースセットに順序を付ける必要があるかどうかを指定します。**true** または **false** に設定できます。デフォルト値は **true** です。
  sequential を **false** に設定すると、セットのメンバーに順序を設定せず、順序の制約にあるセット間で順序付けできます。そのため、このオプションは、制約に複数のセットが登録されている場合に限り有効です。それ以外の場合、制約の効果はありません。
- **require-all** - 続行する前にセットの全リソースがアクティブである必要があるかどうかを指定します。**true** または **false** に設定できます。**require-all** を **false** に設定した場合は、そのセットのリソースが 1 つだけ起動していれば、次のセットに移行します。**require-all** を **false** に設定しても、**sequential** が **false** に設定された、順序付けがないセットと併用しないと効果がありません。デフォルト値は **true** です。
- **action** - 「順序の制約のプロパティー」で説明しているように **start**、**promote**、 **demote** または **stop** に設定できます。
- **role** - **Stopped**、**Started**、**Master**、または **Slave** に設定できます。

pcs constraint order set コマンドの **setoptions** パラメーターに続いて、リソースのセットに対する以下の制約オプションを設定できます。

- **id** - 定義する制約の名前を指定します。
- **kind** - 「順序の制約のプロパティー」で説明しているように、制約を強制する方法を示します。
- **symmetrical** - 「順序の制約のプロパティー」 で説明しているように、逆の制約が逆の作用に適用するかどうかを設定します。

```
pcs constraint order set resource1 resource2 [resourceN]... [options] [set resourceX resourceY ... [options]] [setoptions [constraint_options]]
```

D1、**D2**、**D3** という 3 つのリソースがある場合に、次のコマンドにより、この 3 つのリソースを、順序を指定したリソースセットとして設定します。

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

この例では、**A**、**B**、**C**、**D**、**E**、および **F** という名前の 6 つのリソースがある場合に、以下のように、起動するリソース設定に対する順序制約を設定します。

- **A** と **B** は、互いに独立して起動します。
- **A** または **B** のいずれかが開始すると、**C** が開始します。
- **C** が開始すると、**D** が開始します。
- **D** が開始したら、**E** と **F** が互いに独立して起動します。

symmetrical=false が設定されているため、リソースの停止は、この制約の影響を受けません。

```
# pcs constraint order set A B sequential=false require-all=false set C D set E F
sequential=false setoptions symmetrical=false
```

第12章 クラスターリソースのコロケーション

1 つのリソースの場所を別のリソースの場所に依存させるように指定するには、コロケーションの制約を設定します。

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には重要な副次的な影響を及ぼす点に注意してください。まず、これはノードにリソースを割り当てる順序に影響します。リソース B の場所を把握していない場合は、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 に対してコロケーションするのか、またはリソース B をリソース A に対してコロケーションするのかを考慮する必要があります。

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もう 1 つの点として、リソース A をリソース B に対してコロケーションすると仮定した場合は、クラスターがリソース B に選択するノードを決定する際、リソース A の優先度も考慮に入れます。

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]
```

表12.1「コロケーション制約のプロパティー」 は、コロケーション制約を設定するのに使用するプロパティーおよびオプションを簡単に説明しています。

表12.1 コロケーション制約のプロパティー

フィールド	説明
source_resource	コロケーションソース。制約の条件を満たさない場合、クラスターではこのリソースの実行を許可しないことを決定する可能性があります。
target_resource	コロケーションターゲット。クラスターではこのリソースの配置先を最初に決定してから、ソースリソースの配置先を決定します。
score	正の値を指定するとリソースが同じノードで実行し、負の値を指定すると同じノードで実行しなくなります。デフォルト値である +**INFINITY** を指定すると、*source_resource* は必ず *target_resource* と同じノードで実行されます。値が -**INFINITY** の場合は、*source_resource* を *target_resource* が同じノードで実行されなくなります。

12.1. リソースの強制的な配置の指定

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** では、**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必要がある場合は、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 (理由はともあれ) **myresource1** の実行は許可されません。

または、逆の設定、つまり **myresource1** が **myresource2** と同じマシンでは実行されないように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 with myresource2 score=-INFINITY
```

-INFINITY を指定することで、制約は結合しています。このため、実行できる場所として残っているノードで **myresource2** が実行している場合、**myresource1** はいずれのノードでも実行できなくなります。

12.2. リソースの勧告的な配置の指定

「強制的な配置」が必ず実行する (または必ず実行しない) 場合の設定であれば、「勧告的な配置」は、ある状況下で優先される設定を指します。制約のスコアが **-INFINITY** より大きく **INFINITY** より小さい場合、クラスターはユーザーの希望を優先しようとしますが、クラスターリソースを一部停止することを希望する場合は無視します。勧告的なコロケーション制約と設定の他の要素を組み合わせると、強制的であるように動作させることができます。

12.3. 複数リソースのコロケーション

使用する設定で、コロケーションと順序が適用されるリソースのセットを作成する必要がある場合は、「リソースグループの設定」 に従って、これらのリソースを含むリソースグループを設定できます。ただし、コロケーションする必要があるリソースをリソースグループとして設定することが適切ではない場合があります。

- リソースのセットをコロケーションする必要があるものの、リソースが必ずしも順番に起動する必要がない場合。
- リソース C を、リソース A またはリソース B のいずれかとコロケーションする必要があるものの、リソース A とリソース B との間に関係が設定されていない場合。
- リソース C およびリソース D を、リソース A およびリソース B の両方とコロケーションする必要があるものの、A と B の間、または C と D の間に関係が設定されていない場合。

このような状況では、**pcs constraint colocation set** コマンドを使用して、リソースの 1 つまたは複数のセットでコロケーションの制約を作成できます。

pcs constraint colocation set コマンドを使用すると、リソースのセットに対して以下のオプションを設定できます。

- **sequential** - セットのメンバーで相互のコロケーションが必要であるかどうかを指定します。**true** または **false** に設定できます。
  sequential を **false** に設定すると、このセットのメンバーがアクティブであるかどうかに関係なく、このセットのメンバーを、制約の中で、このセットの後にリストされている他のセットとコロケーションできます。そのため、このオプションは制約でこのセットの後に他のセットが指定されている場合限りに有効です。他のセットが指定されていない場合は、制約の効果がありません。
- **role** - **Stopped**、**Started**、**Master**、または **Slave** に設定できます。

pcs constraint colocation set コマンドの **setoptions** パラメーターの後に、リソースのセットに対する以下の制約オプションを設定できます。

- **id** - 定義する制約の名前を指定します。
- **score** - 制約の優先度を示します。このオプションの詳細は 「単純な場所の制約オプション」を参照してください。

セットのメンバーをリストすると、各メンバーは、自身の前のメンバーとコロケーションされます。たとえば、「set A B」は、B が A とコロケーションされることを意味します。また、複数のセットをリストする場合は、各セットがその後のメンバーとコロケーションされます。たとえば、「set C D sequential=false set A B」は、C と D のセットが、 A と B のセットとコロケーションされることを意味します (ただし、C と D には関係がなく、B は A とコロケーションされます)。

以下のコマンドは、リソースのセットにコロケーションの制約を作成します。

```
pcs constraint colocation set resource1 resource2 [resourceN]... [options] [set resourceX resourceY ... [options]] [setoptions [constraint_options]]
```

12.4. コロケーション制約の削除

コロケーション制約を削除する場合は、source_resource を指定して以下のコマンドを使用します。

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

第13章 リソース制約の表示

設定した制約を表示するには、いくつかのコマンドを使用できます。

13.1. すべての設定済みの制約の表示

以下のコマンドは、現在の場所、順序、ロケーションの制約をすべて表示します。

```
pcs constraint list|show
```

13.2. 場所の制約の表示

以下のコマンドは、現在の場所の制約を一覧表示します。

- **resources** を指定すると、リソース別に場所の制約が表示されます。これはデフォルトの動作です。
- **nodes** を指定すると、ノード別に場所の制約が表示されます。
- 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 [resource...]] | [nodes [node...]]] [--full]
```

13.3. 順序の制約の表示

以下のコマンドは、現在の順序の制約をすべて表示します。**--full** オプションを指定すると、内部の制約 ID を表示します。

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

13.4. コロケーション制約の表示

以下のコマンドは、現在のすべての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 が表示されます。

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

13.5. リソース固有の制約の表示

以下のコマンドは、特定リソースを参照する制約を一覧表示します。

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

第14章 ルールによるリソースの場所の決定

さらに複雑な場所の制約には、Pacemaker のルールを使用してリソースの場所を決定できます。

14.1. PACEMAKER ルール

ルールは、設定をより動的にするのに使用できます。ルールには、(ノード属性を使用して) 時間ベースで異なる処理グループにマシンを割り当て、場所の制約の作成時にその属性を使用する方法があります。

各ルールには、日付などのさまざまな式だけでなく、その他のルールも含めることができます。ルールの **boolean-op** フィールドに応じて各種の式の結果が組み合わされ、最終的にそのルールが **true** または **false** のどちらに評価されるかどうかが決まります。次の動作は、ルールが使用される状況に応じて異なります。

表14.1 ルールのプロパティー

フィールド	説明
role	リソースが指定のロールにある場合にのみ適用するルールを制限します。使用できる値は **Started**、**Slave**、および **Master** です。**role="Master"** が指定されたルールは、クローンインスタンスの最初の場所を判断できないことに注意してください。どのアクティブなインスタンスが昇格されるかにのみ影響します。
score	ルールが **true** に評価される場合に適用されるスコア。場所の制約として、ルールでの使用に制限されます。
score-attribute	ルールが **true** に評価されると検索し、スコアとして使用するノード属性。場所の制約として、ルールでの使用に制限されます。
boolean-op	複数の式オブジェクトからの結果を組み合わせる方法、使用できる値は **and** および **or** です。デフォルト値は **and** です。

14.1.1. 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使用されるノード属性の式です。

表14.2 式のプロパティー

フィールド	説明
attribute	テストするノード属性

フィールド	説明
type	値をテストする方法を判別します。使用できる値は、**string**、**integer**、 **version** です。デフォルト値は **string** です。
operation	実行する比較動作。以下の値が使用できます。 * **lt** - ノード属性の値が **value** 未満の場合は True * **gt** - ノード属性の値が **value** を超える場合は True * **lte** - ノード属性の値が **value** 未満または同等になる場合は True * **gte** - ノード属性の値が **value** を超えるか、または同等になる場合は 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は 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は True
value	比較のためにユーザーが提供した値 (**operation** が **defined** または **not_defined** に設定されていない場合に限り必要)

管理者が追加する属性のほかに、表14.3「組み込みノード属性」で説明されているように、クラスターは使用可能な各ノードに特殊な組み込みノード属性を定義します。

表14.3 組み込みノード属性

名前	説明
#uname	ノード名
#id	ノード ID
#kind	ノードタイプ、使用できる値は **cluster**、**remote**、および **container** です。 **kind** の値は、**ocf:pacemaker:remote** リソースで作成された Pacemaker リモートノードの **remote**、および Pacemaker リモートゲストノードおよびバンドルノードの **container** です。

名前	説明
#is_dc	このノードが指定コントローラー (DC) の場合は **true**、それ以外の場合は **false**
#cluster_name	**cluster-name** クラスタープロパティーの値 (設定されている場合)
#site_name	**site-name** ノード属性の値 (設定されている場合)、それ以外は **#cluster-name** と同じ。
#role	関連する昇格可能なクローンがこのノードで果たす役割。昇格可能なクローンの場所の制約のルール内でのみ有効です。

14.1.2. 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めることができます。

表14.4 日付の式のプロパティー

フィールド	説明
start	ISO 8601 仕様に準じた日付と時刻。
end	ISO 8601 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 と end のいずれかの日付、または両方の日付と比較します。使用できる値は次のとおりです。 * **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、かつ **end** 以前の場合は True * **date-spec** - 現在の日付/時刻に対して cron と同様の比較を実行します。

14.1.3. 日付の詳細

日付の詳細は、時間に関係する cron と同様の式を作成するのに使用されます。各フィールドには 1 つの数字または範囲が含まれます。指定のないフィールドは無視され、デフォルトの 0 としてみなされません。

たとえば、**monthdays="1"** は各月の最初の日と一致し、**hours="09-17"** は午前 9 時から午後 5 時までの時間と一致しますが、複数の範囲が含まれる **weekdays="1,2"** や **weekdays="1-2,5-6"** は指定できません。

表14.5 日付詳細のプロパティー

フィールド	説明
id	日付の一意の名前
hours	使用できる値 - 0~23
monthdays	使用できる値 - 0~31（月と年に応じて異なる）
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 (年に応じて異なる)
months	使用できる値 - 1~12
weeks	使用できる値 - 1~53 (**weekyear** に応じて異なる)
years	グレゴリオ暦 (新暦) に準じる年
weekyears	グレゴリオ暦の年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)。

14.2. ルールを使用した PACEMAKER の場所の制約の設定

以下のコマンドを使用して、ルールを使用する Pacemaker 制約を使用します。**score** が省略される場合は、デフォルトで INFINITY に設定されます。**resource-discovery** が省略される場合は、デフォルトで **always** に設定されます。

resource-discovery オプションの詳細は「ノードのサブセットへリソース検出の制限」を参照してください。

基本的な場所の制約と同様に、これらの制約にリソースの正規表現を使用することもできます。

ルールを使用して場所の制約を設定する場合、**score** は正または負の値にすることができ、正の値は「prefers」および負の値は「avoids」を示します。

> pcs constraint location **rsc** rule [resource-discovery=**option**] [role=master|slave] [score=**score** | score-attribute=**attribute**] **expression**

expression オプションは以下のいずれかにできます。**duration_options** および **date_spec_options** は、「日付詳細のプロパティー」で説明されているように、hours、monthdays、weekdays、yeardays、months、weeks、years、weekyears、moon になります。

- **defined|not_defined attribute**
- **attribute lt|gt|lte|gte|eq|ne [string|integer|version] value**

- **date gt|lt date**
- **date in_range date to date**
- **date in_range date to duration duration_options …**
- **date-spec date_spec_options**
- **expression and|or expression**
- **(expression)**

持続時間は、計算により **in_range** 操作の終了を指定する代替方法です。たとえば、19 カ月間を期間として指定できます。

以下の場所の制約は、現在が 2018 年の任意の時点である場合に true の式を設定します。

```
# pcs constraint location Webserver rule score=INFINITY date-spec years=2018
```

以下のコマンドは、月曜日から金曜日までの 9 am から 5 pm までが true となる式を設定します。hours の値 16 には時間の値が一致する 16:59:59 までが含まれます。

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

以下のコマンドは、 13 日の金曜日が満月であると true になる式を設定します。

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

ルールを削除するには、以下のコマンドを使用します。削除してい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ます。

```
pcs constraint rule remove rule_id
```

第15章 クラスターリソースの管理

本セクションでは、クラスターリソースを管理するのに使用できるさまざまなコマンドを説明します。

15.1. 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 status
```

たとえば、システムを設定していたリソースの名前が **VirtualIP** と **WebSite** の場合は、**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 status
 VirtualIP (ocf::heartbeat:IPaddr2): Started
 WebSite (ocf::heartbeat:apache): Started
```

設定したリソースの一覧と、そのリソースに設定したパラメーターを表示するには、以下のように、**pcs resource config** コマンドの **--full** オプションを使用します。

```
# pcs resource config
 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 config resource_id
```

たとえば、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ます。

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

15.2. リソースパラメーターの修正

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]
```

以下のコマンドシーケンスでは、リソース **VirtualIP** に設定したパラメーターの値を表示するコマンドとその出力内容と、**ip** パラメーターの値を変更するコマンドと、変更したパラメーターを表示するコマンドとその出力内容を示しています。

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

15.3. クラスターリソースの障害ステータスの解除

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ときに障害メッセージが表示されます。このリソースを解決する場合は、**pcs resource cleanup** コマンドで障害状態を消去できます。このコマンドは、リソースの状態と **failcount** をリセットし、リソースの動作履歴を消去して現在の状態を再検出するようクラスターに指示します。

次のコマンドは、**resource_id** で指定したリソースをクリーンアップします。

```
pcs resource cleanup resource_id
```

resource_id を指定しないと、このコマンドは、全リソースのリソース状態と **failcount** をリセットします。

pcs resource cleanup コマンドは、失敗したアクションとして表示されるリソースのみを検証します。全ノードの全リソースを調査するには、次のコマンドを入力します。

```
pcs resource refresh
```

デフォルトでは、**pcs resource refresh** コマンドは、リソースのステータスが分かっているノードだけを検証します。ステータスがわからないすべてのリソースを検証するには、以下のコマンドを実行します。

```
pcs resource refresh --full
```

15.4. クラスター内のリソースの移動

Pacemaker は、リソースを別のノードに移動するように設定し、必要に応じて手動でリソースを移動するように設定するさまざまなメカニズムを提供します。

「クラスターリソースの手動による移行」 に従って、**pcs resource move** コマンドと **pcs resource relocate** コマンドで、クラスターのリソースを手動で移動します。

このコマンドの他にも、「クラスターリソースの有効化、無効化、および禁止」 に従ってリソースを有効、無効、および禁止にすることで、クラスターリソースの挙動を制御することもできます。

失敗した回数が、定義した値を超えると、新しいノードに移動し、外部接続が失われた時にリソースを移動するようにクラスターを設定できます。

15.4.1. 障害発生によるリソースの移動

リソースの作成時に、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。

- 管理者が **pcs resource cleanup** コマンドを使用して、リソースの **failcount** を手動でリセットします。
- リソースの **failure-timeout** 値に到達します。

デフォルトで **migration-threshold** の値が **INFINITY** に設定されています。**INFINITY** は、内部的に非常に大きく有限の数として定義されます。0 にすると、**migration-threshold** 機能を無効にします。

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を設定するのは異なります。

次の例では **dummy_resource** というリソースに、移行しきい値 10 を追加します。この場合は、障害が 10 回発生すると、そのリソースが新しいノードに移動します。

```
# 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できます。

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

リソースの現在の障害ステータスと制限を確認するには、**pcs resource failcount show** コマンドを使用します。

移行しきい値の概念には、リソース起動の失敗とリソース停止の失敗の 2 つの例外があります。クラスタープロパティー **start-failure-is-fatal** が **true** に設定された場合 (デフォルト)、起動の失敗により **failcount** が **INFINITY** に設定され、リソースが常に即座に移動するようになります。

停止時の失敗は、起動時とは若干異なり、重大になります。リソースの停止に失敗し STONITH が有効になっている場合は、リソースを別のノードで起動できるように、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。

15.4.2. 接続状態の変更によるリソースの移動

以下の 2 つのステップに従って、外部の接続が失われた場合にリソースが移動するようにクラスターを設定します。

1. **ping** リソースをクラスターに追加します。**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用して、マシン (DNS ホスト名または IPv4/IPv6 アドレスにより指定される) にアクセス可能であるかをテストし、その結果を使用して **pingd** と呼ばれるノード属性を維持します。
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9.1「リソースエージェント識別子」 では、▌**ping** リソースに設定できるプロパティを示します。

表15.1 ping リソースのプロパティ

フィールド	説明

フィールド	説明
dampen	今後の変更が発生するまでに待機する (dampen) 時間。これにより、クラスターノードが、わずかに異なる時間に接続が失われたことに気が付いたときに、リソースがバウンスされなくなります。
multiplier	接続された ping ノードの数は、ノードの数にこの値を掛けてスコアを取得します。複数の ping ノードが設定された場合に便利です。
host_list	現在の接続状態を判断するために接続するマシン。許可される値には、解決可能な DNS ホスト名、IPv4 および IPv6 のアドレスが含まれます。ホストリストのエントリーはスペースで区切られます。

次のコマンド例は、**gateway.example.com** への接続を検証する **ping** リソースを作成します。実際には、ネットワークゲートウェイやルーターへの接続を検証します。リソースがすべてのクラスターノードで実行するように、**ping** リソースを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=gateway.example.com clone
```

以下の例は、**Webserver** という既存のリソースの場所制約ルールを設定します。これにより、**Webserver** リソースが現在実行しているホストが **gateway.example.com** へ ping できない場合に、 **Webserver** リソースを **www.example.com** へ ping できるホストに移動します。

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

```
 Module included in the following assemblies:
//
// <List assemblies here, each on a new line>
// rhel-8-docs/enterprise/assemblies/assembly_managing-cluster-resources.adoc
```

15.5. モニター操作の無効化

モニタリングが再実行しないようにする最も簡単な方法は、モニタリングを削除することです。ただし、削除するのではなく、一時的に無効にしたい場合があります。このような場合は、操作の定義に **enabled="false"** を追加します。モニタリング操作を再度有効にするには、操作の定義に **enabled="true"** を設定します。

第16章 複数のノードでアクティブなクラスターリソース (クローンリソース) の作成

クラスターリソースが複数のノードでアクティブになるように、リソースのクローンを作成できます。たとえば、クローンにしたリソースを使用して、IP リソースの複数のインスタンスを設定し、クラスター全体でノードを分散できます。リソースエージェントが対応するリソースはすべてクローンできます。クローンは、1つのリソースまたは1つの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。

16.1. クローンリソースの作成および削除

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 [resource options] [meta resource meta options] clone [clone options]
```

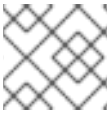
クローンの名前は **resource_id-clone** となります。

1つのコマンドで、リソースグループの作成と、リソースグループのクローン作成の両方を行うことはできません。

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** となります。



注記

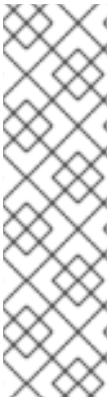
リソース設定の変更が必要なのは、1つのノードのみです。

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、そのクローンには、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** である **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
```



注記

あるクローンを、別のクローンの後にくるように作成する場合は、多くの場合 **interleave=true** オプションを設定する必要があります。これにより、依存元のクローンが停止または開始した時に、依存しているクローンのコピーを停止または開始できます。このオプションを設定しない場合は、次のようになります。クローンリソース B がクローンリソース A に依存している場合に、ノードがクラスターを去ってから戻ってきて、そのノードでリソース A が起動すると、リソース B の全コピーが、その全ノードで再起動します。なぜなら、依存しているクローンリソースに **interleave** オプションが設定されていないため、そのリソースの全インスタンスは、そのリソースが依存するリソースの実行インスタンスに依存するためです。

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
```

クローンのリソースに指定できるオプションを 表16.1「クローンのリソース用オプション」 に示します。

表16.1 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	表9.3「リソースのメタオプション」 の説明どおり、クローンされたリソースから継承されるオプション。
clone-max	起動するリソースのコピーの数。デフォルトは、クラスター内のノード数です。
clone-node-max	1 つのノードで起動できるリソースのコピー数。デフォルト値は **1** です。
notify	クローンのコピーを停止したり起動したりする時に、他のコピーに、事前およびアクセスが成功した時に通知します。使用できる値は **false**、**true** です。デフォルトの値は **false** です。
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、使用できる値は **false** および **true** です。 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すべてのノードで同じ動作を行うため、1 台のマシンごとに実行できるクローンのコピーは 1 つです。 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が、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りません。**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、小さい場合は **false** がデフォルト値になります。

フィールド	説明
ordered	コピーを、(並列ではなく) 連続して開始する必要があります。使用できる値は **false** および **true** です。デフォルト値は **false** です。
interleave	(2 番目のクローンの全インスタンスが起動または停止するまで待つのではなく) 2 番目のクローンと同じノードにあるコピーが起動または停止するとすぐに、最初のクローンのコピーが起動または停止するように、(クローン間の) 順序付けの制約の動作を変更します。使用できる値は **false** および **true** です。デフォルト値は **false** です。
clone-min	値を指定した場合は、**interleave** オプションが **true** に設定されていても、この後に順序付けされたクローンは、元のクローンに指定した数だけインスタンスが実行するまで起動できません。

安定した割り当てパターンを実戦するために、クローンは、デフォルトでわずかに固定されています。これは、実行しているノードにとどまることをやや優先することを示してします。**resource-stickiness** の値が提供されていない場合に、クローンが使用する値は 1 となります。値が小さければ、他のリソースのスコア計算にはほとんど影響がありませんが、Pacemaker がクラスター内でコピーを不要に移動するのを防ぐには十分です、**resource-stickiness** リソースのメタオプションを設定する方法は「リソースのメタオプションの設定」を参照してください。

16.2. クローンリソース制約の表示

ほとんどの場合は、アクティブなクラスターノードに対するクローンのコピーはひとつです。ただし、**clone-max** には、リソースのクローン数の値として、クラスター内のノード合計数より小さい数を設定できます。この場合は、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
```

順序制約の動作はクローンでは若干異なります。以下の例では、開始される必要がある **webfarm-clone** のコピーがすべて開始されるまで待機してから、**webfarm-stats** が開始します。**webfarm-clone** のコピーを 1 つも開始できない場合、**webfarm-stats** はアクティブになりません。さらに、**webfarm-stats** が停止するまで待機してから **webfarm-clone** が停止します。

```
# 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している複数マシンのどこから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。

以下のコマンドは、コロケーション制約を作成し、**webfarm-stats** リソースが **webfarm-clone** のアクティブなコピーと同じノードで実行するようにします。

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

16.3. 昇格可能なクローンリソースの作成

昇格可能なクローンリソースは、**promotable** メタ属性が **true** に設定されているクローンリソースです。昇格可能なクローンリソースにより、インスタンスの操作モードは、**Master** および **Slave** のいずれかにできます。モードの名前に特別な意味はありませんが、インスタンスの起動時に **Slave** 状態にならなければならない制限があります。

16.3.1. 昇格可能なリソースの作成

次のコマンドを実行すると、リソースを昇格可能なクローンとして作成できます。

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

昇格可能なクローンの名前は **resource_id-clone** となります。

また、以下のコマンドを使用して、作成済みのリソースまたはリソースグループから昇格可能なリソースを作成することもできます。昇格可能なクローンの名前は、**resource_id-clone** または **group_name-clone** になります。

```
pcs resource promotable resource_id [clone options]
```

昇格可能なリソースに指定できる追加クローンオプションを 表16.2「昇格可能なクローンに利用できる追加のクローンオプション」 に示します。

表16.2 昇格可能なクローンに利用できる追加のクローンオプション

フィールド	説明
promoted-max	昇格できるリソースのコピー数。デフォルト値は 1 です。
promoted-node-max	1 つのノードで昇格できるリソースのコピー数。デフォルト値は 1 です。

16.3.2. 昇格可能なリソース制約の表示

ほとんどの場合、昇格可能なリソースにはアクティブなクラスターノードごとに 1 つのコピーがあります。そうではない場合は、リソースの場所制約を使用して、クラスターが優先的にコピーを割り当てるノードを指定できます。これらの制約は、通常のリソースと同様に記述されます。

リソースのロール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**]

コロケーション制約の詳細は「クラスターリソースのコロケーション」を参照してください。

昇格可能なリソースが含まれる順序制約を設定する場合は、リソースに指定できるアクションの 1 つがリソースのスレーブロールからマスターへロールの昇格を指定する **promote** です。さらに、**demote** を指定するとリソースをマスターからスレーブに降格できます。

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の詳細は「クラスターリソースの実行順序の決定」を参照してください。

第17章 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を説明します。

17.1. 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、**--all** オプションを使用すると、全ノードのクラスターサービスが停止します。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します。

```
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
```

17.2. クラスターサービスの有効化および無効化

次のコマンドを使用して、クラスターサービスを有効にします。これにより、指定した1つまたは複数のノードで起動時にクラスターサービスが実行するように設定されます。有効にすると、ノードでフェンスされたあとにクラスターが自動的に再参加するようになり、クラスターが最大強度を下回る時間が最小限に抑えられます。クラスターサービスが有効になっていない場合、管理者は、手動でクラスターサービスを開始する前に問題を調査して、ハードウェアの問題が発生したノードで再度問題が発生する可能性がある場合は、クラスターが戻さないようにできます。

- **--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] [...]
```

17.3. クラスターノードの追加

注記

運用保守期間中に、既存のクラスターにノードを追加することが強く推奨されます。これにより、新しいノードとそのフェンシング設定に対して、適切なリソースとデプロイメントのテストを実行できます。

既存クラスターに新しいノードを追加する場合は、以下の手順に従ってください。この手順

は、**corosync** を実行している標準クラスターを追加します。corosync 以外のノードをクラスターに統合する方法は「corosync 以外のノードのクラスターへの統合: pacemaker_remote サービス」を参照してください。

この例では、既存のクラスターノードは **clusternode-01.example.com**、**clusternode-02.example.com**、および **clusternode-03.example.com** になります。新たに追加するノードは **newnode.example.com** になります。

クラスターに追加する新しいノードで、以下の作業を行います。

1. クラスターパッケージをインストールします。クラスターで SBD、Booth チケットマネージャー、またはクォーラムデバイスを使用する場合は、対応するパッケージ (**sbd**、**booth-site**、**corosync-qdevice**) を、新しいノードにもインストールする必要があります。

   ```
   [root@newnode ~]# yum install -y pcs fence-agents-all
   ```

   クラスターパッケージに加えて、クラスターで実行しているすべてのサービスをインストールして構成する必要もあります。これは、既存のクラスターノードにインストールしていました。たとえば、Red Hat の高可用性クラスターで Apache HTTPサーバーを実行している場合は、追加するノードにサーバーをインストールする必要があります。また、サーバーのステータスをチェックする **wget** ツールも必要です。

2. **firewalld** デーモンを実行している場合は、以下のコマンドを実行して、Red Hat High Availability Add-On に必要なポートを有効にします。

   ```
   # firewall-cmd --permanent --add-service=high-availability
   # firewall-cmd --add-service=high-availability
   ```

3. ユーザー ID **hacluster** のパスワードを設定します。クラスターの各ノードに同じパスワードを使用することが推奨されます。

   ```
   [root@newnode ~]# passwd hacluster
   Changing password for user hacluster.
   New password:
   Retype new password:
   passwd: all authentication tokens updated successfully.
   ```

4. 次のコマンドを実行して **pcsd** サービスを開始し、システムの起動時に **pcsd** が有効になるようにします。

   ```
   # systemctl start pcsd.service
   # systemctl enable pcsd.service
   ```

既存クラスターのノードの 1 つで、以下の作業を行います。

1. 新しいクラスターノードでユーザー **hacluster** を認証します。

   ```
   [root@clusternode-01 ~]# pcs host auth newnode.example.com
   Username: hacluster
   Password:
   newnode.example.com: Authorized
   ```

2. 新しいノードを既存のクラスターに追加します。さらに、このコマンドは **corosync.conf** クラスター設定ファイルを、クラスターのすべてのノード (追加する新しいノードを含む) に対して同期します。

```
[root@clusternode-01 ~]# pcs cluster node add newnode.example.com
```

クラスターに追加する新しいノードで、以下の作業を行います。

1. 新しいノードで、クラスターサービスを開始および有効化します。

```
[root@newnode ~]# pcs cluster start
Starting Cluster...
[root@newnode ~]# pcs cluster enable
```

2. 新しいクラスターノードに、フェンシングデバイスを設定してテストするようにしてください。

17.4. クラスターノードの削除

次のコマンドは、指定したノードをシャットダウンして、クラスターにあるその他のすべてのノードで、クラスターの設定ファイル **corosync.conf** からそれを削除します。

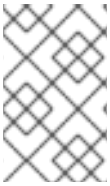
```
pcs cluster node remove node
```

第18章 PACEMAKER クラスターのプロパティー

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。

18.1. 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18.1「クラスターのプロパティー」 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では、デフォルト値を別の値には変更しないことが推奨されます。

表18.1 クラスターのプロパティー

オプション	デフォルト	説明
batch-limit	0	クラスターを並列に実行できるリソースアクションの数。「正しい」値は、ネットワークおよびクラスターノードの速度および負荷によって異なります。デフォルト値の 0 は、任意のノードで CPU の負荷が高い場合に動的に制限を課すことを意味します。
migration-limit	-1 (無制限)	クラスターは、ノードで並行に実行するように許可されている移行ジョブの数。
no-quorum-policy	stop	クラスターにクォーラムがない場合のアクション。使用できる値は以下のようになります。 * 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	リソースは、デフォルトで任意のノードで実行できるかどうかを示します。

オプション	デフォルト	説明
stonith-enabled	true	失敗したノードと、停止できないリソースがあるノードにフェンスが必要であることを示します。データを保護するには、**true** に設定する必要があります。 **true** または未設定の場合は、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ます。 Red Hat は、この値を **true** に設定したクラスターだけをサポートします。
stonith-action	reboot	STONITH デバイスに送るアクション。使用できる値は **reboot**、**off** です。**poweroff** 値も許可されていますが、レガシーデバイスでのみ使用されます。
cluster-delay	60s	(アクションの実行の除く) ネットワークのラウンドトリップ遅延です。「正しい」値は、ネットワークおよびクラスタノードの速度と負荷によって異なります。
stop-orphan-resources	true	削除されたリソースを停止すべきかどうかを示します。
stop-orphan-actions	true	削除されたアクションをキャンセルするかどうかを示します。
start-failure-is-fatal	true	特定ノードのリソースを開始できない障害により、そのノードで開始試行を行わないようにするかを示します。**false** に設定すると、クラスターは、同じノードで開始するかどうかを、リソースが失敗した回数と、移行しきい値により設定します。リソースに **migration-threshold** オプションを設定する方法は「リソースのメタオプションの設定」を参照してください。 **start-failure-is-fatal** を **false** に設定すると、リソースを起動できない障害があるノードが、すべての依存アクションを遅らせる可能性があるというリスクが発生します。これにより、**start-failure-is-fatal** のデフォルトは true となっています。**start-failure-is-fatal=false** を設定するリスクは、移行のしきい値を下げることで、何度も失敗しても他のアクションは発生させることができるようにすることげ軽減できます。

オプション	デフォルト	説明
pe-error-series-max	-1 (all)	「ERROR」となるスケジューラー入力の保存する数。問題を報告する場合に使用されます。
pe-warn-series-max	-1 (all)	「WARNING」となるスケジューラー入力の保存する数。問題を報告する場合に使用されます。
pe-input-series-max	-1 (all)	保存する「normal」のスケジューラー入力の数。問題を報告する場合に使用されます。
cluster-infrastructure		Pacemaker が現在実行しているメッセージングスタック。情報提供および診断目的に使用されます。ユーザーは設定できません。
dc-version		クラスターの DC (Designated Controller) で Pacemaker のバージョン。診断目的に使用され、ユーザーは設定できません。
cluster-recheck-interval	15分	時間ベースでオプションを変更するポーリング間隔。使用できる値は、ポーリングを無効にする 0 (ゼロ) と、秒単位で間隔を示す正の値です (5min など、他の SI 単位が指定されていない場合に限ります)。
maintenance-mode	false	メンテナンスモードの場合は、クラスターが「干渉されない」モードになり、クラスターは指示されない限り、サービスを起動したり、停止したりしません。メンテナンスモードが完了すると、クラスターは、サービスの現在の状態のサニティーチェックを実行してから、これを必要とするサービスを停止するか、または起動します。
shutdown-escalation	20min	適切にシャットダウンして終了するのをやめる時間。高度な使用のみ。
stonith-timeout	60s	STONITH が完了するのを待つ時間。
stop-all-resources	false	クラスターがすべてのリソースの停止。
enable-acl	false	クラスターが、**pcs acl** コマンドで設定したアクセスコントロール一覧を使用できるかどうかを示します。

オプション	デフォルト	説明
placement-strategy	**default**	クラスターノードでリソースの配置を決定する際に、クラスターが利用率属性を考慮にいれるかどうかと、どのぐらい考慮するかを示します。

18.2. 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ーの値を設定するには、次の pcs コマンドを使用します。

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

たとえば、**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
```

または、**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ーを設定から削除できます。これにより、そのプロパティーの値がデフォルト値に戻されます。たとえば、以前に **symmetric-cluster** プロパティーを **false** に設定したことがある場合は、以下のコマンドにより、設定した値が設定から削除され、**symmetric-cluster** の値がデフォルト値の **true** に戻されます。

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

18.3. クラスタープロパティ設定のクエリー

ほとんどの場合は、各種のクラスターコンポーネントの値を表示するのに **pcs** コマンドを使用する場合に、**pcs list** または **pcs show** を交互に使用できます。次の例では、**pcs list** は、複数のプロパティーの設定を完全に表示するのに使用します。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します。

クラスターに設定されたプロパティー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
```

明示的には設定していないプロパティー設定のデフォルト値も含め、クラスターのプロパティー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ー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
```

たとえば、**cluster-infrastructure** プロパティー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ーの全デフォルト値の一覧を表示して、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できます。次のコマンドを使用します。

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

第19章 リソースとしての仮想ドメインの設定

pcs resource create コマンドを使用し、**VirtualDomain** をリソースタイプとして指定して、**libvirt** 仮想化フレームワークにより管理される仮想ドメインを設定できます。

仮想ドメインをリソースとして設定する場合は、以下を考慮してください。

- 仮想ドメインは、クラスターリソースとして設定する前に停止している必要があります。
- 仮想ドメインがクラスターリソースになる場合は、クラスターツールを使用しない限り、起動したり、停止したり、移行したりすることはできません。
- クラスターリソースとして設定した仮想ドメインを、ホストの起動時に起動するように設定することはできません。
- すべてのノードは、それぞれの管理される仮想ドメインの必要な設定ファイルおよびストレージデバイスにアクセスできる必要があります。

クラスターが仮想ドメイン自体のサービスを管理するようにする必要がある場合は、仮想ドメインをゲストノードとして設定できます。

19.1. 仮想ドメインリソースのオプション

表19.1「仮想ドメインリソースのリソースオプション」では、**VirtualDomain** リソースに設定できるリソースオプションを説明しています。

表19.1 仮想ドメインリソースのリソースオプション

フィールド	デフォルト	説明
config		(必須) この仮想ドメインの **libvirt** 設定ファイルへの絶対パス。
hypervisor	システムに依存	接続先のハイパーバイザーの URI。**virsh --quiet uri** コマンドを実行して、システムのデフォルト URI を確認できます。
force_stop	**0**	停止時にドメインを常に強制的にシャットダウン (「破棄」) します。デフォルト動作では、正常なシャットダウンの試行に失敗した後にのみ強制的なシャットダウンを実行します。仮想ドメイン (または仮想化バックエンド) が正常なシャットダウンをサポートしない場合にのみ、これを **true** に設定する必要があります。

フィールド	デフォルト	説明
migration_transport	システムに依存	移行中にリモートハイパーバイザーに接続するのに使用されるトランスポート。このパラメーターを省略すると、リソースは **libvirt** のデフォルトトランスポートを使用してリモートハイパーバイザーに接続します。
migration_network_suffix		専用の移行ネットワークを使用します。移行 URI は、このパラメーターの値をノード名の末尾に追加することにより構成されます。ノード名が完全修飾ドメイン名 (FQDN) の場合は、FQDN の最初のピリオド (.) の直前にサフィックスを挿入します。この構成されたホスト名がローカルに解決可能であり、関連付けられた IP アドレスが優先ネットワークを介して到達可能であることを確認してください。
monitor_scripts		仮想ドメイン内でサービスの監視を追加で実行するには、このパラメーターを監視するスクリプトの一覧と共に追加します。**注記**: 監視スクリプトが使用される場合、**start** および **migrate_from** 操作は、すべての監視スクリプトが正常に完了した場合にのみ完了します。これらの操作のタイムアウトを設定し、この遅延に対応できるようにします。
autoset_utilization_cpu	**true**	**true** に設定すると、監視の実行時に、エージェントは **virsh** から **domainU** の **vCPU** の数を検出し、これをリソースの CPU 利用率に組み込みます。
autoset_utilization_hv_memory	**true**	true に設定すると、監視の実行時に、エージェントは **virsh** から **Max memor** の数を検出し、これをリソースの **hv_memory** の利用率に組み込みます。
migrateport	ランダムハイポート	このポートは、**qemu** 移行 URI で使用されます。これを設定しないと、ポートにはランダムハイポートが使用されます。

フィールド	デフォルト	説明
snapshot		仮想マシンイメージが保存されるスナップショットディレクトリー。このパラメーターが設定されていると、仮想マシンの RAM 状態は、停止時にスナップショットディレクトリーのファイルに保存されます。起動時に、状態ファイルはドメインに表示し、ドメインは最後に停止された直前の状態と同じ状態に復元されます。このオプションには **force_stop** オプションとの互換性がありません。

VirtualDomain リソースオプション以外にも、**allow-migrate** メタデータオプションを、リソースの別のノードへのライブ移行を許可するように設定できます。このオプションが **true** に設定されている場合、リソースは状態を失うことなく移行されます。このオプションがデフォルトの状態である **false** に設定されている場合、仮想ドメインは、ノード間で移行される際に最初のノードでシャットダウンし、2 番目のノードで再起動します。

19.2. 仮想ドメインリソースの作成

以下の手順を使用して **VirtualDomain** リソースを作成します。

1. 仮想マシンを管理するために **VirtualDomain** リソースエージェントを作成する場合、Pacemaker では仮想マシンの **xml** 設定ファイルをディスクのファイルにダンプする必要があります。たとえば、**guest1** という名前の仮想マシンを作成し、ホストにあるファイルに **xml** ファイルをダンプします。任意に選択するファイル名を使用できますが、この例では **/etc/pacemaker/guest1.xml** を使用します。

   ```
   # virsh dumpxml guest1 > /etc/pacemaker/guest1.xml
   ```

2. ゲストノードが実行している場合はシャットダウンします。そのノードがクラスターで設定されている場合は Pacemaker により起動します。

3. **VirtualDoman** リソースを **pcs resource create** コマンドを使用して設定します。たとえば、以下のコマンドは **VM** という名前の **VirtualDomain** リソースを設定します。 **allow-migrate** オプションは **true** に設定されるため、**pcs move VM nodeX** コマンドはライブ移行として実行されます。

   ```
   # pcs resource create VM VirtualDomain config=.../vm.xml \
         migration_transport=ssh meta allow-migrate=true
   ```

第20章 クラスタークォーラム

Red Hat Enterprise Linux High Availability Add-On クラスターは、**votequorum** サービスとフェンシングを併用して、スプリットブレインが発生しないようにします。クラスターの各システムには投票数が割り当てられ、過半数の票がある場合のみクラスターの操作を続行できます。サービスは、すべてのノードにロードするか、すべてのノードにロードしないようにする必要があります。サービスがクラスターノードのサブセットにロードされると、結果が予想できなくなります。**votequorum** サービスの設定および操作の詳細は、man ページの **votequorum**(5) を参照してください。

20.1. クォーラムオプションの設定

pcs cluster setup コマンドを使用してクラスターを作成する場合は、クォーラム設定の特殊な機能を使用できます。これらのオプションは 表20.1「クォーラムオプション」 に記載されています。

表20.1 クォーラムオプション

オプション	説明
auto_tie_breaker	有効にすると、クラスターは決定論的に最大 50% のノードが同時に失敗しても存続されます。クラスターパーティションや、**auto_tie_breaker_node** に設定された **nodeid** (設定されていない場合は最小の **nodeid**) と通信するノードのセットは、クォーラムに達した状態を維持します。その他のノードはクォーラムに達しません。 **auto_tie_breaker** オプションを指定すると、クラスターは均等の分割では操作を継続できるため、偶数個のノードを持つクラスターで使用されます。複数で不均等の分割など、より複雑な障害は「クォーラムデバイス」を参照してください。 **auto_tie_breaker** オプションは、クォーラムデバイスと互換性がありません。
wait_for_all	有効にすると、最低 1 回、同時にすべてのノードが現れた後に、クラスターが初回だけクォーラムに達します。 **wait_for_all** オプションは、主にクォーラムデバイスの **lms** (last man standing) アルゴリズムを使用する、2 ノードまたは偶数のノードで構成されるクラスターで使用されます。 **wait_for_all** オプションは、クラスターに 2 つのノードがあり、クォーラムデバイスを使用せず、**auto_tie_breaker** が無効になっている場合に自動的に有効になります。**wait_for_all** を明示的に 0 に設定すると、このオプションをオーバーライドできます。
last_man_standing	有効にすると、クラスターは特定の状況で **expected_votes** とクォーラムを再計算します。このオプションを有効にするには、**wait_for_all** を有効にする必要があります。**last_man_standing** オプションには、クォーラムデバイスとの互換性がありません。

オプション	説明
last_man_standing_window	クラスターのノードが失われた後、**expected_votes** およびクォーラムを再計算するまでの待ち時間 (ミリ秒単位)。

これらのオプションの設定および使用に関する詳細は、man ページの **votequorum**(5) を参照してください。

20.2. クォーラムオプションの変更

pcs quorum update コマンドを使用してクラスターの一般的なクォーラムオプションを変更できます。このコマンドを実行するにはクラスターを停止する必要があります。クォーラムオプションの詳細は、man ページの **votequorum**(5) を参照してください。

pcs quorum update コマンドの形式は以下のとおりです。

```
pcs quorum update [auto_tie_breaker=[0|1]] [last_man_standing=[0|1]] [last_man_standing_window=
[time-in-ms] [wait_for_all=[0|1]]
```

以下の一連のコマンドは、**wait_for_all** クォーラムオプションを変更し、このオプションの更新された状態を表示します。クラスターの稼働中はこのコマンドを実行できないことに注意してください。

```
[root@node1:~]# pcs quorum update wait_for_all=1
Checking corosync is not running on nodes...
Error: node1: corosync is running
Error: node2: corosync is running

[root@node1:~]# pcs cluster stop --all
node2: Stopping Cluster (pacemaker)...
node1: Stopping Cluster (pacemaker)...
node1: Stopping Cluster (corosync)...
node2: Stopping Cluster (corosync)...

[root@node1:~]# pcs quorum update wait_for_all=1
Checking corosync is not running on nodes...
node2: corosync is not running
node1: corosync is not running
Sending updated corosync.conf to nodes...
node1: Succeeded
node2: Succeeded

[root@node1:~]# pcs quorum config
Options:
  wait_for_all: 1
```

20.3. クォーラム設定およびステータスの表示

クラスターの実行後、以下のクラスタークォーラムコマンドを実行できます。

以下のコマンドは、クォーラムの設定を表示します。

```
pcs quorum [config]
```

以下のコマンドは、クォーラムのランタイムステータスを表示します。

```
pcs quorum status
```

20.4. クォーラムに達しないクラスターの実行

長時間クラスターからノードを除去したためクォーラムの損失が発生した場合は、**pcs quorum expected-votes** コマンドを使用すると、ライブクラスターの **expected_votes** パラメーターの値を変更できます。これにより、クラスターは、クォーラムがなくても操作を継続できます。

警告

ライブクラスターで期待される票数 (vote) を変更する場合は、細心の注意を払って行ってください。期待される票数を手作業で変更したことにより、実行しているクラスターが 50% 未満となる場合は、クラスターの他のノードを別々に起動でき、クラスターサービスを開始できるため、データの破損や予期せぬ結果が発生することがあります。この値を変更する場合は、**wait_for_all** パラメーターが有効になっていることを確認してください。

以下のコマンドは、ライブクラスターの期待される票数を、指定の値に設定します。これはライブクラスターのみに影響し、設定ファイルは変更されません。リロードが行われると、**expected_votes** の値は、設定ファイルの値にリセットされます。

```
pcs quorum expected-votes votes
```

クォーラムに達していない状態でクラスターにリソース管理を続行させたい場合は、**pcs quorum unblock** コマンドを使用して、クォーラムの確立時にクラスターがすべてのノードを待機することのないようにします。

注記

このコマンドは細心の注意を払って使用する必要があります。このコマンドを実行する前に、現在クラスターにないノードを無効にし、共有リソースにアクセスできない状態であることを確認する必要があります。

```
# pcs quorum unblock
```

20.5. クォーラムデバイス

クラスター用のサードパーティーのアービトレーションデバイスとして機能する別のクォーラムデバイスを設定することにより、標準のクォーラムルールにより許容されるノード障害の数よりも多くのノード障害を保持できるようになりました。クォーラムデバイスは、偶数のノードで構成されるクラスターに推奨され、2 ノード構成のクラスターには強く推奨されます。

クォーラムデバイスの設定時には、以下を考慮する必要があります。

- クォーラムデバイスは、クォーラムデバイスを使用するクラスターと同じ場所にある異なる物理ネットワークで実行することが推奨されます。理想的には、クォーラムデバイスホストはメインクラスターとは別のラックに置くようにし、最低でも別の PSU に置くようにしてください。corosync リングと同じネットワークセグメントには置かないでください。

- 複数のクォーラムデバイスをクラスターで同時に使用することはできません。

- 複数のクォーラムデバイスをクラスターで同時に使用することはできませんが、複数のクラスターが 1 つのクォーラムデバイスを同時に使用することはできます。アルゴリズムやクォーラムオプションはクラスターノード自体に保存されるため、同じクォーラムデバイスを使用する各クラスターは、異なるアルゴリズムやクォーラムオプションを使用できます。たとえば、**ffsplit** ((fifty/fifty split) アルゴリズムを使用するクラスターと、**lms** (last man standing) アルゴリズムを使用する別のクラスターが、1 つのクォーラムデバイスを使用できます。

- クォーラムデバイスは、既存のクラスターノードで実行しないでください。

20.5.1. クォーラムデバイスパッケージのインストール

クラスターにクォーラムデバイスを設定するには、以下のパッケージをインストールする必要があります。

- 既存クラスターのノードで、**corosync-qdevice** をインストールします。

```
[root@node1:~]# yum install corosync-qdevice
[root@node2:~]# yum install corosync-qdevice
```

- クォーラムデバイスホストで、**pcs** および **corosync-qnetd** をインストールします。

```
[root@qdevice:~]# yum install pcs corosync-qnetd
```

- **pcsd** サービスを起動し、システムの起動時に **pcsd** がクォーラムデバイスホストで有効になるようにします。

```
[root@qdevice:~]# systemctl start pcsd.service
[root@qdevice:~]# systemctl enable pcsd.service
```

20.5.2. クォーラムデバイスの設定

以下の手順では、クォーラムデバイスを設定し、そのクォーラムデバイスをクラスターに追加します。この例では、以下が当てはまります。

- クォーラムデバイスに使用されるノードは **qdevice** です。

- クォーラムデバイスモデルは **net** で、これは現在サポートされている唯一のクォーラムデバイスモデルです。**net** モデルは、以下のアルゴリズムをサポートします。

  - **ffsplit** (fifty-fifty split) - この場合は、アクティブなノードの数が最も多いパーティションに 1 票が提供されます。

  - **lms** (last-man-standing) - ノードが **qnetd** サーバーを確認できるクラスター内に残っている唯一のノードである場合に、票が返されます。

警告

LMS アルゴリズムは、残りのノードが 1 つしかなくても、クラスターがクォーラムに達した状態であることを許可しますが、これは number_of_nodes - 1 と同じであるため、クォーラムデバイスの議決権が強いことも意味します。クォーラムデバイスとの接続を失うことは、number_of_nodes - 1 票を失うことを意味し、これはすべてのノードがアクティブであるクラスターのみが (クォーラムデバイスへの投票過多により) クォーラムに達した状態にあることを意味し、その他のクラスターはクォーラムに達しない状態になることを意味します。

これらのアルゴリズムの実装の詳細は、man ページの **corosync-qdevice**(8) を参照してください。

- クラスターノードは **node1** と **node2** です。

以下の手順は、クォーラムデバイスを設定し、そのクォーラムデバイスをクラスターに追加します。

1. クォーラムデバイスをホストするために使用するノードで以下のコマンドを使用し、クォーラムデバイスを設定します。このコマンドは、クォーラムデバイスモデルである **net** を設定および開始し、デバイスが起動時に開始されるよう設定します。

```
[root@qdevice:~]# pcs qdevice setup model net --enable --start
Quorum device 'net' initialized
quorum device enabled
Starting quorum device...
quorum device started
```

クォーラムデバイスの設定後、そのステータスをチェックできます。**corosync-qnetd** デーモンが実行していることを確認でき、この時点では接続されているクライアントがないはずです。**--full** コマンドオプションを指定すると詳細が出力されます。

```
[root@qdevice:~]# pcs qdevice status net --full
QNetd address:                  *:5403
TLS:                            Supported (client certificate required)
Connected clients:              0
Connected clusters:             0
Maximum send/receive size:      32768/32768 bytes
```

2. 以下のコマンドを実行して、**firewalld** で **high-availability** サービスを有効にし、 **pcsd** デーモンで必要なファイアウォールのポートおよび **net** クォーラムデバイスを有効にします。

```
[root@qdevice:~]# firewall-cmd --permanent --add-service=high-availability
[root@qdevice:~]# firewall-cmd --add-service=high-availability
```

3. 既存クラスターのいずれかのノードにより、クォーラムデバイスをホストしているノードでユーザー **hacluster** を認証します。これにより、クラスターの **pcs** が **qdevice** ホストの **pcs** にアクセスできるようになりますが、**qdevice** ホストの **pcs** は、クラスターの **pcs** に接続することを許可しません。

```
[root@node1:~] # pcs host auth qdevice
Username: hacluster
Password:
qdevice: Authorized
```

4. クォーラムデバイスをクラスターに追加します。
クォーラムデバイスを追加する前に、クォーラムデバイスの現在の設定と状況をチェックして後で比較できます。このコマンドの出力は、クラスターがクォーラムデバイスを使用していないことを示しています。

```
[root@node1:~]# pcs quorum config
Options:
```

```
[root@node1:~]# pcs quorum status
Quorum information
------------------
Date:             Wed Jun 29 13:15:36 2016
Quorum provider:  corosync_votequorum
Nodes:            2
Node ID:          1
Ring ID:          1/8272
Quorate:          Yes

Votequorum information
----------------------
Expected votes:   2
Highest expected: 2
Total votes:      2
Quorum:           1
Flags:            2Node Quorate

Membership information
----------------------
    Nodeid      Votes    Qdevice Name
         1          1         NR node1 (local)
         2          1         NR node2
```

以下のコマンドは、以前に作成したクォーラムデバイスをクラスターに追加します。複数のクォーラムデバイスをクラスターで同時に使用することはできませんが、複数のクラスターが 1 つのクォーラムデバイスを同時に使用することはできます。このコマンド例は、クォーラムデバイスが **ffsplit** アルゴリズムを使用するよう設定します。クォーラムデバイスの設定オプションに関する情報は、man ページの **corosync-qdevice**(8) を参照してください。

```
[root@node1:~]# pcs quorum device add model net host=qdevice algorithm=ffsplit
Setting up qdevice certificates on nodes...
node2: Succeeded
node1: Succeeded
Enabling corosync-qdevice...
node1: corosync-qdevice enabled
node2: corosync-qdevice enabled
Sending updated corosync.conf to nodes...
node1: Succeeded
node2: Succeeded
Corosync configuration reloaded
```

```
Starting corosync-qdevice...
node1: corosync-qdevice started
node2: corosync-qdevice started
```

5. クォーラムデバイスの設定状態をチェックします。
クラスター側から以下のコマンドを実行すると、設定の変更内容を確認できます。

pcs quorum config は、設定されたクォーラムデバイスを表示します。

```
[root@node1:~]# pcs quorum config
Options:
Device:
  Model: net
    algorithm: ffsplit
    host: qdevice
```

pcs quorum status コマンドは、クォーラムデバイスが使用中であることを示すクォーラムデバイスのステータスを示します。

```
[root@node1:~]# pcs quorum status
Quorum information
------------------
Date:             Wed Jun 29 13:17:02 2016
Quorum provider:  corosync_votequorum
Nodes:            2
Node ID:          1
Ring ID:          1/8272
Quorate:          Yes

Votequorum information
----------------------
Expected votes:   3
Highest expected: 3
Total votes:      3
Quorum:           2
Flags:            Quorate Qdevice

Membership information
----------------------
    Nodeid      Votes    Qdevice Name
         1          1    A,V,NMW node1 (local)
         2          1    A,V,NMW node2
         0          1            Qdevice
```

pcs quorum device status は、クォーラムデバイスのランタイムステータスを表示します。

```
[root@node1:~]# pcs quorum device status
Qdevice information
-------------------
Model:                  Net
Node ID:                1
Configured node list:
    0   Node ID = 1
    1   Node ID = 2
Membership node list:   1, 2
```

```
Qdevice-net information
----------------------
Cluster name:          mycluster
QNetd host:            qdevice:5403
Algorithm:             ffsplit
Tie-breaker:           Node with lowest node ID
State:                 Connected
```

クォーラムデバイスから以下のコマンドを実行すると、**corosync-qnetd** デーモンのステータスを表示できます。

```
[root@qdevice:~]# pcs qdevice status net --full
QNetd address:                  *:5403
TLS:                            Supported (client certificate required)
Connected clients:              2
Connected clusters:             1
Maximum send/receive size:      32768/32768 bytes
Cluster "mycluster":
    Algorithm:          ffsplit
    Tie-breaker:        Node with lowest node ID
    Node ID 2:
        Client address:         ::ffff:192.168.122.122:50028
        HB interval:            8000ms
        Configured node list:   1, 2
        Ring ID:                1.2050
        Membership node list:   1, 2
        TLS active:             Yes (client certificate verified)
        Vote:                   ACK (ACK)
    Node ID 1:
        Client address:         ::ffff:192.168.122.121:48786
        HB interval:            8000ms
        Configured node list:   1, 2
        Ring ID:                1.2050
        Membership node list:   1, 2
        TLS active:             Yes (client certificate verified)
        Vote:                   ACK (ACK)
```

20.5.3. クォーラムデバイスサービスの管理

以下のコマンド例が示すように、PCS はローカルホスト (**corosync-qnetd**) でクォーラムデバイスサービスを管理する機能を提供します。このコマンドは、**corosync-qnetd** サービスにのみ影響することに注意してください。

```
[root@qdevice:~]# pcs qdevice start net
[root@qdevice:~]# pcs qdevice stop net
[root@qdevice:~]# pcs qdevice enable net
[root@qdevice:~]# pcs qdevice disable net
[root@qdevice:~]# pcs qdevice kill net
```

20.5.4. クラスターでのクォーラムデバイス設定の管理

以下のセクションでは、クラスター内のクォーラムデバイスの設定を管理するのに使用できる PCS コマンドを説明します。

20.5.4.1. クォーラムデバイス設定の変更

クォーラムデバイスの設定を変更するには、**pcs quorum device update** コマンドを使用します。

警告

クォーラムデバイスモデル **net** の **host** オプションを変更するには、**pcs quorum device remove** および **pcs quorum device add** コマンドを使用し、設定を適切に行います (変更前のホストと変更後のホストが同じマシンである場合を除く)。

以下のコマンドは、クォーラムデバイスアルゴリズムを **lms** に変更します。

```
[root@node1:~]# pcs quorum device update model algorithm=lms
Sending updated corosync.conf to nodes...
node1: Succeeded
node2: Succeeded
Corosync configuration reloaded
Reloading qdevice configuration on nodes...
node1: corosync-qdevice stopped
node2: corosync-qdevice stopped
node1: corosync-qdevice started
node2: corosync-qdevice started
```

20.5.4.2. クォーラムデバイスの削除

以下のコマンドを使用して、クラスターノードに設定されたクォーラムデバイスを削除します。

```
[root@node1:~]# pcs quorum device remove
Sending updated corosync.conf to nodes...
node1: Succeeded
node2: Succeeded
Corosync configuration reloaded
Disabling corosync-qdevice...
node1: corosync-qdevice disabled
node2: corosync-qdevice disabled
Stopping corosync-qdevice...
node1: corosync-qdevice stopped
node2: corosync-qdevice stopped
Removing qdevice certificates from nodes...
node1: Succeeded
node2: Succeeded
```

クォーラムデバイスを削除した後、クォーラムデバイスの状態を確認すると、以下のエラーメッセージが表示されます。

```
[root@node1:~]# pcs quorum device status
Error: Unable to get quorum status: corosync-qdevice-tool: Can't connect to QDevice socket (is
QDevice running?): No such file or directory
```

20.5.4.3. クォーラムデバイスの破棄

クォーラムデバイスホストのクォーラムデバイスを無効化および停止し、設定ファイルをすべて削除するには、以下のコマンドを実行します。

```
[root@qdevice:~]# pcs qdevice destroy net
Stopping quorum device...
quorum device stopped
quorum device disabled
Quorum device 'net' configuration files removed
```

第21章 COROSYNC 以外のノードのクラスターへの統合: PACEMAKER_REMOTE サービス

pacemaker_remote サービスを使用すると、**corosync** を実行していないノードをクラスターに統合し、そのノードのリソースは、クラスターにより実際のクラスターノードと同様に管理されます。

pacemaker_remote サービスが提供する機能には以下が含まれます。

- **pacemaker_remote** サービスは、**corosync** の 16 ノード制限を超えた拡張を可能にします。
- **pacemaker_remote** サービスを使用すると、仮想環境をクラスターリソースとして管理でき、さらに仮想環境内の個別のサービスをクラスターリソースとして管理できます。

pacemaker_remote サービスは、以下の用語を使用して記述されます。

- cluster node - 高可用性サービスを実行しているノード (**pacemaker** および **corosync**)。
- remote node - **corosync** クラスターメンバーシップを必要としないクラスターにリモートで統合するために **pacemaker_remote** を実行しているノード。リモートノードは、**ocf:pacemaker:remote** リソースエージェントを使用するクラスターリソースとして設定されます。
- guest node - **pacemaker_remote** サービスを実行する仮想ゲストノード。仮想ゲストリソースはクラスターにより管理されます。これはクラスターにより起動し、リモートノードとしてクラスターに統合されます。
- pacemaker_remote - Pacemaker クラスター環境のリモートノードおよびゲストノード内でリモートアプリケーション管理を実行できるサービスデーモン。このサービスは、corosync を実行していないノードでリソースをリモートで管理できる Pacemaker のローカル実行プログラムデーモン (**pacemaker-execd**) の拡張バージョンです。

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。

- リモートノードおよびゲストノードは **pacemaker_remote** サービスを実行します (仮想マシン側で必要な設定はほとんどありません)。
- 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** および **corosync**) はリモートノードで **pacemaker_remote** サービスに接続するため、クラスターに統合できます。
- 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** および **corosync**) はゲストノードを開始し、ゲストノードで **pacemaker_remote** サービスに即座に接続するため、クラスターに統合できます。

クラスターノードと、クラスターノードが管理するリモートおよびゲストノードの主な違いは、リモートおよびゲストノードはクラスタースタックを実行しないことです。そのため、リモートおよびゲストノードには以下の制限があります。

- クォーラムでは実行されない
- フェンスデバイスの動作を実行しない
- クラスターの指定コントローラー (DC) として機能できない
- **pcs** コマンドは一部しか実行できない

その一方で、リモートノードおよびゲストノードは、クラススタックに関連するスケーラビリティーの制限に拘束されません。

前述の制限以外では、リモートノードはリソース管理に関してクラスターノードと同様に動作し、リモートおよびゲストノード自体をフェンシングできます。クラスターは、各リモートノードおよびゲストノードのリソースを完全に管理し、監視できます。これらのノードに制約を作成したり、これらのノードをスタンバイ状態にできます。または、**pcs** コマンドを使用して、クラスターノードでその他の動作を実行することもできます。リモートおよびゲストノードは、クラスターノードと同様にクラスターステータスの出力に表示されます。

21.1. PACEMAKER_REMOTE ノードのホストおよびゲスト認証

クラスターノードと pacemaker_remote 間の接続には、TLS (Transport Layer Security) が使用され、PSK (Pre-Shared Key) の暗号化と TCP 上の認証 (デフォルトでポート 3121 を使用) でセキュア化されます。そのため、**pacemaker_remote** を実行しているクラスターノードとノードは、同じプライベートキーを共有する必要があります。デフォルトでは、クラスターノードとリモートノードの両方でこのキーを **/etc/pacemaker/authkey** に配置する必要があります。

pcs cluster node add-guest コマンドは、ゲストノードに **authkey** をセットアップし、**pcs cluster node add-remote** コマンドは、リモートノードに **authkey** をセットアップします。

21.2. KVM ゲストノードの設定

Pacemaker ゲストノードは、**pacemaker_remote** サービスを実行する仮想ゲストノードです。仮想ゲストノードはクラスターにより管理されます。

21.2.1. ゲストノードリソースのオプション

ゲストノードとして動作するように仮想マシンを設定する場合は、仮想マシンを管理する **VirtualDomain** リソースを作成します。**VirtualDomain** リソースに設定できるオプションの説明は、表 19.1「仮想ドメインリソースのリソースオプション」 を参照してください。

VirtualDomain リソースオプションのほかにも、メタデータオプションはリソースをゲストノードとして定義し、接続パラメーターを定義します。 **pcs cluster node add-guest** コマンドを使用して、これらのリソースオプションを設定します。表21.1「KVM リソースをリモートノードとして設定するためのメタデータオプション」 では、これらのメタデータオプションを説明しています。

表21.1 KVM リソースをリモートノードとして設定するための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>	このリソースが定義するゲストノードの名前。リソースをゲストノードとして有効にし、ゲストノードの識別に使用される一意名を定義します。 **警告**: この値はリソースやノードの ID と重複してはなりません。
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します。

フィールド	デフォルト	説明
remote-addr	**pcs host auth** コマンドで指定されるアドレス	接続先の 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

21.2.2. 仮想マシンのゲストノードとしての統合

以下のセクションでは、**libvirt** と KVM 仮想ゲストを使用して、Pacemaker で仮想マシンを起動し、そのマシンをゲストノードとして統合するステップの概要を説明します。

1. **VirtualDomain** リソースを設定します。
2. すべての仮想マシンで以下のコマンドを実行し、 **pacemaker_remote** パッケージをインストールし、**pcsd** サービスを起動し、これを起動時に実行できるようにし、ファイアウォールを介して TCP のポート 3121 を許可します。

```
# yum install pacemaker-remote resource-agents pcs
# systemctl start pcsd.service
# systemctl enable pcsd.service
# firewall-cmd --add-port 3121/tcp --permanent
# firewall-cmd --add-port 2224/tcp --permanent
# firewall-cmd --reload
```

3. 各仮想マシンに、すべてのノードが認識できる静的ネットワークアドレスと一意なホスト名を割り当てます。ゲスト仮想マシンに静的 IP アドレスを設定する方法は**『仮想化の導入および管理ガイド』**を参照してください。
4. ゲストノードとして統合しようとしているノードに対して **pcs** を実行していない場合は実行します。

```
# pcs host auth nodename
```

5. 以下のコマンドを使用して、既存の **VirtualDomain** リソースをゲストノードに変換します。このコマンドはクラスターノードで実行され、追加されるゲストノードで実行してはなりません。リソースを変換する以外にも、このコマンドは **/etc/pacemaker/authkey** をゲストノードにコピーし、ゲストノードで **pacemaker_remote** デーモンを起動し、これを有効にします。任意に定義できるゲストノードのノード名は、ノードのホスト名とは異なる可能性があります。

```
# pcs cluster node add-guest nodename resource_id [options]
```

6. **VirtualDomain** リソースの作成後は、ゲストノードを、クラスターの他のノードと同じように扱うことができます。たとえば、以下のコマンドをクラスターノードから実行すると、リソースを作成後にリソース制約をこのリソースに配置し、ゲストノードで実行するようになります。ゲストノードはグループで含めることができ、これによりストレージデバイス、ファイルシステム、および VM をグループ化できます。

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

21.3. PACEMAKER リモートノードの設定

リモートノードは、**ocf:pacemaker:remote** がリソースエージェントとして指定された状態で、クラスターリソースとして定義されます。**pcs cluster node add-remote** コマンドを使用してこのリソースを作成します。

21.3.1. リモートノードリソースのオプション

表21.2「リモートノードのリソースオプション」 では、 **remote** リソースに設定できるリソースオプションを説明しています。

表21.2 リモートノードのリソースオプション

フィールド	デフォルト	説明
reconnect_interval	0	リモートノードへのアクティブな接続が切断された後、リモートノードへの再接続を試みる前に待機する時間 (秒単位)。待機期間は繰り返されます。待機期間の後に再接続に失敗した場合は、さらなる待機期間の後に再接続を試みます。このオプションが使用されると、Pacemaker は待機期間の後に無限にリモートノードへ接続を試みます。
server	**pcs host auth** コマンドで指定されるアドレス	接続するサーバーの場所。IP アドレスまたはホスト名を指定できます。
port		接続する TCP ポート。

21.3.2. 設定の概要: リモートノード

本セクションでは、Pacemaker リモートノードを設定し、そのノードを既存の Pacemaker クラスター環境に統合する手順の概要を説明します。

1. リモートノードを設定するノードで、ローカルファイアウォールを介してクラスター関連のサービスを許可します。

```
# firewall-cmd --permanent --add-service=high-availability
success
# firewall-cmd --reload
success
```

注記

iptables を直接使用する場合や、 **firewalld** 以外のファイアウォールソリューションを使用する場合は、単に TCP ポート 2224 および 3121 を開きます。

2. リモートノードで、**pacemaker_remote** デーモンをインストールします。

```
# yum install -y pacemaker-remote resource-agents pcs
```

3. リモートノードで **pcsd** を開始し、有効にします。

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

4. まだ実行していない場合は、ゲストノードとして追加しようとしているノードに **pcs** を実行します。

```
# pcs host auth remote1
```

5. 以下のコマンドを使用して、リモートノードリソースをクラスターに追加します。このコマンドは、関連するすべての設定ファイルを新規ノードに追加し、ノードを起動し、これを起動時に **pacemaker_remote** を開始するように設定することもできます。このコマンドは、クラスターノードで実行する必要があり、追加するリモートで実行することはできません。

```
# pcs cluster node add-remote remote1
```

6. **remote** リソースのクラスーへの追加後に、リモートノードをクラスターの他のノードと同じように扱うことができます。たとえば、以下のコマンドをクラスターノードから実行すると、リソースを作成し、そのリソースにリソース制約を配置して、リモートノードで実行できます。

```
# pcs resource create webserver apache configfile=/etc/httpd/conf/httpd.conf op
monitor interval=30s
# pcs constraint location webserver prefers remote1
```

警告

リソースグループ、コロケーション制約、または順序制約でノード接続リソースを利用しないでください。

7. リモートノードのフェンスリソースを設定します。リモートノードは、クラスターノードと同様にフェンシングされます。クラスターノードと同様に、リモートノードと使用するフェンスリソースを設定します。リモートノードはフェンスアクションを開始してはならないことに注意してください。他のノードに対してフェンシング操作を実行できるのはクラスターノードのみです。

21.4. ポートのデフォルトの場所の変更

Pacemaker または **pacemaker_remote** のいずれかのポートのデフォルトの場所を変更する必要がある場合は、これらのデーモンのどちらにも影響を与える **PCMK_remote_port** を設定できます。この環境変数は、以下のように **/etc/sysconfig/pacemaker** に配置して有効にできます。

```
====# Pacemaker Remote
...
#
```

```
# Specify a custom port for Pacemaker Remote connections
PCMK_remote_port=3121
```

特定のゲストノードまたはリモートノードにより使用されるデフォルトのポートを変更する場合は、**PCMK_remote_port** 変数を、そのノードの **/etc/sysconfig/pacemaker** ファイルに設定する必要があります。また、ゲストノードまたはリモートノードの接続を作成するクラスターリソースを、同じポート番号で設定する必要もあります (ゲストノードの場合は **remote-port** メタデータオプション、リモートノードの場合は **port** オプションを使用します)。

21.5. PACEMAKER_REMOTE ノードを含むシステムのアップグレード

アクティブな Pacemaker リモートノードで **pacemaker_remote** サービスが停止すると、クラスターは、ノードの停止前に、リソースをノードから正常に移行します。これにより、クラスターからノードを削除せずに、ソフトウェアのアップグレードやその他の定期的なメンテナンスを実行できます。**pacemaker_remote** がシャットダウンすると、クラスターは即時に再接続を試行します。リソースのモニタータイムアウト内に **pacemaker_remote** が再起動しないと、クラスターは監視操作が失敗したと判断します。

アクティブな Pacemaker リモートノードで、**pacemaker_remote** サービスが停止したときに監視が失敗しないようにするには、以下の手順に従って、**pacemaker_remote** を停止する可能性があるシステム管理を実行する前に、ノードをクラスターから削除します。

1. ノードからすべてのサービスを移行する **pcs resource disable resourcename** を使用して、ノードの接続リソースを停止します。ゲストノードの場合は、VM も停止するため、(**virsh** などを使用して) VM をクラスターの外部で起動し、メンテナンスを実行する必要があります。

2. 必要なメンテナンスを実行します。

3. ノードをクラスターに戻す準備ができたら、**pcs resource enable** でリソースを再度有効にします。

第22章 クラスターメンテナンスの実行

クラスターのノードでメンテナンスを実行するには、そのクラスターで実行しているリソースおよびサービスを停止するか、または移行する必要がある場合があります。または、サービスを変更しない状態で、クラスターソフトウェアの停止が必要になる場合があります。Pacemaker は、システムメンテナンスを実行するための方法を各種提供します。

- クラスターの別のノードでサービスが継続的に実行している状態で、クラスター内のあるノードを停止する必要がある場合は、そのクラスターノードをスタンバイモードにできます。スタンバイノードのノードは、リソースをホストすることができなくなります。ノードで現在アクティブなリソースは、他のノードに移行するか、または他のノードがそのリソースを実行できない場合は停止します。スタンドバイモードの詳細は「ノードをスタンバイモードにする」を参照してください。

- リソースを停止せずに、実行しているノードから個別のリソースを移行する必要がある場合は、**pcs resource move** コマンドを使用してリソースを別のノードに移行できます。**pcs resource move** コマンドの詳細は「クラスターリソースの手動による移行」を参照してください。
  pcs resource move コマンドを実行すると、現在実行しているノードでそれが実行されないように、制約がリソースに追加されます。リソースを戻す準備ができたら、**pcs resource clear** または **pcs constraint delete** コマンドを実行すると、制約を削除できます。このコマンドを実行しても、リソースが必ずしも元のノードに戻る訳ではありません。この時点でリソースが実行できる場所は、リソースの最初の設定方法によって異なるためです。「リソースの優先ノードへの移動」の説明どおりに、リソースは **pcs resource relocate run** コマンドを実行して、優先ノードに移動できます。

- 実行中のリソースを完全に停止し、クラスターが再び起動しないようにする必要がある場合は、**pcs resource disable** コマンドを使用できます。**pcs resource disable** コマンドの詳細は「クラスターリソースの有効化、無効化、および禁止」を参照してください。

- Pacemaker がリソースに対するアクションを実行するのを防ぐ必要がある場合 (たとえば、リソースに対するメンテナンスの実行中に復元アクションを無効にする必要がある場合や、**/etc/sysconfig/pacemaker** 設定をリロードする必要がある場合)、「リソースのアンマネージドモードへの設定」で説明されているように **pcs resource unmanage** コマンドを使用します。Pacemaker Remote 接続リソースは、アンマネージドモードにすることはできません。

- クラスターを、サービスが開始したり停止したりしない状態にする必要がある場合は、**maintenance-mode** クラスタープロパティーを設定できます。クラスターをメンテナンスモードにすると、すべてのリソースが自動的にアンマネージドモードになります。メンテナンスモードのクラスターの詳細は「クラスターをメンテナンスモードにする」を参照してください。

- RHEL High Availability および Resilient Storage Add-On を更新する必要がある場合は、「Red Hat Enterprise Linux High Availability クラスターの更新」で説明されているように、一度に 1 つのパッケージを更新するか、または全体のクラスターに対して更新を行うことができます。

- Pacemaker リモートノードでメンテナンスを実行する必要がある場合は、「リモートノードおよびゲストノードのアップグレード」で説明されているように、リモートノードリソースを無効にすることで、ノードをクラスターから削除できます。

22.1. ノードをスタンバイモードに

クラスターノードがスタンバイモードの場合、ノードはリソースをホストできなくなります。ノードで現在アクティブなすべてのリソースは、別のノードに移行します。

以下のコマンドは、指定ノードをスタンバイモードにします。**--all** を指定する場合、このコマンドはすべてのノードをスタンバイモードにします。

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できます。また、設定をテストして、実際にはノードをシャットダウンせずにリカバリーのシュミレーションを行う場合にも、このコマンドを使用できます。

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

以下のコマンドは、指定したノードをスタンバイモードから外します。この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** を指定すると、このコマンドはすべてのノードをスタンバイモードから外します。

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

pcs resource ban コマンドを実行すると、場所の制約である -INFINITY がリソースに追加され、リソースが指定のノードで実行されないようにします。**pcs resource clear** コマンドまたは **pcs constraint delete** コマンドを実行すると制約を削除できます。このコマンドを実行しても、リソースが必ずしも指定のノードに戻る訳ではありません。この時点でリソースが実行できる場所は、リソースの最初の設定によって異なります。

22.2. クラスターリソースの手動による移行

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 つ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
- 個別に指定したリソースを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。

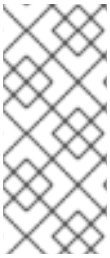
個別に指定したリソースは、以下の方法のいずれかで移動できます。

- **pcs resource move** コマンドを使用して、現在実行しているノードからリソースを移行できます。
- **pcs resource relocate run** コマンドを使用して、現在のクラスターのステータス、制約、リソースの場所、およびその他の設定により決定される優先ノードへ、リソースを移動します。

22.2.1. 現在のノードからのリソースの移動

現在実行しているノードからリソースを移動するには、以下のコマンドを使用し、リソースの **resource_id** を定義どおりに指定します。移動するリソースを実行する移動先のノードを指定する場合は、**destination_node** を使用します。

```
pcs resource move resource_id [destination_node] [--master] [lifetime=lifetime]
```



注記

pcs resource move コマンドを実行すると、現在実行しているノードで実行されないように制約がリソースに追加されます。**pcs resource clear** または **pcs constraint delete** コマンドを実行すると、制約を削除できます。このコマンドを実行しても、リソースが必ずしも元のノードに戻る訳ではありません。この時点でリソースが実行できる場所は、リソースの最初の設定方法によって異なります。

pcs resource move コマンドの **--master** パラメーターを指定すると、制約の範囲がマスターロールに限定され、**resource_id** の代わりに **master_id** を指定する必要があります。

任意で **pcs resource move** コマンドの **lifetime** パラメーターを設定すると、制限が維持される期間を指定できます。ISO 8601 に定義された形式に従って、**lifetime** パラメーターの単位を指定できます。ISO 8601 では、Y (年)、M (月)、W (週)、D (日)、H (時)、M (分)、S (秒) のように、単位を大文字で指定する必要があります。

分単位の M と、月単位の M を区別するには、分単位の値の前に PT を指定する必要があります。たとえば、5M の **lifetime** パラメーターは 5 カ月の間隔を示し、PT5M の **lifetime** パラメーターは 5 分の間隔を示します。

lifetime パラメーターは、**cluster-recheck-interval** クラスタープロパティーに定義した間隔でチェックされます。デフォルト値は 15 分です。このパラメーターを頻繁にチェックする必要がある場合は、以下のコマンドを実行して、この値をリセットできます。

```
pcs property set cluster-recheck-interval=value
```

任意で、**pcs resource move** コマンドに **--wait[=*n*]** パラメーターを設定し、移動先のノードでリソースが起動するまでの待機時間 (秒単位) を指定できます。待機時間がこの値を超えると、リソースが起動した場合は 0 が返され、リソースが起動しなかった場合は 1 が返されます。n の値を指定しないと、デフォルトのリソースタイムアウト値が使用されます。

以下のコマンドは、リソース **resource1** をノード **example-node2** へ移動し、1 時間 30 分以内に元のノードへ戻らないようにします。

```
pcs resource move resource1 example-node2 lifetime=PT1H30M
```

以下のコマンドは、リソース **resource1** をノード **example-node2** へ移動し、30 分以内に元のノードへ戻らないようにします。

```
pcs resource move resource1 example-node2 lifetime=PT30M
```

22.2.2. リソースの優先ノードへの移動

フェイルオーバーや管理者の手作業によるノードの移動により、リソースが移動された後、フェイルオーバーの原因となった状況が改善されてもそのリソースは必ずしも元のノードに戻るわけではありません。リソースを優先ノードへ移動するには、以下のコマンドを実行します。優先ノードは、現在のクラスター状態、制約、リソースの場所、およびその他の設定により決定され、時間とともに変更する可能性があります。

```
pcs resource relocate run [resource1] [resource2] ...
```

リソースを指定しないと、すべてのリソースが優先ノードに移動されます。

このコマンドは、resource stickiness を無視し、各リソースの優先ノードを算出します。優先ノードの

算出後、リソースを優先ノードに移動させる場所の制約を作成します。リソースが移動した後、制約は自動的に削除されます。**pcs resource relocate run** コマンドにより作成された制約をすべて削除するには、**pcs resource relocate clear** コマンドを実行します。リソースの現在の状態と、resource stickiness を無視した最適なノードを表示するには、**pcs resource relocate show** コマンドを実行します。

22.3. クラスターリソースの有効化、無効化、および禁止

pcs resource move コマンドや **pcs resource relocate** コマンドのほかにも、クラスターリソースの動作を制御するために使用できるコマンドは複数あります。

実行中のリソースを手作業で停止し、クラスターが再起動しないようにするには、以下のコマンドを使用します。その他の設定 (制約、オプション、失敗など) によっては、リソースが起動した状態のままになる可能性があります。**--wait** オプションを指定すると、pcs はリソースが停止するまで「n」秒間待機し、リソースが停止した場合は 0 を返し、リソースが停止しなかった場合は 1 を返します。「n」が指定されない場合、デフォルトは 60 分に設定されます。

```
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、以下のコマンドを使用します。他の設定によっては、リソースが停止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 はリソースが停止するまで最長で「n」秒間待機し、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。「n」が指定されない場合、デフォルトは 60 分に設定されます。

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

指定したノードでリソースが実行し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への設定になります。

```
pcs resource ban resource_id [node] [--master] [lifetime=lifetime] [--wait[=n]]
```

pcs resource ban コマンドを実行すると、場所の制約である -INFINITY がリソースに追加され、リソースが指定のノードで実行されないようにします。**pcs resource clear** または **pcs constraint delete** コマンドを実行すると制約を削除できます。このコマンドを実行しても、リソースが必ずしも指定のノードに戻る訳ではありません。この時点でリソースが実行できる場所は、リソースの最初の設定によって異なります。

pcs resource ban コマンドの **--master** パラメーターを指定すると、制約の範囲がマスターロールに限定され、resource_id の代わりに master_id を指定する必要があります。

オプションで **pcs resource ban** コマンドに **lifetime** パラメーターを設定し、制約が持続する期間を指定できます。

オプションで **pcs resource ban** コマンドに **--wait[=n]** パラメーターを設定し、移動先のノードでリソースが起動するまでの待機時間 (秒単位) を指定できます。待機時間がこの値を超えると、リソースが起動した場合に 0 が返され、リソースが起動しなかった場合は 1 が返されます。n の値を指定しないと、デフォルトのリソースタイムアウト値が使用されます。

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。この場合、クラスターの推奨は無視され、起動しているリソースからの出力が表示されます。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** コマンドを使用し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か (システムログでデバッグ

します)、リソースが起動しないよう制約が設定されているか、またはリソースが無効になっているかのいずれかが原因になります。この場合は、このコマンドを使用してリソースの設定をテストができます。ただし、これはクラスター内でのリソースの通常の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
```

22.4. リソースのアンマネージドモードへの設定

リソースが **unmanaged** モードの場合、リソースは引き続き設定で維持されますが Pacemaker はこのリソースを管理しません。

以下のコマンドは、指定のリソースを **unmanaged** モードに設定します。

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

以下のコマンドは、リソースをデフォルトの **managed** モードに設定します。

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

pcs resource manage コマンドまたは **pcs resource unmanage** コマンドを使用してリソースグループの名前を指定できます。コマンドは、グループのすべてのリソースに対して実行するため、1 つのコマンドでグループのリソースをすべて **managed** または **unmanaged** モードに設定し、組み込まれているリソースを個別に管理できます。

22.5. クラスターをメンテナンスモードに

クラスターがメンテナンスモードの場合、クラスターは指示されない限り、サービスを起動したり、停止したりしません。メンテナンスモードが完了すると、クラスターは、サービスの現在の状態のサニティーチェックを実行してから、これを必要とするサービスを停止するか、または起動します。

クラスターをメンテナンスモードにするには、以下のコマンドを使用して、**maintenance-mode** クラスタープロパティーを **true** に設定します。

```
# pcs property set maintenance-mode=true
```

クラスターをメンテナンスモードから外すには、以下のコマンドを使用して、**maintenance-mode** クラスタープロパティーを **false** に設定します。

```
# pcs property set maintenance-mode=false
```

設定からクラスタープロパティーを削除するには、以下のコマンドを実行します。

```
pcs property unset property
```

または、**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ーを設定から削除できます。これにより、そのプロパティーの値がデフォルト値に戻されます。たとえば、以前に **symmetric-cluster** プロパティーを **false** に設定したことがある場合は、以下のコマンドにより、設定した値が設定から削除され、**symmetric-cluster** の値がデフォルト値の **true** に戻されます。

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

22.6. RED HAT ENTERPRISE LINUX HIGH AVAILABILITY クラスターの更新

RHEL High Availability および Resilient Storage Add-On を構成するパッケージを、個別または全体として更新するには、以下に示す一般的な方法のいずれかを使用できます。

- **ローリングアップデート** - サービスからノードを、一度に 1 つずつ削除し、そのソフトウェアを更新してから、これをクラスターに戻します。これにより、各ノードが更新されている間に、クラスターがサービスの提供とリソースの管理を継続できます。
- **クラスター全体の更新** - クラスター全体を停止し、更新をすべてのノードに適用してから、クラスターのバックアップを開始します。

警告

Red Hat Enterprise LInux High Availability および Resilient Storage クラスターのソフトウェア更新手順を実行する際に、その更新が開始する前に、更新を行うノードがクラスターのアクティブなメンバーではないことを確認する必要があります。

これらの各方法の詳細な説明および更新手順は「Recommended Practices for Applying Software Updates to a RHEL High Availability or Resilient Storage Cluster」を参照してください。

22.7. リモートノードおよびゲストノードのアップグレード

アクティブなリモートノードまたはゲストノードで **pacemaker_remote** サービスが停止した場合に、クラスターがノードの停止前に、リソースをノードから正常に移行するようになりました。これにより、クラスターからノードを削除せずに、ソフトウェアのアップグレードやその他の定期的なメンテナンスを実行できるようになりました。**pacemaker_remote** がシャットダウンすると、クラスターは即座に再接続を試みます。リソースのモニタータイムアウトが発生する前に **pacemaker_remote** が再起動されないと、クラスターは監視操作が失敗したと判断します。

アクティブな Pacemaker リモートノードで、**pacemaker_remote** サービスが停止したときに監視が失敗しないようにするには、以下の手順に従って、**pacemaker_remote** を停止する可能性があるシステム管理を実行する前に、ノードをクラスターから削除します。

1. ノードからすべてのサービスを移行する **pcs resource disable resourcename** を使用して、ノードの接続リソースを停止します。ゲストノードの場合は、VM も停止するため、(**virsh** などを使用して) VM をクラスターの外部で起動し、メンテナンスを実行する必要があります。
2. 必要なメンテナンスを実行します。
3. ノードをクラスターに戻す準備ができたら、**pcs resource enable** でリソースを再度有効にします。