Red Hat Enterprise Linux 9

高可用性クラスターの設定および管理

Red Hat High Availability Add-On を使用した Pacemaker クラスターの作成と保守

Last Updated: 2024-06-28

Red Hat Enterprise Linux 9 高可用性クラスターの設定および管理

Red Hat High Availability Add-On を使用した Pacemaker クラスターの作成と保守

法律上の通知

概要

Red Hat High Availability Add-On は、Pacemaker クラスターリソースマネージャーを使用する高可用性クラスターを設定します。このタイトルでは、Pacemaker クラスターの設定を理解するための手順と、アクティブ/アクティブクラスターおよびアクティブ/パッシブクラスターを設定する手順の例を説明します。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **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追加してください。
5. ダイアログの下部にある **Create** 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** または **メンバー** と呼ばれています) を指します。クラスターを使用すると、可用性の高いサービスまたはリソースを提供できます。複数のマシンによ r 冗長性は、様々な障害から保護す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して、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、(read-write でマウントされたファイルシステム経由で) データの読み取りや書き込みを行います。したがって、ある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で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確認できません。また、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、高可用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。

Red Hat は、Red Hat 高可用性クラスターの計画、設定、保守に関する多様なドキュメントを提供しています。Red Hat クラスターのさまざまな分野に関するドキュメントのガイド付きインデックスを提供する記事リストについては、Red Hat High Availability Add-On Documentation Guide を参照してください。

1.1. HIGH AVAILABILITY ADD-ON コンポーネント

Red Hat High Availability Add-On は、高可用性サービスを提供する複数のコンポーネントで構成されます。

High Availability Add-On の主なコンポーネントは以下のとおりです。

- 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、ならびに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、ブロックレベルにおいて、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** に対応するには、クラスターインフラストラクチャーも必要になります。
- HAProxy - 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。

1.2. HIGH AVAILABILITY ADD-ON の概念

Red Hat High Availability Add-On クラスターの主要な概念を以下に示します。

1.2.1. フェンシング

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。当該クラスターノードが応答しない可能性があるため、当該クラスターノードと通信して行うことはできません。代わりに、フェンスエージェントを使用した、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、クラスター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、以前使用していたリソースが解放されていることを切断されているクラスターノードが把握できず、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し、データが破損または損失する可能性もあります。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度参加するまで、クラスターの通常の動作を再開することができません。

フェンシングの詳細は Fencing in a Red Hat High Availability Cluster を参照してください。

1.2.2. クォーラム

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や損失を防ぎます。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。クラスターでクォーラムが確立されない場合は、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、問題のあるノードを分離するように投票し、必要に応じて、接続を切断して別のノードに切り替えてサービスを継続 (フェンス) 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は、**スプレットブレイン** と呼ばれる状況が発生しないようにします。スプレットブレインは、クラスターが通信から分離されたあとも、各部分が別のクラスターとして機能し続けることで、同じデータの書き込みや、データの破壊または損失が発生する可能性がある現象です。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、**votequorum** サービスをフェンシングと併用します。クラスターの各システムには多くの投票数が割り当てられ、過半数の票を取得しているものだけがクラスターの操作を継続できます。

1.2.3. クラスターリソース

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インターフェイスを提供する **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** を指定することで設定できます。以下の制約のカテゴリーを設定できます。

- 場所の制約 - リソースを実行できるノードを設定する
- 順序の制約 - リソースを実行する順序を設定する
- コロケーションの制約 - 他のリソースに対して相対的なリソースの配置先を設定する

クラスターの最も一般的な設定要素の 1 つがリソースセットです。リソースセットはまとめて配置し、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。

1.3. 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。

1.3.1. 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設定されます。

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、DC (Designated Coordinator) (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の情報を分散し、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ることができます。また、リソースを移動したり、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 とリソースの間のインターフェイスとして動作する LRMd (Local Resource Manager daemon) 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、強制的にノードをシャットダウンし、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。

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になります。これは、High Availability Add-On が機能するのに必要です。

corosync は、このようなメンバーシップとメッセージング機能のほかに、以下も提供します。

- 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に渡って調整または動作するアプリケーションへの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。

1.3.2. Pacemaker の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イスは、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、ならびにクラスターの起動、停止、および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イスです。

1.3.3. クラスターと Pacemaker の設定ファイル

Red Hat High Availability Add-On の設定ファイルは、**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。通常は、直接 **corosync.conf** を編集するのではなく、**pcs** インターフェイスまたは **pcsd** インターフェイスを使用します。

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において現在の状態を表す 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** インターフェイスまたは **pcsd** インターフェイスを使用してください。

1.4. 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。

以下のクラスター設定を選択できます。

- アクティブ/パッシブのフェイルオーバー設定の HA-LVM (High Availability LVM) ボリューム。クラスターで同時に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。

1.4.1. 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では、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て、LVM メタデータを保護します。この保護は、複数のクラスター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設定することにより決まります。

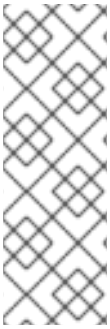
- アクティブ/パッシブで共有リソースを管理するように HA クラスターを設定し、指定した LVM ボリュームに同時にアクセスするメンバーを 1 つのみにした場合は、HA-LVM で **lvmlockd** ロックサービスを使用する必要はありません。

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できるように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、どちらの LVM を使用するかを選択します。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ように制限されているため、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。

1.4.2. クラスター内での LVM ボリュームの設定

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。



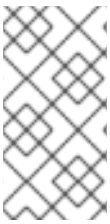
注記

Pacemaker クラスターが使用する LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように、ターゲット用に **systemd resource-agents-deps** ターゲットと **systemd** ドロップインユニットを設定することを推奨します。**systemd resource-agents-deps** ターゲットを設定する方法は、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

- 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
- **lvmlockd** デーモンを使用してストレージデバイスをアクティブ/アクティブ設定で管理する共有 LVM ボリュームを設定する手順については、クラスター内の GFS2 ファイルシステム および Red Hat High Availability クラスターでのアクティブ/アクティブ Samba サーバーの設定 を参照してください。

第2章 PACEMAKER の使用の開始

Pacemaker クラスターの作成に使用するツールとプロセスに慣れるために、次の手順を実行できます。ここで説明する内容は、クラスターソフトウェアの概要と、作業用のクラスターを設定せずに管理する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。Red Hat のサポートポリシー、要件、および制限の詳細は、RHEL 高可用性クラスターのサポートポリシー を参照してください。

2.1. PACEMAKER の使用方法

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。

この例では、以下のように設定されています。

- ノード: **z1.example.com**
- Floating IP アドレス: 192.168.122.120

前提条件

- RHEL 9 を実行しているノード 1 つ
- このノードで静的に割り当てられている IP アドレスの 1 つと同じネットワーク上にある Floating IP アドレス
- **/etc/hosts** ファイルに、実行中のノード名が含まれている

手順

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** サービスを起動して有効にします。

```
# dnf 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にあるクラスターの各ノードに対して、**hacluster** ユーザーの認証を行います。この例では、ノードを 1 つだけ使用し、そのノードからコマンドを実行していますが、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、この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 1 つのコマンドで、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。この要件が必要になる理由は Fencing in a Red Hat High Availability Cluster を参照してください。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 の使用は、実稼働クラスターには完全に適していません。これにより、障害が発生したノードが適切にフェンスされていることを装うようにクラスターに指示されます。

```
# 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

システムの起動時に使用する場合は、**systemctl enable** で、クラスターが管理するサービスを有効にしないでください。

```
# dnf 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リソース **IPaddr2** および **apache** を作成します。IPaddr2 は Floating 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 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 Floating 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リストを表示する場合は、**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場合は、**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** 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 アドレスリソースと apache リソースの両方が **apachegroup** 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 Floating 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を確認する際に、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. クラスターと、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。(この手順を試すために、実際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てください。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と、サービスを実行しているノードでサービス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、どのようにしてサービスを利用可能のままにしているかを確認できます。

この例では、以下のように設定されています。

- ノード: **z1.example.com** および **z2.example.com**
- Floating IP アドレス: 192.168.122.120

前提条件

- 相互に通信が可能な RHEL 9 を実行するノード 2 つ
- このノードで静的に割り当てられている IP アドレスの 1 つと同じネットワーク上にある Floating IP アドレス
- **/etc/hosts** ファイルに、実行中のノード名が含まれている

手順

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** サービスを起動して有効にします。

```
# dnf 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、両方のノードで、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、**hacluster** ユーザー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に対して **hacluster** ユーザーの認証を行います。

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

4. 両方のノードで、クラスターメンバーとなるクラスター **my_cluster** を作成します。この1つのコマンドで、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。この要件が必要になる理由は Fencing in a Red Hat High Availability Cluster を参照してください。ただし、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示す場合は、**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 の使用は、実稼働クラスターには完全に適していません。これにより、障害が発生したノードが適切にフェンスされていることを装うようにクラスターに指示されます。

```
# 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。

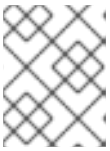
注記

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

システムの起動時に使用する場合は、**systemctl enable** で、クラスターが管理するサービスを有効にしないでください。

```
# dnf 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リソース **IPaddr2** および **apache** を作成します。IPaddr2 は Floating 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 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 Floating 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リストを表示する場合は、**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場合は、**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** 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、次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 Floating 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するため、Web サイトに引き続きアクセスできるはずです。

      ```
      # 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) ステータスを削除します。

```
# 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。フェンシングを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。サービスの切断はありません。表示メッセージには、サービスを実行しているノードが含まれるはずです。

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 node 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イス

pcs コマンドラインインターフェイスを使用すると、**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。

次のコマンドは、**pcs resource** コマンドのパラメーターを表示します。

```
# pcs resource -h
```

3.2. 未編集のクラスター設定の表示

クラスター設定ファイルは直接編集しないようにしてください。未編集のクラスター設定は、**pcs cluster cib** コマンドで表示できます。

pcs cluster cib filename コマンドを使用すると、未編集のクラスター設定を、指定したファイルに保存できます。クラスターを事前に設定していて、アクティブな CIB が存在する場合は、以下のコマンドを実行して、未編集の 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、未編集の CIB の xml ファイルが保存されます。

```
# 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際に、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、未編集の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。

手順

以下は、CIB のファイルに変更をプッシュする際に推奨される手順です。この手順は、保存した 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は、ここで、**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、変更したファイルの差異だけが 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、設定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. 更新したファイルを、アクティブな CIB にプッシュします。元のファイルに加えた変更のみをプッシュするように指定します。

```
# 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て、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

さまざまなコマンドを使用して、クラスターおよびそのコンポーネントのステータスを表示できます。

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
```

3.6. PCS コマンドによる COROSYNC.CONF ファイルの変更

pcs コマンドを使用して、**corosync.conf** ファイルのパラメーターを変更できます。

次のコマンドは、**corosync.conf** ファイルのパラメーターを変更します。

```
pcs cluster config update [transport pass:quotes[transport options]] [compression pass:quotes[compression options]] [crypto pass:quotes[crypto options]] [totem pass:quotes[totem options]] [--corosync_conf pass:quotes[path]]
```

以下のコマンド例は、**knet_pmtud_interval** トランスポート値と、**token** と **join** の totem の値を更新します。

```
# pcs cluster config update transport knet_pmtud_interval=35 totem token=10000 join=100
```

関連情報

- 既存クラスターにノードを追加および削除する方法は、クラスターノードの管理 を参照してください。
- 既存クラスターのリンクの追加および変更の詳細は、既存クラスターへのリンクの追加および変更 を参照してください。
- クラスターでクォーラムオプションを変更する方法およびクォーラムデバイスの設定を管理する方法は、クラスタークォーラムの設定 および クォーラムデバイスの設定 を参照してください。

3.7. PCS コマンドでの COROSYNC.CONF ファイルの表示

次のコマンドは、**corosync.conf** クラスター設定ファイルの内容を表示します。

```
# pcs cluster corosync
```

以下のように **pcs cluster config** コマンドで、人間が判読できる形式で **corosync.conf** ファイルの内容を出力できます。

クラスターが RHEL 9.1 以降で作成された場合、または UUID によるクラスターの識別 で説明されているように UUID が手動で追加された場合、このコマンドの出力にはクラスターの UUID が含まれます。

```
[root@r8-node-01 ~]# pcs cluster config
Cluster Name: HACluster
Cluster UUID: ad4ae07dcafe4066b01f1cc9391f54f5
Transport: knet
Nodes:
  r8-node-01:
    Link 0 address: r8-node-01
```

```
    Link 1 address: 192.168.122.121
    nodeid: 1
  r8-node-02:
    Link 0 address: r8-node-02
    Link 1 address: 192.168.122.122
    nodeid: 2
Links:
  Link 1:
    linknumber: 1
    ping_interval: 1000
    ping_timeout: 2000
    pong_count: 5
Compression Options:
  level: 9
  model: zlib
  threshold: 150
Crypto Options:
  cipher: aes256
  hash: sha256
Totem Options:
  downcheck: 2000
  join: 50
  token: 10000
Quorum Device: net
  Options:
    sync_timeout: 2000
    timeout: 3000
  Model Options:
    algorithm: lms
    host: r8-node-03
  Heuristics:
    exec_ping: ping -c 1 127.0.0.1
```

--output-format=cmd オプションを指定して **pcs cluster config show** コマンドを実行し、以下の例のように、既存の **corosync.conf** ファイルの再作成に使用できる **pcs** 設定コマンドを表示できます。

```
[root@r8-node-01 ~]# pcs cluster config show --output-format=cmd
pcs cluster setup HACluster \
  r8-node-01 addr=r8-node-01 addr=192.168.122.121 \
  r8-node-02 addr=r8-node-02 addr=192.168.122.122 \
  transport \
  knet \
    link \
      linknumber=1 \
      ping_interval=1000 \
      ping_timeout=2000 \
      pong_count=5 \
    compression \
      level=9 \
      model=zlib \
      threshold=150 \
    crypto \
      cipher=aes256 \
      hash=sha256 \
  totem \
```

```
downcheck=2000 \
join=50 \
token=10000
```

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** コマンドラインインターフェイスを使用して、2 ノードの Red Hat High Availability クラスターを作成します。

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シングデバイス。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

以下の手順では、クラスターソフトウェアをインストールし、システムでクラスターの作成を設定するように設定します。

手順

1. クラスター内の各ノードで、システムアーキテクチャーに対応する高可用性のリポジトリーを有効にします。たとえば、x86_64 システムの高可用性リポジトリーを有効にするには、以下の **subscription-manager** コマンドを入力します。

```
# subscription-manager repos --enable=rhel-9-for-x86_64-highavailability-rpms
```

2. 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。

```
# dnf install pcs pacemaker fence-agents-all
```

または、次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。

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

次のコマンドは、利用できるフェンスエージェントのリストを表示します。

```
# 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にインストールすると、予期しない動作が発生する可能性があります。詳細は 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更新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。

3. **firewalld** デーモンを実行している場合は、次のコマンドを実行して、Red Hat High Availability Add-On で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場合は、**rpm -q firewalld** コマンドを実行します。firewalld デーモンがインストールされている場合は、**firewall-cmd --state** コマンドで、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イスがあるかどうか、オフホストのファイアウォールがあるかどうかを検討しないといけない場合があります。この例では、Pacemaker クラスターで通常必要となるポートを開きますが、ローカル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 で有効にするポートと、各ポートの使用目的が説明されています。

4. **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.
```

5. クラスターを設定する前に、各ノードで、システムの起動時に **pcsd** デーモンを起動できるように、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、システムの起動時に **pcsd** サービスが起動し、**pcsd** が有効になるように設定します。

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

4.2. PCP-ZEROCONF パッケージのインストール (推奨)

クラスターを設定する際に、PCP (Performance Co-Pilot) ツールの **pcp-zeroconf** パッケージをインストールすることが推奨されます。PCP は、RHEL システムに推奨される Red Hat の resource-monitoring ツールです。**pcp-zeroconf** パッケージをインストールすると、PCP を実行してパフォーマンス監視データを収集して、フェンシング、リソース障害、およびクラスターを中断するその他のイベントの調査に役立てることができます。

注記

PCP が有効になっているクラスターデプロイメントには、**/var/log/pcp/** を含むファイルシステムで PCP が取得したデータ用に十分な領域が必要です。PCP による一般的な領域使用はデプロイメントごとに異なりますが、通常 **pcp-zeroconf** のデフォルト設定を使用する場合は 10Gb で十分であり、環境によっては必要な量が少なくなることがあります。一般的なアクティビティーの 14 日間におけるこのディレクトリーの使用状況を監視すると、より正確な使用状況の概算値が得られます。

手順

pcp-zeroconf パッケージをインストールするには、次のコマンドを実行します。

```
# dnf install pcp-zeroconf
```

このパッケージは **pmcd** を有効にし、10 秒間隔でデータキャプチャーを設定します。

PCP データを確認する方法は、Red Hat カスタマーポータルの Why did a RHEL High Availability cluster node reboot - how can I prevent it again を参照してください。

4.3. 高可用性クラスターの作成

以下の手順では、Red Hat High Availability Add-On クラスターを作成します。この手順の例では、ノード **z1.example.com** および **z2.example.com** で構成されるクラスターを作成します。

手順

1. **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。
   次のコマンドは、**z1.example.com** と **z2.example.com** で構成される 2 ノードクラスターの両ノードに対して、**z1.example.com** の **hacluster** ユーザーを認証します。

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

2. **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 **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、クラスターの両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、クラスターの両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
```

3. 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のまま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 **pcs cluster start** コマンドを使用して手作業でサービスを起動しなければならないので注意してください。

```
[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のに時間が少し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

...
```

4.4. 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基本的なコマンドの形式は、以下のとおりです。

```
pcs cluster setup pass:quotes[cluster_name] pass:quotes[node1_name]
addr=pass:quotes[node1_link0_address] addr=pass:quotes[node1_link1_address]
pass:quotes[node2_name] addr=pass:quotes[node2_link0_address]
addr=pass:quotes[node2_link1_address]
```

このコマンドの完全な構文は、**pcs**(8) の man ページを参照してください。

複数のリンクを持つクラスターを作成する場合は、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** に使用され、2 番目以降のアドレスは **link1** 以降に順に使用されます。

- デフォルトでは、リンクに **link_priority** が指定されていない場合、リンクの優先度はリンク番号と同じになります。リンクの優先度は、指定された順序に従って 0、1、2、3 などになり、0 はリンクの優先順位が最も高くなります。
- デフォルトのリンクモードは **passive** で、リンク優先度の数字が最も小さいアクティブなリンクが使用されます。
- **link_mode** および **link_priority** のデフォルト値では、指定された最初のリンクが最も優先度の高いリンクとして使用され、そのリンクが失敗した場合は、指定された次のリンクが使用されます。
- 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。
- **pcs cluster link add** コマンド、**pcs cluster link remove** コマンド、**pcs cluster link delete** コマンド、および **pcs cluster link update** コマンドを使用して、既存クラスターのリンクを追加、削除、および変更できます。
- シングルリンククラスターと同様、1 つのリンクで IPv4 アドレスと IPv6 アドレスを混在させないでください。ただし、1 つのリンクで IPv4 を実行し、別のリンクで IPv6 を実行することはできます。
- シングルリンククラスターと同様、1 つのリンクで IPv4 アドレスと IPv6 アドレスが混在しない IPv4 アドレスまたは IPv6 アドレスで名前が解決される限り、アドレスを IP アドレスまたは名前として指定できます。

以下の例では、**rh80-node1** と **rh80-node2** の 2 つのノードがある **my_twolink_cluster** という名前の 2 ノードクラスターを作成します。**rh80-node1** には、IP アドレスが 192.168.122.201 (**link0**) と 192.168.123.201 (**link1**) の 2 つのインターフェイスがあります。**rh80-node2** には IP アドレスが 192.168.122.202 (**link0**) と 192.168.123.202 (**link1**) の 2 つのインターフェイスがあります。

```
# 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
```

リンク優先度を、リンク番号であるデフォルト値とは異なる値に設定するには、**pcs cluster setup** コマンドの **link_priority** オプションを使用してリンクの優先度を設定します。以下の 2 つのコマンド例のそれぞれは、2 つのインターフェイスを持つ 2 ノードクラスターを作成し、最初のリンクであるリンク 0 のリンク優先度は 1 で、2 番目のリンクであるリンク 1 のリンク優先度は 0 です。リンク 1 が最初に使用され、リンク 0 はフェイルオーバーリンクとして機能します。リンクモードが指定されていないため、デフォルトの passive に設定されます。

これら 2 つのコマンドは同等です。**link** キーワードの後にリンク番号を指定しないと、**pcs** インターフェイスは使用されていない最も小さいリンク番号からリンク番号を自動的に追加します。

```
# 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 transport knet
link link_priority=1 link link_priority=0

# 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 transport knet
link linknumber=1 link_priority=0 link link_priority=1
```

以下の例のように、**pcs cluster setup** コマンドの **link_mode** オプションを使用して、リンクモードをデフォルト値の **passive** とは異なる値に設定できます。

```
# 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 transport knet
link_mode=active
```

以下の例では、リンクモードとリンク優先度の両方を設定します。

```
# 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 transport knet
link_mode=active link link_priority=1 link link_priority=0
```

リンクが複数ある既存クラスターにノードを追加する方法は、複数のリンクを持つクラスターへのノードの追加 を参照してください。

リンクが複数ある既存クラスターのリンクを変更する方法は、既存クラスターへのリンクの追加および変更 を参照してください。

4.5. フェンシングの設定

クラスターの各ノードにフェンシングデバイスを設定する必要があります。フェンシング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 Fencing in a Red Hat High Availability Cluster を参照してください。

注記

フェンシングデバイスを設定する場合は、そのデバイスが、クラスター内のノードまたはデバイスと電源を共有しているかどうかに注意する必要があります。ノードとそのフェンスデバイスが電源を共有していると、その電源をフェンスできず、フェンスデバイスが失われた場合は、クラスターがそのノードをフェンスできない可能性があります。このようなクラスターには、フェンスデバイスおよびノードに冗長電源を提供するか、電源を共有しない冗長フェンスデバイスが存在する必要があります。SBD やストレージフェンシングなど、その他のフェンシング方法でも、分離した電源供給の停止時に冗長性を得られます。

手順

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スし、**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** オプションを使用して、両方のフェンシングデバイスを 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、フェンシング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、**stonith** リソース **myapc** を設定します。**pcmk_host_map** オプションにより、**z1.example.com** がポート 1 にマップされ、**z2.example.com** がポート 2 にマップされます。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。

また、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ます。

```
[root@z1 ~]# pcs stonith create myapc fence_apc_snmp ipaddr="zapc.example.com"
pcmk_host_map="z1.example.com:1;z2.example.com:2" login="apc" passwd="apc"
```

以下のコマンドは、既存のフェンシング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
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。フェンスデバイスをテストする方法は 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イ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ります。

注記

フェンシングを設定してクラスターが起動すると、タイムアウトに到達していなくても、ネットワークの再起動時に、ネットワークを再起動するノードのフェンシングが発生します。このため、ノードで意図しないフェンシングが発生しないように、クラスターサービスの実行中はネットワークサービスを再起動しないでください。

4.6. クラスター設定のバックアップおよび復元

以下のコマンドは、tar アーカイブのクラスター設定のバックアップを作成し、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。

手順

以下のコマンドを使用して、tar アーカイブでクラスター設定をバックアップします。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の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クラスターノードのクラスター設定ファイルを復元します。**--local** オプションを指定すると、このコマンドを実行したノードでのみクラスター設定ファイルが復元されます。ファイル名を指定しないと、標準入力が使用されます。

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

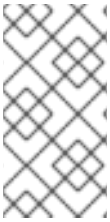
4.7. HIGH AVAILABILITY ADD-ON のポートの有効化

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イスがあるかどうか、オフホストのファイアウォールがあるかどうかを検討しないといけない場合があります。

firewalld デーモンを実行している場合は、次のコマンドを実行して、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 --add-service=high-availability
```

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場合は、**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、そのデーモンが実行しているかどうかを確認できます。

以下の表は、Red Hat High Availability Add-On で有効にするポートとポートの使用目的を説明します。

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** ポートで必須 (pcsd Web UI で必要、ノード間通信で必須) です。**/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。 ポート 2224 を開いて、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ようにする必要があります。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、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。

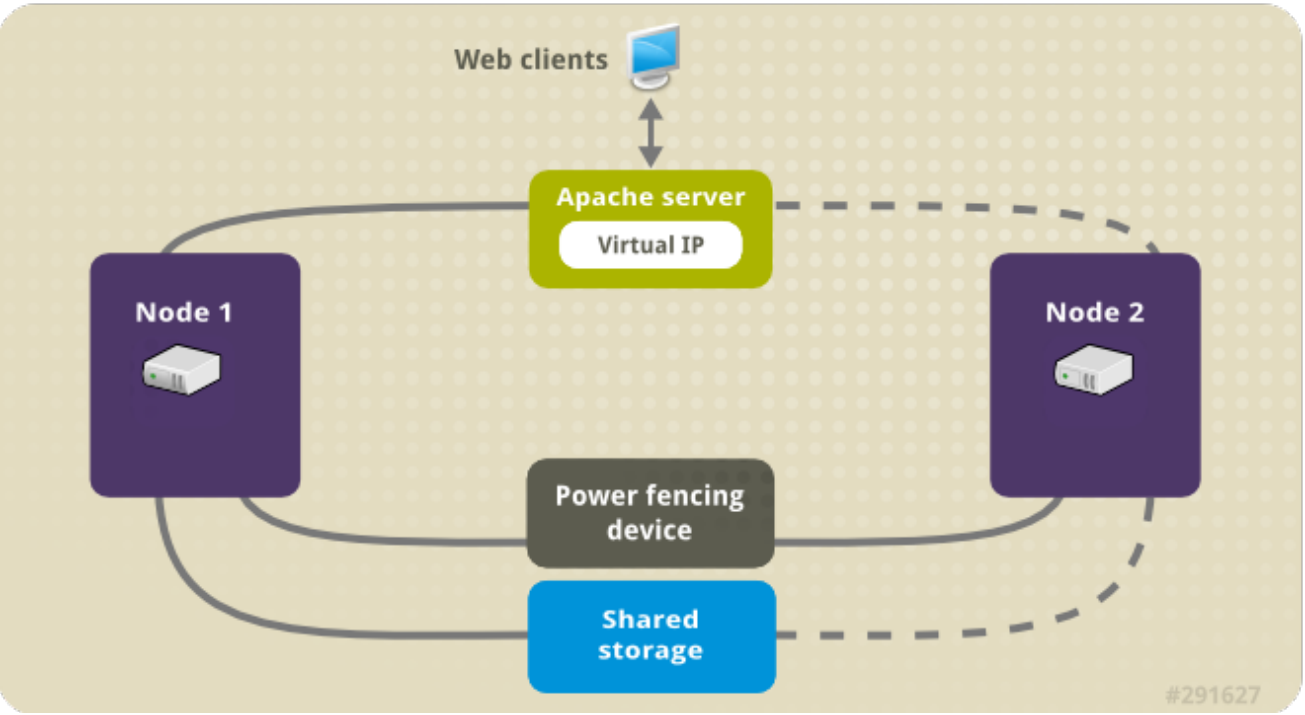
ポート	必要になる場合
TCP 3121	クラスターに Pacemaker リモートノードがある場合に、すべてのノードで必須です。 完全なクラスターノードにある Pacemaker の **pacemaker-based** デーモンは、ポート 3121 で Pacemaker リモートノードの **pacemaker_remoted** デーモンへの通信を行います。クラスター通信に別のインターフェイスを使用する場合は、そのインターフェイスでポートを開くことのみが必要になります。少なくとも、ポートは、Pacemaker リモートノードの全クラスターノードに対して開いている必要があります。ユーザ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、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、すべてのノードにポートを開くと便利になる場合があります。ノード以外のホストに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クォーラムデバイスホストで必須です。デフォルト値は、**corosync-qnetd** コマンドの **-p** オプションで変更できます。
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。ポート 5404-5412 を開いて、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する必要があります。
TCP 21064	DLM が必要なリソースがクラスターに含まれる場合に、すべてのノードで必須です (例: **GFS2**)。
TCP 9929、UDP 9929	Booth チケットマネージャーを使用してマルチサイトクラスターを確立するときに、すべてのクラスターノード、および同じノードのいずれかからの接続に対して Booth arbitrator ノードで開いている必要があります。

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、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 Apache HTTP サーバーを設定します。このユースケースでは、クライアントは Floating 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、クラスターにある 2 つのノードのいずれかで実行します。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 番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。

以下の図は、クラスターがネットワーク電源スイッチと共有ストレージで設定された 2 ノードの Red Hat High Availability クラスターであるクラスターの高レベルの概要を示しています。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 のいずれかで実行します。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。この図では、Web サーバーが ノード 1 で実行しており、ノード 1 が正常に動作しなくなると、ノード 2 がサーバーを実行できます。

図5.1 2 ノードの 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、ファイバーチャネル、またはその他の共有ネットワークデバイスを使用する、クラスター内のノードの共有ストレージ。

クラスターは、Web サーバーで必要な 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設定されます。このリソースグループは、クラスター内のあるノードから別のノードへのフェイルオーバーが

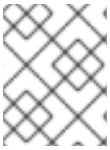
可能なため、いずれのノードでも Web サーバーを実行できます。このクラスターのリソースグループを作成する前に、以下の手順を実行します。

1. 論理ボリューム **my_lv** 上に XFS 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 XFS ファイルシステムを使用して LVM ボリュームを設定する

この手順では、クラスターのノード間で共有されているストレージに LVM 論理ボリュームを作成します。



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。

次の手順では、LVM 論理ボリュームを作成し、そのボリューム上に Pacemaker クラスターで使用する XFS ファイルシステムを作成します。この例では、LVM 論理ボリュームを作成する LVM 物理ボリュームを保管するのに、共有パーティション **/dev/sdb1** が使用されます。

手順

1. クラスターの両ノードで以下の手順を実行し、LVM システム ID の値を、システムの **uname** 識別子の値に設定します。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なります。

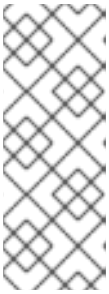
   a. **/etc/lvm/lvm.conf** 設定ファイルの **system_id_source** 設定オプションを **uname** に設定します。

   ```
   # 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、そのボリューム上に XFS 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、1 つのノードでのみ実行してください。



注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

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。

```
[root@z1 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

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。**--setautoactivation n** フラグを指定して、クラスターで Pacemaker が管理するボリュームグループが起動時に自動的にアクティブにならないようにします。作成する LVM ボリュームに既存のボリュームグループを使用している場合は、ボリュームグループで **vgchange --setautoactivation n** コマンドを使用して、このフラグをリセットできます。

```
[root@z1 ~]# vgcreate --setautoactivation n 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。

```
[root@z1 ~]# 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。

```
[root@z1 ~]# lvcreate -L450 -n my_lv my_vg
  Rounding up size to full physical extent 452.00 MiB
  Logical volume "my_lv" created
```

lvs コマンドを使用して論理ボリュームを表示してみます。

```
[root@z1 ~]# 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** 上に XFS ファイルシステムを作成します。

```
[root@z1 ~]# mkfs.xfs /dev/my_vg/my_lv
meta-data=/dev/my_vg/my_lv       isize=512    agcount=4, agsize=28928 blks
         =                       sectsz=512   attr=2, projid32bit=1
...
```

3. **lvm.conf** ファイルの **use_devicesfile = 1** パラメーターでデバイスファイルの使用が有効になっている場合は、クラスター内の 2 番目のノードのデバイスファイルに共有デバイスを追加します。この機能はデフォルトで有効化されています。

```
[root@z2 ~]# lvmdevices --adddev /dev/sdb1
```

5.2. APACHE HTTP サーバーの設定

次の手順に従って Apache HTTP サーバーを設定します。

手順

1. クラスターの各ノードに、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確認するには、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。

```
# dnf install -y httpd wget
```

   firewalld デーモンを実行している場合は、クラスターの各ノードで、Red Hat High Availability Add-On に必要なポートを有効にし、**httpd** の実行に必要なポートを有効にします。以下の例では、一般からのアクセス用に **httpd** のポートを有効にしていますが、**httpd** 用に有効にする具体的なポートは、本番環境のユースケースでは異なる場合があります。

```
# firewall-cmd --permanent --add-service=http
# firewall-cmd --permanent --zone=public --add-service=http
# firewall-cmd --reload
```

2. Apache リソースエージェントが、クラスターの各ノードで 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
   Require local
</Location>
END
```

3. Apache で提供する Web ページを作成します。
   クラスター内の 1 つのノードで、XFS ファイルシステムを使用した LVM ボリュームの設定 で作成した論理ボリュームがアクティブになっていることを確認し、作成したファイルシステムをその論理ボリュームにマウントし、そのファイルシステムにファイル **index.html** を作成します。次に、ファイルシステムをアンマウントします。

```
# lvchange -ay my_vg/my_lv
# 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

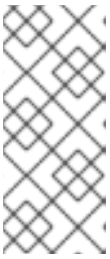
次の手順でクラスターのリソースを作成します。すべてのリソースが必ず同じノードで実行するように、このリソースを、リソースグループ **apachegroup** に追加します。作成するリソースは以下のとおりで、開始する順に記載されています。

1. XFS ファイルシステムを使用した LVM ボリュームの設定 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-activate** リソース。
2. XFS ファイルシステムを使用した LVM ボリュームの設定 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 Floating 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 デバイスを指定していない場合は、そのノードに静的に割り当てられている IP アドレスの 1 つと同じネットワークに Floating IP が存在していないと、Floating 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. Apache HTTP サーバーの設定で定義した **index.html** ファイルと Apache 設定を使用する **Website** という名前の **apache** リソース

以下の手順で、**apachegroup** リソースグループと、このグループに追加するリソースを作成します。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。

手順

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

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** なリソースを複数設定するとデータが破損する場合があるため、そのようなリソースは 1 つ以上設定しないでください。また、**LVM が有効** なリソースは、アクティブ/パッシブの HA 設定のクローンリソースとして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すると、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、作成したリソースを既存の **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="xfs" --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-activate): 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
```

クラスターのフェンシング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。

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. **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 を再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
```

削除した行を次の 3 行に置き換え、PID ファイルパスとして **/var/run/httpd-website.pid** を指定します。ここの **website** は、Apache リソースの名前になります。この例では、Apache リソース名は **Website** です。

```
/usr/bin/test -f /var/run/httpd-Website.pid >/dev/null 2>/dev/null &&
/usr/bin/ps -q $(/usr/bin/cat /var/run/httpd-Website.pid) >/dev/null 2>/dev/null &&
/usr/sbin/httpd -f /etc/httpd/conf/httpd.conf -c "PidFile /var/run/httpd-Website.pid" -k graceful >
/dev/null 2>/dev/null || true
```

5.4. リソース設定のテスト

次の手順でクラスター内のリソース設定をテストします。

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** ノードで実行されます。以下の手順に従い、1 番目のノードを **スタンバイ** モードにし、リソースグループが **z2.example.com** ノードにフェイルオーバーするかどうかをテストします。1 番目のノードをスタンバイモードにすると、このノードはリソースをホストできなくなります。

手順

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-activate): 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 node 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。これは、リソースの **resource-stickiness** 値により異なります。**resource-stickiness** メタ属性については、現在のノードを優先するようにリソースを設定する を参照してください。

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

Red Hat High Availability Add-On は、共有ストレージを使用して Red Hat Enterprise Linux High Availability アドオンクラスターで高可用性アクティブ/パッシブ NFS サーバーを実行するためのサポートを提供します。次の例では、クライアントが Floating IP アドレスを介して NFS ファイルシステムにアクセスする 2 ノードクラスターを設定します。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 番目のノードで再起動し、サービスの中断が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。

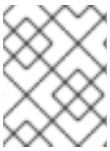
- 各ノードに電源フェンスが設定されている 2 ノードの Red Hat High Availability クラスター。プライベートネットワークが推奨されますが、必須ではありません。この手順では、Pacemaker を使用した Red Hat High Availability クラスターの作成 で説明されているサンプルのクラスターを使用します。
- NFS サーバーに必要なパブリック仮想 IP アドレス。
- iSCSI、ファイバーチャネル、またはその他の共有ネットワークデバイスを使用する、クラスター内のノードの共有ストレージ。

既存の 2 ノードの 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、以下の手順を実行する必要があります。

1. クラスターのノード用に、共有ストレージの LVM 論理ボリュームにファイルシステムを設定します。
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 XFS ファイルシステムを使用して LVM ボリュームを設定する

この手順では、クラスターのノード間で共有されているストレージに LVM 論理ボリュームを作成します。



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。

次の手順では、LVM 論理ボリュームを作成し、そのボリューム上に Pacemaker クラスターで使用する XFS ファイルシステムを作成します。この例では、LVM 論理ボリュームを作成する LVM 物理ボリュームを保管するのに、共有パーティション **/dev/sdb1** が使用されます。

手順

1. クラスターの両ノードで以下の手順を実行し、LVM システム ID の値を、システムの **uname** 識別子の値に設定します。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なります。

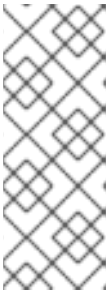
a. **/etc/lvm/lvm.conf** 設定ファイルの **system_id_source** 設定オプションを **uname** に設定します。

```
# 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、そのボリューム上に XFS 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、1 つのノードでのみ実行してください。

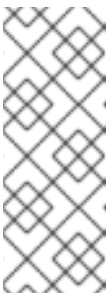


注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

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。

```
[root@z1 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```



注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

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。**--setautoactivation n** フラグを指定して、クラスターで Pacemaker が管理するボリュームグループが起動時に自動的にアクティブにならないようにします。作成する LVM ボリュームに既存のボリュームグループを使用している場合は、ボリュームグループで **vgchange --setautoactivation n** コマンドを使用して、このフラグをリセットできます。

```
[root@z1 ~]# vgcreate --setautoactivation n 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。

```
[root@z1 ~]# 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。

```
[root@z1 ~]# lvcreate -L450 -n my_lv my_vg
  Rounding up size to full physical extent 452.00 MiB
  Logical volume "my_lv" created
```

lvs コマンドを使用して論理ボリュームを表示してみます。

```
[root@z1 ~]# 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** 上に XFS ファイルシステムを作成します。

```
[root@z1 ~]# mkfs.xfs /dev/my_vg/my_lv
meta-data=/dev/my_vg/my_lv       isize=512    agcount=4, agsize=28928 blks
         =                       sectsz=512   attr=2, projid32bit=1
...
```

3. **lvm.conf** ファイルの **use_devicesfile = 1** パラメーターでデバイスファイルの使用が有効になっている場合は、クラスター内の 2 番目のノードのデバイスファイルに共有デバイスを追加します。この機能はデフォルトで有効化されています。

```
[root@z2 ~]# lvmdevices --adddev /dev/sdb1
```

6.2. NFS 共有の設定

次の手順では、NFS サービスのフェイルオーバー用の NFS 共有を設定します。

手順

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. XFS ファイルシステムを使用した LVM ボリュームの設定 で作成した論理ボリュームを確認します。アクティブ化されたら、**/nfsshare** ディレクトリーの論理ボリューム上に作成したファイルシステムをマウントします。

```
[root@z1 ~]# lvchange -ay my_vg/my_lv
[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、テストファイル **clientdatafile1** および **clientdatafile2** を作成します。

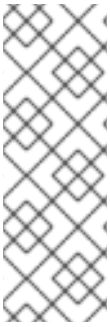
```
[root@z1 ~]# touch /nfsshare/exports/export1/clientdatafile1
[root@z1 ~]# touch /nfsshare/exports/export2/clientdatafile2
```

d. 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化します。

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

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

以下の手順で、クラスター内の NFS サーバーのクラスターリソースを設定します。



注記

クラスターにフェンシング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。

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。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。

手順

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. 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。

警告

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** なリソースを複数設定しないでください。また、**LVM が有効** なリソースは、アクティブ/パッシブの HA 設定のクローンリソースとして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
```

2. 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-activate):   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
```

3. クラスターに **Filesystem** リソースを設定します。
次のコマンドは、**nfsshare** という名前の XFS **Filesystem** リソースを **nfsgroup** リソースグループの一部として設定します。このファイルシステムは、XFS ファイルシステムを使用した LVM ボリュームの設定 で作成した LVM ボリュームグループと XFS ファイルシステムを使用し、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。

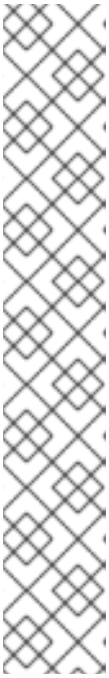
```
[root@z1 ~]# pcs resource create nfsshare Filesystem device=/dev/my_vg/my_lv
directory=/nfsshare fstype=xfs --group nfsgroup
```

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。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。

4. **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-activate):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
...
```

5. **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 サーバーが、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 サーバーは、このリソースグループを再配置する必要がある場合に別のノードで使用できるデバイスに、ステートフル情報を保存します。この例では、以下のように設定されています。

- **/nfsshare** は、**Filesystem** リソースにより管理される shared-storage ディレクトリーです。
- **/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
...
```

6. **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。

注記

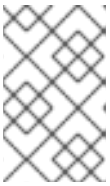
fsid=0 オプションは、NFSv4 クライアントに仮想ディレクトリーを作成する場合にのみ必要です。詳細は、How do I configure the fsid option in an NFS server's /etc/exports file? を参照してください。//

```
[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
```

7. 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 Floating IP アドレス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。このデプロイメント例では、192.168.122.200 を Floating 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
```

8. 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、Floating IP アドレスにホスト名が関連付けられており、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
```

9. 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確認できます。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-activate):   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

以下の手順を使用して、高可用性クラスターで NFS リソース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。

6.4.1. NFS エクスポートのテスト

1. クラスターノードで **firewalld** デーモンを実行している場合は、システムが NFS アクセスに必要とするポートがすべてのノードで有効になっていることを確認してください。

2. 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 共有をマウントして、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
```

3. NFSv4 で NFS 共有をマウントできることを確認する場合は、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
```

4. NFSv3 で NFS 共有をマウントできることを確認します。マウントしたら、テストファイル **clientdatafile1** が表示されてい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。テスト後に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
```

6.4.2. フェイルオーバーのテスト

1. 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** ファイル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
```

2. クラスター内で、**nfsgroup** を実行しているノードを確認します。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-activate):   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
...
```

3. 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
```

4. **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。

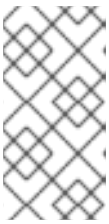
```
[root@z1 ~]# pcs status
...
Full list of resources:
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM-activate):   Started z2.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z2.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z2.example.com
    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

5. NFS 共有をマウントしたクラスターの外部のノードから、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が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、マウントの復旧に最大 90 秒かかることがあります。この 90 秒は、システムの起動時にサーバーが監視する 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。

6. 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。



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。これは、リソースの **resource-stickiness** 値により異なります。**resource-stickiness** メタ属性については、現在のノードを優先するようにリソースを設定する を参照してください。

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

第7章 クラスター内の GFS2 ファイルシステム

Red Hat 高可用性クラスターで GFS2 ファイルシステムを設定するには、次の管理手順を使用します。

7.1. クラスターに GFS2 ファイルシステムを設定

次の手順で、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できます。この例では、2 ノードクラスター内の 3 つの論理ボリューム上に 3 つの GFS2 ファイルシステムを作成します。

前提条件

- 両方のクラスターノードにクラスターソフトウェアをインストールして起動し、基本的な 2 ノードクラスターを作成している。
- クラスターのフェンシングを設定している。

Pacemaker クラスターの作成とクラスターのフェンシングの設定については、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。

手順

1. クラスター内の両方のノードで、システムアーキテクチャーに対応する Resilient Storage のリポジトリーを有効にします。たとえば、x86_64 システムの Resilient Storage リポジトリーを有効にするには、以下の **subscription-manager** コマンドを入力します。

   ```
   # subscription-manager repos --enable=rhel-9-for-x86_64-resilientstorage-rpms
   ```

   Resilient Storage リポジトリーは、High Availability リポジトリーのスーパーセットであることに注意してください。Resilient Storage リポジトリーを有効にする場合は、High Availability リポジトリーを有効にする必要はありません。

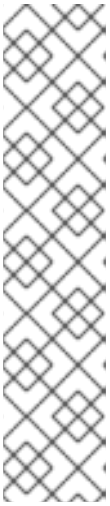
2. クラスターの両方のノードで、**lvm2-lockd** パッケージ、**gfs2-utils** パッケージ、および **dlm** パッケージをインストールします。AppStream チャンネルおよび Resilient Storage チャンネルにサブスクライブして、これらのパッケージをサポートする必要があります。

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

3. クラスターの両方のノードで、**/etc/lvm/lvm.conf** ファイルの **use_lvmlockd** 設定オプションを **use_lvmlockd=1** に設定します。

   ```
   ...
   use_lvmlockd = 1
   ...
   ```

4. グローバル Pacemaker パラメーター **no-quorum-policy** を **freeze** に設定します。



注記

デフォルトでは、**no-quorum-policy** の値は **stop** に設定され、定足数が失われると、残りのパーティションのリソースがすべて即座に停止されます。通常、このデフォルト設定は最も安全なオプションで最適なおプションですが、ほとんどのリソースとは異なり、GFS2 が機能するにはクォーラムが必要です。クォーラムが失われると、GFS2 マウントを使用したアプリケーション、GFS2 マウント自体の両方が正しく停止できません。クォーラムなしでこれらのリソースを停止しようとすると失敗し、最終的にクォーラムが失われるたびにクラスター全体がフェンスされます。

この状況に対処するには、GFS2 の使用時の **no-quorum-policy** を **freeze** に設定します。この設定では、クォーラムが失われると、クォーラムが回復するまで残りのパーティションは何もしません。

```
[root@z1 ~]# pcs property set no-quorum-policy=freeze
```

5. **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、リソースグループ **locking** に追加します。

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

6. 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
```

7. **locking** リソースグループの一部として **lvmlockd** リソースを設定します。

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

8. 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 ]
```

9. 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。

注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

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...
```

10. クラスター内の 2 番目のノードで以下を実行します。

    a. **lvm.conf** ファイルの **use_devicesfile = 1** パラメーターでデバイスファイルの使用が有効になっている場合は、共有デバイスをデバイスファイルに追加します。この機能はデフォルトで有効になっています。

    ```
    [root@z2 ~]# lvmdevices --adddev /dev/vdb
    [root@z2 ~]# lvmdevices --adddev /dev/vdc
    ```

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

11. 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。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。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。ロックテーブル名の形式は、**ClusterName:FSName** です。**ClusterName** は、GFS2 ファイルシステムが作成されているクラスターの名前です。**FSname** はファイルシステム名です。これは、クラスター経由のすべての **lock_dlm** ファイルシステムで一意である必要があります。

```
[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
```

12. 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、各論理ボリュームに **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
    ```

13. 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
    ```

14. **dlm** リソースおよび **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)
```

15. コロケーション制約を設定して、**vg1** および **vg2** のリソースグループが **locking** リソースグループと同じノードで起動するようにします。

```
[root@z1 ~]# pcs constraint colocation add shared_vg1-clone with locking-clone
[root@z1 ~]# pcs constraint colocation add shared_vg2-clone with locking-clone
```

16. クラスターの両ノードで、論理ボリュームがアクティブであることを確認します。数秒の遅延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
```

17. 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の一部として指定できます。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
```

検証手順

1. 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)
```

2. 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
[...]

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 z1.example.com
     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 ]

...
```

関連情報

- GFS2 ファイルシステムの設定
- Microsoft Azure での Red Hat High Availability クラスターの設定
- AWS での Red Hat High Availability クラスターの設定
- Google Cloud Platform での Red Hat High Availability クラスターの設定

7.2. クラスターでの暗号化 GFS2 ファイルシステムの設定

次の手順で、LUKS で暗号化した GFS2 ファイルシステムを含む Pacemaker クラスターを作成できます。この例では、論理ボリュームに 1 つの GFS2 ファイルシステムを作成し、そのファイルシステムを暗号化します。暗号化された GFS2 ファイルシステムは、LUKS 暗号化に対応する **crypt** リソースエージェントを使用してサポートされます。

この手順は、以下の 3 つの部分で設定されます。

- Pacemaker クラスター内で共有論理ボリュームを設定する
- 論理ボリュームを暗号化して **crypt** リソースを作成する
- GFS2 ファイルシステムで暗号化された論理ボリュームをフォーマットしてクラスター用のファイルシステムリソースを作成する

7.2.1. Pacemaker クラスター内での共有論理ボリュームの設定

前提条件

- 2 つのクラスターノードにクラスターソフトウェアをインストールして起動し、基本的な 2 ノードクラスターを作成している。
- クラスターのフェンシングを設定している。

Pacemaker クラスターの作成とクラスターのフェンシングの設定については、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。

手順

1. クラスター内の両方のノードで、システムアーキテクチャーに対応する Resilient Storage のリポジトリーを有効にします。たとえば、x86_64 システムの Resilient Storage リポジトリーを有効にするには、以下の **subscription-manager** コマンドを入力します。

   ```
   # subscription-manager repos --enable=rhel-9-for-x86_64-resilientstorage-rpms
   ```

   Resilient Storage リポジトリーは、High Availability リポジトリーのスーパーセットであることに注意してください。Resilient Storage リポジトリーを有効にする場合は、High Availability リポジトリーを有効にする必要はありません。

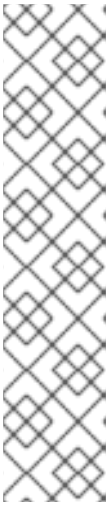
2. クラスターの両方のノードで、**lvm2-lockd** パッケージ、**gfs2-utils** パッケージ、および **dlm** パッケージをインストールします。AppStream チャンネルおよび Resilient Storage チャンネルにサブスクライブして、これらのパッケージをサポートする必要があります。

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

3. クラスターの両方のノードで、**/etc/lvm/lvm.conf** ファイルの **use_lvmlockd** 設定オプションを **use_lvmlockd=1** に設定します。

   ```
   ...
   use_lvmlockd = 1
   ...
   ```

4. グローバル Pacemaker パラメーター **no-quorum-policy** を **freeze** に設定します。



注記

デフォルトでは、**no-quorum-policy** の値は **stop** に設定され、定足数が失われると、残りのパーティションのリソースがすべて即座に停止されます。通常、このデフォルト設定は最も安全なオプションで最適なおプションですが、ほとんどのリソースとは異なり、GFS2 が機能するにはクォーラムが必要です。クォーラムが失われると、GFS2 マウントを使用したアプリケーション、GFS2 マウント自体の両方が正しく停止できません。クォーラムなしでこれらのリソースを停止しようとすると失敗し、最終的にクォーラムが失われるたびにクラスター全体がフェンスされます。

この状況に対処するには、GFS2 の使用時の **no-quorum-policy** を **freeze** に設定します。この設定では、クォーラムが失われると、クォーラムが回復するまで残りのパーティションは何もしません。

```
[root@z1 ~]# pcs property set no-quorum-policy=freeze
```

5. **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、リソースグループ **locking** に追加します。

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

6. 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
```

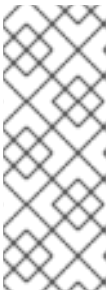
7. **lvmlockd** リソースを、**locking** グループに追加します。

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

8. 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 ]
```

9. クラスターの 1 つのノードで、共有ボリュームグループを作成します。



注記

LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Red Hat は、Pacemaker が起動する前にサービスが開始されるように設定することを推奨します。Pacemaker クラスターによって使用されるリモート物理ボリュームの起動順序の設定については、Pacemaker で管理されないリソース依存関係の起動順序の設定 を参照してください。

以下のコマンドは、共有ボリュームグループ **shared_vg1** を **/dev/sda1** に作成します。

```
[root@z1 ~]# vgcreate --shared shared_vg1 /dev/sda1
  Physical volume "/dev/sda1" successfully created.
  Volume group "shared_vg1" successfully created
  VG shared_vg1 starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

10. クラスター内の 2 番目のノードで以下を実行します。

    a. **lvm.conf** ファイルの **use_devicesfile = 1** パラメーターでデバイスファイルの使用が有効になっている場合は、クラスター内の 2 番目のノードのデバイスファイルに共有デバイスを追加します。この機能はデフォルトで有効化されています。

    ```
    [root@z2 ~]# lvmdevices --adddev /dev/sda1
    ```

    b. 共有ボリュームグループのロックマネージャーを起動します。

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

11. 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ます。

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

12. 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、論理ボリュームに **LVM が有効** なリソースを作成します。
以下のコマンドは、ボリュームグループ **shared_vg1** の論理グループ **shared_lv1** に、名前が **sharedlv1** で、**LVM が有効な** リソースを作成します。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
```

13. 新しいリソースグループのクローンを作成します。

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

14. **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)
```

15. コロケーション制約を設定して、**vg1** および **vg2** のリソースグループが **locking** リソースグループと同じノードで起動するようにします。

```
[root@z1 ~]# pcs constraint colocation add shared_vg1-clone with locking-clone
```

検証手順

クラスターの両ノードで、論理ボリュームがアクティブであることを確認します。数秒の遅延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

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

7.2.2. 論理ボリュームの暗号化および暗号化リソースの作成

前提条件

- Pacemaker クラスターに共有論理ボリュームを設定している。

手順

1. クラスター内の 1 つのノードで、crypt キーを含めて新しいファイルを作成し、ファイルにパーミッションを設定して root でのみ読み取りできるようにします。

```
[root@z1 ~]# touch /etc/crypt_keyfile
[root@z1 ~]# chmod 600 /etc/crypt_keyfile
```

2. crypt キーを作成します。

```
[root@z1 ~]# dd if=/dev/urandom bs=4K count=1 of=/etc/crypt_keyfile
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000306202 s, 13.4 MB/s
[root@z1 ~]# scp /etc/crypt_keyfile root@z2.example.com:/etc/
```

3. **-p** パラメーターを使用して設定したパーミッションを保持した状態で、cript キーファイルをクラスター内の他のノードに配布します。

```
[root@z1 ~]# scp -p /etc/crypt_keyfile root@z2.example.com:/etc/
```

4. LVM ボリュームに暗号化デバイスを作成して、暗号化された GFS2 ファイルシステムを設定します。

```
[root@z1 ~]# cryptsetup luksFormat /dev/shared_vg1/shared_lv1 --type luks2 --key-file=/etc/crypt_keyfile
WARNING!
========
This will overwrite data on /dev/shared_vg1/shared_lv1 irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
```

5. **shared_vg1** ボリュームグループの一部として crypt リソースを作成します。

```
[root@z1 ~]# pcs resource create crypt --group shared_vg1 ocf:heartbeat:crypt crypt_dev="luks_lv1" crypt_type=luks2 key_file=/etc/crypt_keyfile encrypted_dev="/dev/shared_vg1/shared_lv1"
```

検証手順

crypt リソースが crypt デバイスを作成していることを確認します。この例では crypt デバイスは **/dev/mapper/luks_lv1** です。

```
[root@z1 ~]# ls -l /dev/mapper/
...
lrwxrwxrwx 1 root root 7 Mar 4 09:52 luks_lv1 -> ../dm-3
...
```

7.2.3. GFS2 ファイルシステムで暗号化された論理ボリュームをフォーマットしてクラスター用のファイルシステムリソースを作成します。

前提条件

- 論理ボリュームを暗号化し、crypt リソースを作成している。

手順

1. クラスター内の 1 つのノードで、GFS2 ファイルシステムを使用してボリュームをフォーマットします。ファイルシステムをマウントするノードごとに、ジャーナルが 1 つ必要になります。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。ロックテーブル名の形式は、**ClusterName:FSName** です。**ClusterName** は、GFS2 ファイルシステムが作成されているクラスターの名前です。**FSname** はファイルシステム名です。これは、クラスター経由のすべての **lock_dlm** ファイルシステムで一意である必要があります。

```
[root@z1 ~]# mkfs.gfs2 -j3 -p lock_dlm -t my_cluster:gfs2-demo1 /dev/mapper/luks_lv1
/dev/mapper/luks_lv1 is a symbolic link to /dev/dm-3
This will destroy any data on /dev/dm-3
Are you sure you want to proceed? [y/n] y
Discarding device contents (may take a while on large devices): Done
Adding journals: Done
Building resource groups: Done
Creating quota file: Done
Writing superblock and syncing: Done
Device:                    /dev/mapper/luks_lv1
Block size:                4096
```

```
Device size:               4.98 GB (1306624 blocks)
Filesystem size:           4.98 GB (1306622 blocks)
Journals:                  3
Journal size:              16MB
Resource groups:           23
Locking protocol:          "lock_dlm"
Lock table:                "my_cluster:gfs2-demo1"
UUID:                      de263f7b-0f12-4d02-bbb2-56642fade293
```

2. ファイルシステムリソースを作成し、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。
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の一部として指定できます。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。このコマンドは、対象のファイルシステムの論理ボリュームリソースを含むリソースグループに、リソースを追加します。

```
[root@z1 ~]# pcs resource create sharedfs1 --group shared_vg1
ocf:heartbeat:Filesystem device="/dev/mapper/luks_lv1" directory="/mnt/gfs1"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence
```

検証手順

1. 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/luks_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)

[root@z2 ~]# mount | grep gfs2
/dev/mapper/luks_lv1 on /mnt/gfs1 type gfs2 (rw,noatime,seclabel)
```

2. 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
[...]

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
```

```
        crypt       (ocf::heartbeat:crypt) Started z2.example.com
        sharedfs1       (ocf::heartbeat:Filesystem):    Started z2.example.com
    Resource Group: shared_vg1:1
        sharedlv1       (ocf::heartbeat:LVM-activate):  Started z1.example.com
        crypt       (ocf::heartbeat:crypt)  Started z1.example.com
        sharedfs1       (ocf::heartbeat:Filesystem):    Started z1.example.com
      Started:  [z1.example.com z2.example.com ]
...
```

関連情報

- GFS2 ファイルシステムの設定

第8章 RED HAT HIGH AVAILABILITY クラスターでのアクティブ/アクティブ SAMBA サーバーの設定

Red Hat High Availability Add-On は、アクティブ/アクティブクラスター設定で Samba を設定するためのサポートを提供します。次の例では、2 ノードの RHEL クラスターでアクティブ/アクティブ Samba サーバーを設定します。

Samba のサポートポリシーについては、Red Hat Customer Portal の RHEL High Availability のサポートポリシー - ctdb 一般ポリシー および RHEL 復元ストレージのサポートポリシー - 他のプロトコルを介した gfs2 コンテンツのエクスポート を参照してください。

アクティブ/アクティブクラスターで Samba を設定するには:

1. GFS2 ファイルシステムとそれに関連するクラスターリソースを設定します。
2. クラスターノードで Samba を設定します。
3. Samba クラスターリソースを設定します。
4. 設定した Samba サーバーをテストします。

8.1. 高可用性クラスターでの SAMBA サービス用の GFS2 ファイルシステムの設定

Pacemaker クラスターでアクティブ/アクティブ Samba サービスを設定する前に、クラスターの GFS2 ファイルシステムを設定します。

前提条件

- ノードごとにフェンシングが設定された 2 ノードの Red Hat High Availability クラスター
- 各クラスターノードで利用可能な共有ストレージ
- 各クラスターノードの AppStream チャネルと Resilient Storage チャネルへのサブスクリプション

Pacemaker クラスターの作成とクラスターのフェンシングの設定については、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。

手順

1. クラスター内の両方のノードで、次の初期設定手順を実行します。
   a. システムアーキテクチャーに対応する Resilient Storage のリポジトリーを有効にします。たとえば、x86_64 システムの Resilient Storage リポジトリーを有効にするには、次の **subscription-manager** コマンドを入力します。

   ```
   # subscription-manager repos --enable=rhel-9-for-x86_64-resilientstorage-rpms
   ```

   Resilient Storage リポジトリーは、High Availability リポジトリーのスーパーセットです。Resilient Storage リポジトリーを有効にする場合は、High Availability リポジトリーを有効にする必要はありません。

   b. **lvm2-lockd**、**gfs2-utils**、および **dlm** パッケージをインストールします。

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

c. **/etc/lvm/lvm.conf** ファイルの **use_lvmlockd** 設定オプションを **use_lvmlockd=1** に設定します。

```
...

use_lvmlockd = 1

...
```

2. クラスター内の1つのノードで、Pacemaker のグローバルパラメーター **no-quorum-policy** を **freeze** に設定します。

注記

デフォルトでは、**no-quorum-policy** の値は **stop** に設定され、定足数が失われると、残りのパーティションのリソースがすべて即座に停止されます。通常、このデフォルト設定は最も安全なオプションで最適なおプションですが、ほとんどのリソースとは異なり、GFS2 が機能するにはクォーラムが必要です。クォーラムが失われると、GFS2 マウントを使用したアプリケーション、GFS2 マウント自体の両方が正しく停止できません。クォーラムなしでこれらのリソースを停止しようとすると失敗し、最終的にクォーラムが失われるたびにクラスター全体がフェンスされます。

この状況に対処するには、GFS2 の使用時の **no-quorum-policy** を **freeze** に設定します。この設定では、クォーラムが失われると、クォーラムが回復するまで残りのパーティションは何もしません。

```
[root@z1 ~]# pcs property set no-quorum-policy=freeze
```

3. **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、リソースグループ **locking** に追加します。以前にクラスターのフェンシングを設定していない場合、この手順は失敗し、**pcs status** コマンドはリソース障害メッセージを表示します。

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

4. 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
```

5. **locking** リソースグループの一部として **lvmlockd** リソースを設定します。

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

6. 共有デバイス **/dev/vdb** に物理ボリュームと共有ボリュームグループを作成します。この例では、共有ボリュームグループ **csmb_vg** を作成します。

```
[root@z1 ~]# pvcreate /dev/vdb
```

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

7. クラスター内の 2 番目のノードで以下を実行します。

8. **lvm.conf** ファイルの **use_devicesfile = 1** パラメーターでデバイスファイルの使用が有効になっている場合は、クラスター内の 2 番目のノードのデバイスファイルに共有デバイスを追加します。この機能はデフォルトで有効化されています。

```
[root@z2 ~]# lvmdevices --adddev /dev/vdb
```

   a. 共有ボリュームグループのロックマネージャーを起動します。

```
[root@z2 ~]# vgchange --lockstart csmb_vg
  VG csmb_vg starting dlm lockspace
  Starting locking.  Waiting until locks are ready...
```

9. クラスター内の 1 つのノードで論理ボリュームを作成し、CTDB が内部ロックのために排他的に使用する GFS2 ファイルシステムでボリュームをフォーマットします。デプロイメントで複数の共有をエクスポートする場合でも、クラスター内に必要なファイルシステムは 1 つだけです。
mkfs.gfs2 コマンドの **-t** オプションでロックテーブル名を指定する場合は、指定する **clustername:filesystemname** の最初の要素がクラスターの名前と一致していることを確認してください。この例では、クラスター名は **my_cluster** です。

```
[root@z1 ~]# lvcreate -L1G -n ctdb_lv csmb_vg
[root@z1 ~]# mkfs.gfs2 -j3 -p lock_dlm -t my_cluster:ctdb /dev/csmb_vg/ctdb_lv
```

10. Samba で共有される GFS2 ファイルシステムごとに論理ボリュームを作成し、そのボリュームを GFS2 ファイルシステムでフォーマットします。この例では、単一の GFS2 ファイルシステムと Samba 共有を作成しますが、複数のファイルシステムと共有を作成できます。

```
[root@z1 ~]# lvcreate -L50G -n csmb_lv1 csmb_vg
[root@z1 ~]# mkfs.gfs2 -j3 -p lock_dlm -t my_cluster:csmb1 /dev/csmb_vg/csmb_lv1
```

11. **LVM_Activate** リソースをセットアップして、必要な共有ボリュームがアクティブ化されるようにします。この例では、**LVM_Activate** リソースをリソースグループ **shared_vg** の一部として作成し、そのリソースグループのクローンを作成して、クラスター内のすべてのノードで実行されるようにします。
必要な順序の制約を設定する前にリソースが自動的に開始されないように、リソースを無効にして作成します。

```
[root@z1 ~]# pcs resource create --disabled --group shared_vg ctdb_lv
ocf:heartbeat:LVM-activate lvname=ctdb_lv vgname=csmb_vg
activation_mode=shared vg_access_mode=lvmlockd
[root@z1 ~]# pcs resource create --disabled --group shared_vg csmb_lv1
ocf:heartbeat:LVM-activate lvname=csmb_lv1 vgname=csmb_vg
activation_mode=shared vg_access_mode=lvmlockd
[root@z1 ~]# pcs resource clone shared_vg interleave=true
```

12. **shared_vg** リソースグループのメンバーの前に、**locking** リソースグループのすべてのメンバーを開始するように、順序制約を設定します。

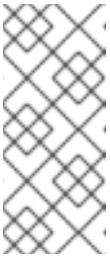
```
[root@z1 ~]# pcs constraint order start locking-clone then shared_vg-clone
Adding locking-clone shared_vg-clone (kind: Mandatory) (Options: first-action=start then-action=start)
```

13. **LVM-activate** リソースを有効にします。

```
[root@z1 ~]# pcs resource enable ctdb_lv csmb_lv1
```

14. クラスター内の 1 つのノードで、次の手順を実行して、必要な **Filesystem** リソースを作成します。

    a. 以前に LVM ボリュームに設定した GFS2 ファイルシステムを使用して、クローンリソースとして **Filesystem** リソースを作成します。これにより、Pacemaker がファイルシステムをマウントおよび管理するように設定されます。



注記

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。**options=options** を使用して、リソース設定の一部としてマウントオプションを指定できます。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。

```
[root@z1 ~]# pcs resource create ctdb_fs Filesystem device="/dev/csmb_vg/ctdb_lv" directory="/mnt/ctdb" fstype="gfs2" op monitor interval=10s on-fail=fence clone interleave=true
[root@z1 ~]# pcs resource create csmb_fs1 Filesystem device="/dev/csmb_vg/csmb_lv1" directory="/srv/samba/share1" fstype="gfs2" op monitor interval=10s on-fail=fence clone interleave=true
```

    b. 共有ボリュームグループ **shared_vg** の起動後に Pacemaker がファイルシステムをマウントするように、順序制約を設定します。

```
[root@z1 ~]# pcs constraint order start shared_vg-clone then ctdb_fs-clone
Adding shared_vg-clone ctdb_fs-clone (kind: Mandatory) (Options: first-action=start then-action=start)
[root@z1 ~]# pcs constraint order start shared_vg-clone then csmb_fs1-clone
Adding shared_vg-clone csmb_fs1-clone (kind: Mandatory) (Options: first-action=start then-action=start)
```

8.2. 高可用性クラスターでの SAMBA の設定

Pacemaker クラスターで Samba サービスを設定するには、クラスター内のすべてのノードでサービスを設定します。

前提条件

- 高可用性クラスターでの Samba サービス用の GFS2 ファイルシステムの設定 で説明したように、GFS2 ファイルシステムで設定された 2 ノードの Red Hat High Availability クラスター。
- Samba 共有に使用するために GFS2 ファイルシステム上に作成されたパブリックディレクトリー。この例では、ディレクトリーは **/srv/samba/share1** です。

- このクラスターによってエクスポートされた Samba 共有へのアクセスに使用できるパブリック仮想 IP アドレス。

手順

1. クラスター内の両方のノードで、Samba サービスを設定し、共有定義をセットアップします。

   a. Samba および CTDB パッケージをインストールします。

   ```
   # dnf -y install samba ctdb cifs-utils samba-winbind
   ```

   b. **ctdb**、**smb**、**nmb**、および **winbind** サービスが実行されておらず、起動時に開始されていないことを確認してください。

   ```
   # systemctl disable --now ctdb smb nmb winbind
   ```

   c. **/etc/samba/smb.conf** ファイルで、Samba サービスを設定し、1 つの共有を持つスタンドアロンサーバーの次の例のように、共有定義をセットアップします。

   ```
   [global]
      netbios name = linuxserver
      workgroup = WORKGROUP
      security = user
      clustering = yes
   [share1]
      path = /srv/samba/share1
      read only = no
   ```

   d. **/etc/samba/smb.conf** ファイルを検証します。

   ```
   # testparm
   ```

2. クラスター内の両方のノードで、CTDB を設定します。

   a. **/etc/ctdb/nodes** ファイルを作成し、このノードファイルの例のように、クラスターノードの IP アドレスを追加します。

   ```
   192.0.2.11
   192.0.2.12
   ```

   b. **/etc/ctdb/public_addresses** ファイルを作成し、クラスターのパブリックインターフェイスの IP アドレスとネットワークデバイス名をファイルに追加します。**public_addresses** ファイルで IP アドレスを割り当てる場合は、これらのアドレスが使用されていないこと、および目的のクライアントからルーティング可能であることを確認してください。**/etc/ctdb/public_addresses** ファイルの各エントリーの 2 番目のフィールドは、対応するパブリックアドレスのクラスターマシンで使用するインターフェイスです。この例の **public_addresses** ファイルでは、インターフェイス **enp1s0** がすべてのパブリックアドレスに使用されます。

   ```
   192.0.2.201/24 enp1s0
   192.0.2.202/24 enp1s0
   ```

   クラスターのパブリックインターフェイスは、クライアントがネットワークから Samba にアクセスするために使用するインターフェイスです。負荷分散のために、クラスターの各

パブリック IP アドレスの A レコードを DNS ゾーンに追加します。これらの各レコードは、同じホスト名に解決される必要があります。クライアントはホスト名を使用して Samba にアクセスし、DNS はクライアントをクラスターのさまざまなノードに分散します。

c. **firewalld** サービスを実行している場合は、**ctdb** および **samba** サービスに必要なポートを有効にします。

```
# firewall-cmd --add-service=ctdb --add-service=samba --permanent
# firewall-cmd --reload
```

3. クラスター内の 1 つのノードで、SELinux コンテキストを更新します。

a. GFS2 共有上の SELinux コンテキストを更新します。

```
[root@z1 ~]# semanage fcontext -at ctdbd_var_run_t -s system_u "/mnt/ctdb(/.)?"
[root@z1 ~]# restorecon -Rv /mnt/ctdb
```

b. Samba で共有されているディレクトリーの SELinux コンテキストを更新します。

```
[root@z1 ~]# semanage fcontext -at samba_share_t -s system_u
"/srv/samba/share1(/.)?"
[root@z1 ~]# restorecon -Rv /srv/samba/share1
```

関連情報

- この例のように、Samba をスタンドアロンサーバーとして設定する方法の詳細については、ネットワークファイルサービスの設定と使用 の章 Samba をサーバーとして使用する を参照してください。
- BIND プライマリーサーバーでの正引きゾーンの設定。

8.3. SAMBA クラスターリソースの設定

2 ノードの高可用性クラスターの両方のノードで Samba サービスを設定したら、クラスターの Samba クラスターリソースを設定します。

前提条件

- 高可用性クラスターでの Samba サービス用の GFS2 ファイルシステムの設定 で説明したように、GFS2 ファイルシステムで設定された 2 ノードの Red Hat High Availability クラスター。
- 高可用性クラスターでの Samba の設定 で説明したように、両方のクラスターノードで設定された Samba サービス。

手順

1. クラスター内の 1 つのノードで、Samba クラスターリソースを設定します。

a. グループ **samba-group** に CTDB リソースを作成します。CTDB リソースエージェントは、**pcs** コマンドで指定された **ctdb_*** オプションを使用して、CTDB 設定ファイルを作成します。必要な順序の制約を設定する前にリソースが自動的に開始されないように、リソースを無効にして作成します。

```
[root@z1 ~]# pcs resource create --disabled ctdb --group samba-group
ocf:heartbeat:CTDB ctdb_recovery_lock=/mnt/ctdb/ctdb.lock
ctdb_dbdir=/var/lib/ctdb ctdb_logfile=/var/log/ctdb.log op monitor interval=10
timeout=30 op start timeout=90 op stop timeout=100
```

b. **samba-group** リソースグループを複製します。

```
[root@z1 ~]# pcs resource clone samba-group
```

c. すべての **Filesystem** リソースが **samba-group** 内のリソースの前に実行されるように、順序制約を作成します。

```
[root@z1 ~]# pcs constraint order start ctdb_fs-clone then samba-group-clone
[root@z1 ~]# pcs constraint order start csmb_fs1-clone then samba-group-clone
```

d. リソースグループ **samba-group** に **samba** リソースを作成します。これにより、追加された順序に基づいて、CTDB と Samba の間に暗黙的な順序制約が作成されます。

```
[root@z1 ~]# pcs resource create samba --group samba-group systemd:smb
```

e. **ctdb** および **samba** リソースを有効にします。

```
[root@z1 ~]# pcs resource enable ctdb samba
```

f. すべてのサービスが正常に開始されたことを確認します。

注記

CTDB が Samba を起動し、共有をエクスポートして安定するまでに数分かかる場合があります。このプロセスが完了する前にクラスターのステータスを確認すると、**samba** サービスがまだ実行されていないことがわかる場合があります。

```
[root@z1 ~]# pcs status

...

Full List of Resources:
  * fence-z1    (stonith:fence_xvm): Started z1.example.com
  * fence-z2    (stonith:fence_xvm): Started z2.example.com
  * Clone Set: locking-clone [locking]:
 * Started: [ z1.example.com z2.example.com ]
  * Clone Set: shared_vg-clone [shared_vg]:
 * Started: [ z1.example.com z2.example.com ]
  * Clone Set: ctdb_fs-clone [ctdb_fs]:
 * Started: [ z1.example.com z2.example.com ]
  * Clone Set: csmb_fs1-clone [csmb_fs1]:
 * Started: [ z1.example.com z2.example.com ]
   * Clone Set: samba-group-clone [samba-group]:
 * Started: [ z1.example.com z2.example.com ]
```

2. クラスター内の両方のノードで、テスト共有ディレクトリーのローカルユーザーを追加します。

   a. ユーザーを追加します。

   ```
   # useradd -M -s /sbin/nologin example_user
   ```

   b. ユーザーのパスワードを設定します。

   ```
   # passwd example_user
   ```

   c. ユーザーの SMB パスワードを設定します。

   ```
   # smbpasswd -a example_user
   New SMB password:
   Retype new SMB password:
   Added user example_user
   ```

   d. Samba データベースでユーザーをアクティブ化します。

   ```
   # smbpasswd -e example_user
   ```

   e. Samba ユーザーの GFS2 共有に対するファイルの所有権と権限を更新します。

   ```
   # chown example_user:users /srv/samba/share1/
   # chmod 755 /srv/samba/share1/
   ```

8.4. クラスター化された SAMBA 設定の確認

クラスター化された Samba 設定が成功した場合は、Samba 共有をマウントできます。共有をマウントした後、Samba 共有をエクスポートしているクラスターノードが使用できなくなった場合、Samba の復旧をテストできます。

手順

1. クラスターノードの **/etc/ctdb/public_addresses** ファイルで設定された 1 つ以上のパブリック IP アドレスにアクセスできるシステムで、これらのパブリック IP アドレスのいずれかを使用して Samba 共有をマウントします。

   ```
   [root@testmount ~]# mkdir /mnt/sambashare
   [root@testmount ~]# mount -t cifs -o user=example_user //192.0.2.201/share1
   /mnt/sambashare
   Password for example_user@//192.0.2.201/public: XXXXXXX
   ```

2. ファイルシステムがマウントされていることを確認します。

   ```
   [root@testmount ~]# mount | grep /mnt/sambashare
   //192.0.2.201/public on /mnt/sambashare type cifs
   (rw,relatime,vers=1.0,cache=strict,username=example_user,domain=LINUXSERVER,uid=0,nof
   orceuid,gid=0,noforcegid,addr=192.0.2.201,unix,posixpaths,serverino,mapposix,acl,rsize=1048
   76,wsize=65536,echo_interval=60,actimeo=1,user=example_user)
   ```

3. マウントされたファイルシステムにファイルを作成できることを確認します。

```
[root@testmount ~]# touch /mnt/sambashare/testfile1
[root@testmount ~]# ls /mnt/sambashare
testfile1
```

4. Samba 共有をエクスポートしているクラスターノードを特定します。

   a. 各クラスターノードで、**public_addresses** ファイルで指定されたインターフェイスに割り当てられた IP アドレスを表示します。次のコマンドは、各ノードの **enp1s0** インターフェイスに割り当てられた IPv4 アドレスを表示します。

   ```
   [root@z1 ~]# ip -4 addr show enp1s0 | grep inet
       inet 192.0.2.11/24 brd 192.0.2.255 scope global dynamic noprefixroute enp1s0
       inet 192.0.2.201/24 brd 192.0.2.255 scope global secondary enp1s0

   [root@z2 ~]# ip -4 addr show enp1s0 | grep inet
       inet 192.0.2.12/24 brd 192.0.2.255 scope global dynamic noprefixroute enp1s0
       inet 192.0.2.202/24 brd 192.0.2.255 scope global secondary enp1s0
   ```

   b. **ip** コマンドの出力で、共有をマウントしたときに **mount** コマンドで指定した IP アドレスを持つノードを見つけます。
   この例では、mount コマンドで指定された IP アドレスは 192.0.2.201 です。**ip** コマンドの出力は、IP アドレス 192.0.2.201 が **z1.example.com** に割り当てられていることを示しています。

5. Samba 共有をエクスポートするノードを **standby** モードにします。これにより、ノードはクラスターリソースをホストできなくなります。

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

6. ファイルシステムをマウントしたシステムから、ファイルシステム上にファイルを作成できることを確認します。

```
[root@testmount ~]# touch /mnt/sambashare/testfile2
[root@testmount ~]# ls /mnt/sambashare
testfile1  testfile2
```

7. 作成したファイルを削除して、ファイルシステムが正常にマウントされたことを確認します。ファイルシステムをマウントする必要がなくなった場合は、この時点でアンマウントします。

```
[root@testmount ~]# rm /mnt/sambashare/testfile1 /mnt/sambashare/testfile2
rm: remove regular empty file '/mnt/sambashare/testfile1'? y
rm: remove regular empty file '/mnt/sambashare/testfile1'? y
[root@testmount ~]# umount /mnt/sambashare
```

8. クラスターノードの 1 つから、以前にスタンバイモードにしたノードにクラスターサービスを復元します。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。

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

第9章 PCSD WEB UI の使用

pcsd Web UI は、Pacemaker クラスターおよび 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イスです。

9.1. PCSD WEB UI の設定

次の手順で、**pcsd** Web UI を使用してクラスターを設定するようにシステムをセットアップします。

前提条件

- Pacemaker 設定ツールがインストールされている。
- お使いのシステムがクラスター設定用にセットアップされている。

クラスターソフトウェアをインストールし、クラスター設定用にシステムをセットアップする手順については、クラスターソフトウェアのインストール を参照してください。

手順

1. いずれかのシステムで以下の URL をブラウザーで開き、クラスターのノードのいずれかを指定します (**https** プロトコルを使用することに注意してください)。これにより、**pcsd** Web UI のログイン画面が表示されます。

   ```
   https://nodename:2224
   ```

2. ユーザー **hacluster** としてログインします。これにより、**Clusters** ページが表示されます。

9.2. 高可用性の PCSD WEB UI の設定

pcsd Web UI を使用すると、クラスターのノードのいずれかに接続して、クラスター管理ページを表示できます。接続先のノードがダウンするか、使用できなくなった場合は、クラスターの別のノードを指定する URL でブラウザーを開くと、クラスターに再接続できます。ただし、高可用性に **pcsd** Web UI 自体を設定することもできます。この場合は、新しい URL を入力しなくても、引き続きクラスターを管理できます。

手順

高可用性に **pcsd** Web UI を設定するには、以下の手順を実行します。

1. **/etc/sysconfig/pcsd** 設定ファイルで **PCSD_SSL_CERT_SYNC_ENABLED** を **true** に設定して、**pcsd** 証明書がクラスターのノード間で同期されるようにします。証明書の同期を有効にすると、**pcsd** がクラスター設定およびノードの追加コマンドの証明書を同期します。**PCSD_SSL_CERT_SYNC_ENABLED** はデフォルトで **false** に設定されます。

2. **pcsd** Web UI への接続に使用する Floating IP アドレスである **IPaddr2** クラスターリソースを作成します。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 デバイスを指定していない場合は、そのノードに静的に割り当てられている IP アドレスの 1 つと同じネットワークに Floating IP が存在していないと、Floating 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。

3. **pcsd** を使用するためにカスタムの SSL 証明書を作成し、**pcsd** Web UI への接続に使用するノードのアドレスに対して有効であることを確認します。

   a. カスタムの SSL 証明書を作成するには、ワイルドカード証明書を使用するか、SAN

(Subject Alternative Name: サブジェクトの別名) 証明書の延長を使用できます。Red Hat Certificate System の詳細は、Red Hat Certificate System 管理ガイド を参照してください。

b. **pcs pcsd certkey** コマンドを使用して **pcsd** のカスタム証明書をインストールします。

c. **pcs pcsd sync-certificates** コマンドを使用して、**pcsd** 証明書を、クラスター内のすべてのノードに同期させます。

4. クラスターリソースとして設定した Floating IP アドレスを使用して、**pcsd** Web UI に接続します。

注記

高可用性に **pcsd** Web UI を設定している場合でも、ユーザーが接続しているノードがダウンすると、再びログインするように求められます。

第10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場合は、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、そのノードが完全にオフラインであることを確認できます。

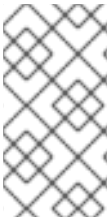
STONITH はクラスター化したサービスを停止できない場合にも役に立ちます。この場合は、クラスターが STONITH を使用してノード全体を強制的にオフラインにし、その後サービスを別の場所で開始すると安全です。

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 Fencing in a Red Hat High Availability Cluster 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。

10.1. 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、利用可能なフェンスエージェントと、特定のフェンスエージェントで利用可能なオプションを表示できます。



注記

システムのハードウェアによって、クラスターに使用するフェンシングデバイスのタイプが決まります。サポートされているプラットフォームとアーキテクチャー、およびさまざまなフェンシングデバイスについては、記事 Cluster Platforms and Architectures の Support Policies for RHEL High Availability Clusters セクションを参照してください。

使用可能なすべてのフェンスエージェントのリストを表示するには、次のコマンドを実行します。フィルターを指定すると、フィルターに一致するフェンスエージェントのみが表示されます。

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

指定したフェンスエージェントのオプションを表示するには、次のコマンドを実行します。

```
pcs stonith describe [stonith_agent]
```

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

```
# pcs stonith describe fence_apc
Stonith options for: fence_apc
  ipaddr (required): IP Address or Hostname
  login (required): Login Name
  passwd: Login password or passphrase
  passwd_script: Script to retrieve password
  cmd_prompt: Force command prompt
  secure: SSH connection
  port (required): Physical plug number or name of virtual machine
  identity_file: Identity file for ssh
```

```
switch: Physical switch number on device
inet4_only: Forces agent to use IPv4 addresses only
inet6_only: Forces agent to use IPv6 addresses only
ipport: TCP port to use for connection with device
action (required): Fencing Action
verbose: Verbose mode
debug: Write debug information to given file
version: Display version information and exit
help: Display help and exit
separator: Separator for CSV created by operation list
power_timeout: Test X seconds for status change after ON/OFF
shell_timeout: Wait X seconds for cmd prompt after issuing command
login_timeout: Wait X seconds for cmd prompt after login
power_wait: Wait X seconds after issuing ON/OFF
delay: Wait X seconds before fencing is started
retry_on: Count of attempts to retry power on
```

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、**fence_sbd** エージェントを除き、**cycle** の値はサポートされていないため、データの破損を引き起こす可能性があるため、この値は指定しないでください。ただし、**fence_sbd** であっても、メソッドを指定せず、代わりにデフォルト値を使用してください。

10.2. フェンスデバイスの作成

フェンスデバイスを作成するコマンドの形式は、以下のとおりです。利用可能なフェンスデバイス作成オプションのリストは、**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]
```

以下のコマンドは、1つのノードに対して、1つのフェンシングデバイスを作成します。

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

1つのノードのみをフェンスできるフェンスデバイスや、複数のノードをフェンスできるデバイスもあります。フェンシングデバイスの作成時に指定するパラメーターは、フェンシングデバイスが対応しているか、必要としているかにより異なります。

- フェンスデバイスの中には、フェンスできるノードを自動的に判断できるものがあります。
- フェンシングデバイスの作成時に **pcmk_host_list** パラメーターを使用すると、フェンシングデバイスで制御されるすべてのマシンを指定できます。
- フェンスデバイスによっては、フェンスデバイスが理解する仕様へのホスト名のマッピングが必要となるものがあります。フェンシングデバイスの作成時に、**pcmk_host_map** パラメーターを使用して、ホスト名をマッピングできます。

pcmk_host_list パラメーターおよび **pcmk_host_map** パラメーターの詳細は、フェンシングデバイスの一般的なプロパティー を参照してください。

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。フェンスデバイスをテストする方法は フェンスデバイスのテスト を参照してください。

10.3. フェンシングデバイスの一般的なプロパティー

フェンシングデバイスにも設定可能な一般的なプロパティーや、フェンスの動作を決定するさまざまなクラスタープロパティーがあります。

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。

- フェンシングデバイスは、**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シング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場合は、Red Hat ではクラスターがサポートされないことに注意してください。

以下の表は、フェンシング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10.1 フェンシング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		ホスト名を、ホスト名に対応していないデバイスのポート番号へマッピングします。例: **node1:1;node2:2,3** は、ノード 1 にポート 1 を使用し、ノード 2 にポート 2 と 3 を使用するようにクラスターに指示します。**pcmk_host_map** プロパティーは、値の前にバックスラッシュを使用して **pcmk_host_map** 値内の特殊文字をサポートします。たとえば、**pcmk_host_map="node3:plug\ 1"** を指定して、ホストエイリアスにスペースを含めることができます。
pcmk_host_list	文字列		このデバイスで制御するマシンのリストです (**pcmk_host_check=static-list** 以外は任意)。

フィールド	タイプ	デフォルト	説明
pcmk_host_check	文字列	* **pcmk_host_list** または **pcmk_host_map** が設定されている場合は **static-list** * それが設定されておらず、フェンスデバイスが **list** アクションに対応する場合は **dynamic-list** になります。 * それ以外で、フェンスデバイスが **status** アクションに対応している場合は **status** になります。 * それ以外は、**none** になります。	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** (デバイスへの問い合わせ)、**static-list** (**pcmk_host_list** 属性の確認)、なし (すべてのデバイスで全マシンのフェンスが可能と見なされる) です。

以下の表では、フェンシングデバイスに設定できるその他のプロパティーをまとめています。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10.2 フェンシング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
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。このパラメーターを使用して、デバイス固有の代替パラメーターを指定します。これは、フェンシングするマシンを示します。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場合は、**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。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。この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代替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_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。再起動が完了するまでに通常より長い時間を要するデバイスもあれば、通常より短い時間で完了するデバイスもあります。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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が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。Pacemaker による再起動の動作の再試行回数を変更する場合に使用します。
pcmk_off_action	文字列	off	**off** の代替コマンドです。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。このような場合は、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。デバイスによって、この操作が完了するのにかかる時間が、通常と大きく異なる場合があります。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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が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。Pacemaker によるオフ動作の再試行回数を変更する場合に使用します。
pcmk_list_action	文字列	list	**list** の代替コマンドです。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。このような場合は、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。デバイスによって、この操作が完了するのにかかる時間が、通常と大きく異なる場合があります。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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が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。Pacemaker による list 動作の再試行回数を変更する場合に使用します。
pcmk_monitor_action	文字列	monitor	**monitor** の代替コマンドです。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。このような場合は、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、この操作が完了するのにかかる時間が、通常と大きく異なる場合があります。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。
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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が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。Pacemaker による監視操作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。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。このような場合は、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。

フィールド	タイプ	デフォルト	説明
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、この操作が完了するのにかかる時間が、通常と大きく異なる場合があります。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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が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。Pacemaker による status 動作の再試行回数を変更する場合に使用します。
pcmk_delay_base	文字列	0s	フェンシング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。**pcmk_delay_base** パラメーターを使用して、ノードごとに異なる値を指定できます。フェンシング遅延パラメーターとその相互作用に関する一般的な情報については、フェンシング遅延 を参照してください。
pcmk_delay_max	時間	0s	フェンシング操作のランダム遅延を有効にし、ベース遅延とランダム遅延を組み合わせた最大値である最大遅延を指定します。たとえば、ベース遅延が 3 で、**pcmk_delay_max** が 10 の場合、ランダム遅延は 3 - 10 になります。フェンシング遅延パラメーターとその相互作用に関する一般的な情報については、フェンシング遅延 を参照してください。
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 (これがデフォルト値です)。値を -1 にすると無制限になります。
pcmk_on_action	文字列	on	高度な使用のみ - **on** の代替コマンドです。標準的なコマンドに対応していないデバイスや、別のコマンドを提供しているデバイスがあります。このような場合は、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	高度な使用のみ - **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。デバイスによって、この操作が完了するのにかかる時間が、通常と大きく異なる場合があります。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	高度な使用のみ - 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。複数の接続に対応していないデバイスもあります。デバイスが別のタスクでビジー状態になると操作が **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。Pacemaker による **on** 動作の再試行回数を変更する場合に使用します。

個々のフェンスデバイスに設定できるプロパティーのほかにも、以下の表で説明しているように、フェンス動作を判断するクラスタープロパティーも設定できます。

表10.3 フェンスの動作を決定するクラスタープロパティー

オプション	デフォルト	説明
stonith-enabled	true	障害が発生したノードと、停止できないリソースが含まれるノードをフェンスする必要が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、この値が **true** に設定されたクラスターのみをサポートします。
stonith-action	reboot	フェンシングデバイスに送信するアクション。使用できる値は **reboot**、**off** です。**poweroff** 値も使用できますが、レガシーデバイスでのみ使用されます。
stonith-timeout	60s	STONITH アクションが完了するのを待つ時間。
stonith-max-attempts	10	クラスターがすぐに再起動できなくなるまで、ターゲットでフェンシングが失敗する回数。

オプション	デフォルト	説明
stonith-watchdog-timeout		ノードがハードウェアウォッチドッグによって強制終了すまで待機する最大時間。この値は、ハードウェアウォッチドッグのタイムアウト値の倍に設定することが推奨されます。このオプションは、ウォッチドッグのみの SBD 設定がフェンシングに使用される場合にのみ必要です。
concurrent-fencing	true	フェンシング操作を並行して実行できるようにします。
fence-reaction	stop	独自のフェンシングの通知を受信した場合は、クラスターノードがどのように反応するかを決定します。クラスターノードは、フェンシングの設定が間違っている場合に独自のフェンシングの通知を受信するか、ファブリックフェンシングがクラスター通信を遮断しない状態である可能性があります。許可される値は、Pacemaker をすぐに停止し、停止したままにする **stop** と、ローカルノードを直ちに再起動して失敗した場合に停止する **panic** です。 このプロパティーのデフォルト値は **stop** ですが、この値に最も安全な選択肢は **panic** であり、ローカルノードを直ちに再起動しようとします。停止動作を希望する場合は、おそらくファブリックフェンシングと併用する場合は、明示的に指定することが推奨されます。
priority-fencing-delay	0 (無効)	フェンシング遅延を設定すると、スプリットブレインが発生した場合に、リソースの実行数が最も少ない、または実行中のリソースの重要性が最も低いノードがフェンスされるように、2 ノードクラスターを設定できます。フェンシング遅延パラメーターとその相互作用に関する一般的な情報については、フェンシング遅延 を参照してください。

クラスターのプロパティーの設定は、クラスターのプロパティーの設定と削除 を参照してください。

10.4. フェンシング遅延

2 ノードクラスターでクラスター通信が失われると、一方のノードがこれを先に検出し、他方のノードをフェンスすることがあります。ただし、両方のノードが同時に検出した場合、各ノードが他方のノードのフェンシングを開始した結果、両方のノードの電源がオフになるかリセットされる可能性がありま

す。フェンシング遅延を設定すると、両方のクラスターノードが相互にフェンスし合う可能性を減らすことができます。3 つ以上のノードを持つクラスターでも遅延を設定できますが、クォーラムのあるパーティションしかフェンシングを開始しないため、通常は利点がありません。

システム要件に応じて、さまざまなタイプのフェンシング遅延を設定できます。

- **静的フェンシング遅延**
  静的フェンシング遅延は、事前に定義された固定の遅延です。1 つのノードに静的遅延を設定すると、そのノードがフェンシングされる可能性が高くなります。これは、通信の切断を検出した後、他のノードが先にフェンシングを開始する可能性が高まるためです。active/passive クラスターでは、passive ノードに遅延を設定すると、通信が切断されたときに passive ノードがフェンスされる可能性が高くなります。静的遅延を設定するには、**pcs_delay_base** クラスタープロパティーを使用します。このプロパティーは、各ノードに個別のフェンスデバイスが使用される場合、または単一のフェンスデバイスがすべてのノードに使用される場合に設定できます。

- **動的フェンシング遅延**
  動的フェンシング遅延はランダムです。この遅延は変化する可能性があり、フェンシングが必要なタイミングで決定されます。ランダム遅延を設定し、ベース遅延とランダム遅延を組み合わせた最大値を **pcs_delay_max** クラスタープロパティーで指定します。各ノードのフェンシング遅延がランダムの場合、どのノードがフェンスされるかもランダムです。この機能は、active/active 設計のすべてのノードに対して単一のフェンスデバイスを使用してクラスターが設定されている場合に便利です。

- **優先フェンシング遅延**
  優先フェンシング遅延は、アクティブなリソースの優先度に基づきます。すべてのリソースの優先度が同じ場合、実行中のリソースが最も少ないノードがフェンスされます。ほとんどの場合、遅延関連のパラメーターは 1 つしか使用しませんが、複数のパラメーターを組み合わせることも可能です。遅延関連のパラメーターを組み合わせると、リソースの優先度の値が加算されて、総遅延となります。優先フェンシング遅延は、**priority-fencing-delay** クラスタープロパティーを使用して設定します。この機能を使用すると、ノード間の通信が失われたときに、実行中のリソースが最も少ないノードがフェンスされる可能性が高くなるため、active/active クラスター設計で役立つ場合があります。

pcmk_delay_base クラスタープロパティー

pcmk_delay_base クラスタープロパティーを設定すると、フェンシングのベース遅延が有効になり、ベース遅延の値が指定されます。

pcmk_delay_base プロパティーに加えて **pcmk_delay_max** クラスタープロパティーを設定すると、この静的遅延にランダム遅延を追加した合計値が最大遅延を下回るように、全体の遅延が導出されます。**pcmk_delay_base** を設定し、**pcmk_delay_max** を設定しない場合は、遅延にランダムなコンポーネントは含まれず、遅延は **pcmk_delay_base** の値となります。

pcmk_delay_base パラメーターを使用して、ノードごとに異なる値を指定できます。これにより、ノードごとに異なる遅延を使用して、単一のフェンスデバイスを 2 ノードクラスターで使用できます。別個の遅延を使用するために 2 つの別個のデバイスを設定する必要はありません。ノードごとに異なる値を指定するには、**pcmk_host_map** と同様の構文を使用して、ホスト名をそのノードの遅延値にマップします。たとえば、**node1:0;node2:10s** は、**node1** をフェンシングするときに遅延を使用せず、**node2** をフェンシングするときに 10 秒の遅延を使用します。

pcmk_delay_max クラスタープロパティー

pcmk_delay_max クラスタープロパティーを設定すると、フェンシング操作のランダム遅延が有効になり、ベース遅延とランダム遅延を組み合わせた最大値である最大遅延が指定されます。たとえば、ベース遅延が 3 で、**pcmk_delay_max** が 10 の場合、ランダム遅延は 3 - 10 になります。

pcmk_delay_max プロパティーに加えて **pcmk_delay_base** クラスタープロパティーを設定すると、この静的遅延にランダム遅延を追加した合計値が最大遅延を下回るように、全体の遅延が導出されます。**pcmk_delay_max** を設定し、**pcmk_delay_base** を設定しない場合は、遅延に静的なコンポーネントは含まれません。

priority-fencing-delay クラスタープロパティー

priority-fencing-delay クラスタープロパティーを設定すると、スプリットブレインが発生した場合に、リソースの実行数が最も少ない、または実行中のリソースの重要性が最も低いノードがフェンスされるように、2 ノードクラスターを設定できます。

priority-fencing-delay プロパティーは期間に設定できます。このプロパティーのデフォルト値は 0 (無効) です。このプロパティーがゼロ以外の値に設定されている場合や、priority メタ属性が 1 つ以上のリソースに対して設定されている場合は、スプリットブレインが発生すると、実行中の全リソースの合計優先度が最も高いノードが稼働状態を維持する可能性が高くなります。たとえば、**pcs resource defaults update priority=1** と **pcs property set priority-fencing-delay=15s** を設定し、他の優先度が設定されていない場合には、最も多くのリソースを実行するノード以外はフェンシングを開始するまで 15 秒間待機するため、最も多くのリソースを実行するノードが稼働状態を維持する可能性が高くなります。特定のリソースが他のリソースよりも重要である場合は、優先度を高く設定できます。

昇格可能なクローンに優先度が設定されている場合、そのクローンのプロモートロールを実行しているノードの優先度が 1 ポイント追加されます。

フェンシング遅延の相互作用

複数のタイプのフェンシング遅延を設定すると、以下のようになります。

- **priority-fencing-delay** プロパティーで遅延を設定すると、その遅延は **pcmk_delay_base** と **pcmk_delay_max** のフェンスデバイスプロパティーの遅延に追加されます。この動作により、両方のノードの優先度が同等の場合、またはノードの損失以外の理由で両方のノードをフェンシングする必要がある場合 (たとえば、**on-fail=fencing** がリソースモニター操作用に設定されている場合)、ある程度の遅延を許容します。これらの遅延を組み合わせて設定する場合は、優先ノードが優先されるよう、**priority-fencing-delay** プロパティーを、**pcmk_delay_base** および **pcmk_delay_max** の最大遅延よりもはるかに大きい値に設定します。このプロパティーを 2 倍の値に設定すると、常に安全です。

- Pacemaker 自体がスケジュールしたフェンシングしか、フェンシング遅延を監視しません。**dlm_controld** などの外部コードでスケジュールされるフェンシングや、**pcs stonith fence** コマンドで実装されるフェンシングは、フェンスデバイスに必要な情報を提供しません。

- 個々のフェンスエージェントの中には遅延パラメーターが実装されたものがあります。このパラメーターは、エージェントによって決定された名前を持ち、**pcmk_delay_*** プロパティーで設定された遅延の影響は受けません。両方の遅延が設定されている場合は、その両方が一緒に追加され、通常は併用されません。

10.5. 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設定しているため、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。

手順

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。
フェンシング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がフェンシングデバイスへのアクセスを妨げていないこと、フェンシング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

これらの例では、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、パスワードなどの情報と、その他のフェンスデバイスに関する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、フェンシングは実行せずに、別のノードのフェンスデバイスインターフェイ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、動作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 1 つの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
```

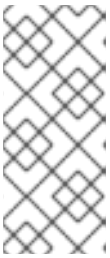
フェンスエージェントがステータス、オフ、オン、または再起動の動作を適切に実行しない場合は、ハードウェア、フェンスデバイスの設定、およびコマンドの構文を確認する必要があります。さらに、デバッグ出力を有効にした状態で、フェンスエージェントスクリプトを実行できます。デバッグ出力は、一部のフェンスエージェントで、フェンスデバイスにログインする際に、フェンスエージェントスクリプトに問題が発生しているイベントシーケンスの場所を確認するのに役に立ちます。

```
# 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に 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、継続して失敗したその他の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 コマンドに成功すると、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すると、クラスター管理が取得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、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。パスワードとユーザー名を引用符で囲んで入力すると、この問題に対処できます。
- **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ます。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。ただし、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イ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イ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
```

10.6. 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に対応します。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。

Pacemaker は、以下のようにフェンシングレベルを処理します。

- 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、現在のレベルの処理が終了します。同レベルのデバイスには試行されず、次のレベルが試行されます。
- すべてのデバイスの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 をコンマ区切りのリストとして指定します。stonith ID が、指定したレベルで試行されます。

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

次のコマンドを使用すると現在設定されている全フェンスレベルが表示されます。

```
pcs stonith level
```

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** に障害が発生し、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、レベル設定後の **pcs stonith level** コマンドの出力も表示されています。

```
# 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って指定します。空白は入力しないでください。以下に例を示します。

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

次のコマンドは、フェンスレベルで指定されたフェンスデバイスとノードがすべて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** がフェンスデバイス **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** がフェンスデバイス **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
```

10.7. 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。このとき、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
```

10.8. 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** が指定されている場合、コマンドはその設定されたフェンシングデバイスのみのオプションを表示します。**--full** オプションが指定されている場合、すべての設定されたフェンシングオプションが表示されます。

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

10.9. PCS コマンドとしてのフェンスデバイスのエクスポート

Red Hat Enterprise Linux 9.1 では、**pcs stonith config** コマンドの **--output-format=cmd** オプションを使用して、別のシステムに設定済みのフェンスデバイスを再作成するのに使用できる **pcs** コマンドを表示できます。

次のコマンドは、**fence_apc_snmp** フェンスデバイスを作成し、デバイスを再作成するために使用できる **pcs** コマンドを表示します。

```
# pcs stonith create myapc fence_apc_snmp ip="zapc.example.com"
pcmk_host_map="z1.example.com:1;z2.example.com:2" username="apc" password="apc"
# pcs stonith config --output-format=cmd
Warning: Only 'text' output format is supported for stonith levels
pcs stonith create --no-default-ops --force -- myapc fence_apc_snmp \
  ip=zapc.example.com password=apc 'pcmk_host_map=z1.example.com:1;z2.example.com:2'
username=apc \
  op \
    monitor interval=60s id=myapc-monitor-interval-60s
```

10.10. フェンスデバイスの修正と削除

次のコマンドを使用して、現在設定されているフェンシングデバイスのオプションを変更または追加します。

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

pcs stonith update コマンドを使用して SCSI フェンシングデバイスを更新すると、フェンシングリソースが実行されていたものと同じノードで実行中のすべてのリソースが再起動されます。以下のコマンドのいずれかのバージョンを使用して、他のクラスターリソースを再起動しなくても SCSI デバイスを更新できます。RHEL 9.1 では、SCSI フェンシングデバイスをマルチパスデバイスとして設定できます。

```
pcs stonith update-scsi-devices stonith_id set device-path1 device-path2
pcs stonith update-scsi-devices stonith_id add device-path1 remove device-path2
```

現在の設定からフェンシング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
```

10.11. 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スできます。**--off** を指定すると、stonith に API コールの **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、フェンスデバイスがそのノードをフェンスできない状況では、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、ノードの電源が切れたことを手動で確認した後、次のコマンドを入力して、ノードの電源が切れたことをクラスターに確認し、そのリソースを回復のために解放できます。

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
```

10.12. フェンスデバイスの無効化

フェンシングデバイス/リソースを無効にする場合は、**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
```

10.13. ノードがフェンシングデバイスを使用しないように設定する手順

特定のノードがフェンシング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
```

10.14. 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**)。それ以外の場合は、ACPI Soft-Off が有効になっていると、統合フェンスデバイスがノードをオフにするのに 4 秒以上かかることがあります (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、フェンシングが遅延するか、失敗します。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

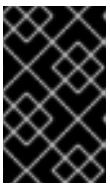
ノードのフェンシングにかかる時間は、使用している統合フェンスデバイスによって異なります。統合フェンスデバイスの中には、電源ボタンを押し続けるのと同じ動作を実行するものもあります。この場合は、ノードがオフになるのに 4 秒から 5 秒かかります。また、電源ボタンを押してすぐ離すのと同等の動作を行い、ノードの電源をオフにする行為をオペレーティングシステムに依存する統合フェンスデバイスもあります。この場合は、ノードがオフになるのにかかる時間は 4~5 秒よりも長くなります。

- 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。

- **/etc/systemd/logind.conf** ファイルに **HandlePowerKey=ignore** を設定し、以下のように logind.conf ファイルで ACPI Soft-Off の無効化に記載されているように、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されているように、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。

10.14.1. 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。

手順

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) 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同等の設定) に設定します。**BIOS CMOS 設定ユーティリティー** 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。

注記

ACPI Function、**Soft-Off by PWR-BTTN**、および **Instant-Off** に相当するものは、コンピューターによって異なります。ただし、この手順の目的は、電源ボタンを使用して遅延なしにコンピューターをオフにするように BIOS を設定することです。

4. **BIOS CMOS Setup Utility** プログラムを終了して、BIOS 設定を保存します。
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。フェンスデバイスをテストする方法は 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。

10.14.2. logind.conf ファイルで ACPI Soft-Off の無効化

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

手順

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

```
HandlePowerKey=ignore
```

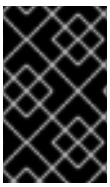
2. **systemd-logind** サービスを再起動します。

```
# systemctl restart systemd-logind.service
```

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。フェンスデバイスをテストする方法は フェンスデバイスのテスト を参照してください。

10.14.3. GRUB 2 ファイルでの 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。

手順

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。フェンスデバイスをテストする方法は フェンスデバイスのテスト を参照してください。

第11章 クラスターリソースの設定

次のコマンドを使用して、クラスターリソースを作成および削除します。

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_id] [clone_options] | promotable [clone_id] [clone_options] [--
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。

- **--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。

クラスター内に作成できるリソースの数に制限はありません。

リソースの制約を設定して、クラスター内のそのリソースの動作を指定できます。

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** で、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かどうかを確認します。

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

または、以下のように、standard フィールドおよび provider フィールドを省略できます。規格とプロバイダーはそれぞれ **ocf** と **heartbeat** にデフォルト設定されます。

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

設定済みリソースの削除

次のコマンドを使用して、設定済みのリソースを削除します。

```
pcs resource delete resource_id
```

たとえば、次のコマンドは、リソース ID が **VirtualIP** の既存リソースを削除します。

```
# pcs resource delete VirtualIP
```

11.1. 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。

以下の表は、リソースエージェントのこれらのプロパティーについて説明しています。

表11.1 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規格。使用でき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ます。OCF スタイルのメタデータは、**/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。

以下の表には、利用可能なリソースプロパティーを表示するコマンドをまとめています。

表11.2 リソースプロパティーを表示させるコマンド

pcs 表示コマンド	出力
pcs resource list	利用できる全リソースのリストを表示
pcs resource standards	利用できるリソースエージェントのリストを表示
pcs resource providers	利用できるリソースエージェントプロバイダーのリストを表示
pcs resource list string	利用できるリソースを指定文字列でフィルターしたリストを表示。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。

11.2. 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、**apache** タイプ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.

...
```

11.3. リソースのメタオプションの設定

リソースには、リソース固有のパラメーターの他に、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。

以下の表は、リソースのメタオプションを示しています。

表11.3 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**	クラスターがこのリソースを維持しようとする状態を示します。設定できる値は以下のとおりです。 * **Stopped** - リソースの強制停止 * **Started** - リソースの起動を許可 (昇格可能なクローンの場合、適切であれば昇格される) * **Promoted** - リソースの起動を許可し、適切であれば昇格を許可 * **Unpromoted** - リソースの起動を許可、ただしリソースが昇格可能な場合、アンプロモートモードに限る
is-managed	**true**	クラスターによるリソースの起動および停止を許可するかどうかを示します。使用できる値は **true** または **false** です。
resource-stickiness	1	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	リソースを起動できる条件を示します。 以下の条件を除き、**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** - クラスターは常にリソースを開始できます。 * **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を起動できます。 * **unfencing** - 設定されているノードの過半数がアクティブで、**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おり、**フェンシング** されていないノードに **限り**、クラスターはこのリソースを起動できます。**provides=unfencing stonith** メタオプションがフェンシング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、そのノードには、このリソースのホストとして不適格とするマークが付けられます。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付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。障害が発生していないかのように動作し、障害が発生したノードにリソースを戻せるようになるまで待機する秒数を示します。
multiple-active	**stop_start**	リソースが複数のノードでアクティブであることが検出された場合に、クラスターが実行すべき動作を示します。設定できる値は以下のとおりです。 * **block** - リソースに unmanaged のマークを付けます。 * **stop_only** - 実行中の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。 * **stop_unexpected** - (RHEL 9.1 以降) リソースの予期しないインスタンスのみを停止し、完全な再起動は必要ありません。ユーザーは、サービスとそのリソースエージェントが、完全に再起動しなくても追加のアクティブインスタンスで機能することを確認する必要があります。
critical	**true**	リソースがリソースグループに含まれる場合に作成された暗黙的なコロケーションの成約など、従属リソース (**target_resource**) としてリソース関連のコロケーション成約すべてに、**influence** オプションのデフォルト値を設定します。**influence** コロケーション制約オプションは、従属リソースが移行のしきい値に達して失敗した場合に、クラスターで別のノードにプライマリーリソースと従属リソースを移行するか、クラスターでサーバーの切り替えなしに従属リソースをオフラインのままにするかを決定します。**critical** リソースのメタオプションには、**true** または **false** の値を指定できます。デフォルト値は **true** です。

フィールド	デフォルト	説明
allow-unhealthy-nodes	**false**	(RHEL 9.1 以降) **true** に設定されている場合、ノードの正常性が低下したことでリソースがノードから強制的に切り離されることはありません。正常性リソースにこの属性が設定されている場合、クラスターはノードの正常性が回復したかどうかを自動的に検出し、リソースをノードに戻すことができます。ノードの正常性は、ローカルの状態に基づいて正常性リソースエージェントによって設定された正常性属性と、クラスターがそれらの状態にどのように反応するかを決定する戦略関連のオプションの組み合わせによって決定されます。

11.3.1. リソースオプションのデフォルト値の変更

pcs resource defaults update コマンドを使用して、すべてのリソースのリソースオプションのデフォルト値を変更できます。たとえば、次のコマンドは、**resource-stickiness** のデフォルト値を 100 にリセットします。

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

以前のリリースのすべてのリソースのデフォルトを設定する元の **pcs resource defaults name=value** コマンドは、複数のデフォルトが設定されない限りサポートされます。ただし、**pcs resource defaults update** が、コマンドの推奨されるバージョンになりました。

11.3.2. リソースセットのリソースオプションのデフォルト値の変更

pcs resource defaults set create コマンドを使用して、複数のリソースのデフォルトセットを作成できます。これにより、**resource** 式を含むルールを指定できます。このコマンドで指定したルールでは、**and**、**or** および括弧を含め、**resource**、**date** 式のみを使用できます。

pcs resource defaults set create コマンドを使用して、特定タイプの全リソースにデフォルトのリソース値を設定できます。たとえば、停止に時間がかかるデータベースを実行している場合は、データベースタイプの全リソースで **resource-stickiness** のデフォルト値を増やすことで、想定している頻度よりも多く、このようなリソースが他のノードに移動されるのを回避できます。

以下のコマンドは、**pqsql** タイプの全リソースに、**resource-stickiness** のデフォルト値を 100 に設定します。

- リソースのデフォルトセット名を指定する **id** オプションは必須ではありません。このオプションを設定すると、**pcs** が自動的に ID を生成します。この値を設定すると、より分かりやすい名前に設定できます。
- この例では、**::pgsql** は、クラスやプロバイダーは任意でタイプが **pgsql** を指定します。
  - **ocf:heartbeat:pgsql** を指定すると、クラスが **ocf**、プロバイダーが **heartbeat**、タイプが **pgsql** に指定されます。
  - **ocf:pacemaker:** を指定すると、タイプは任意でクラスが **ocf**、プロバイダーが **pacemaker** に指定されます。

```
# pcs resource defaults set create id=pgsql-stickiness meta resource-stickiness=100 rule
resource ::pgsql
```

既存セットのデフォルト値を変更する場合は、**pcs resource defaults set update** コマンドを使用します。

11.3.3. 現在設定されているリソースのデフォルトの表示

pcs resource defaults コマンドは、指定したルールなど、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リストを表示します。

次の例では **resource-stickiness** のデフォルト値を 100 にリセットした後のコマンド出力を示しています。

```
# pcs resource defaults
Meta Attrs: rsc_defaults-meta_attributes
  resource-stickiness=100
```

以下の例では、タイプが **pqsql** の全リソースの **resource-stickiness** のデフォルト値を 100 にリセットし、**id** オプションを **id=pgsql-stickiness** に設定します。

```
# pcs resource defaults
Meta Attrs: pgsql-stickiness
  resource-stickiness=100
  Rule: boolean-op=and score=INFINITY
    Expression: resource ::pgsql
```

11.3.4. 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におけるデフォルト値のリセットの有無に関わらず、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** コマンドの形式です。

```
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
```

以下の例では、**dummy_resource** という名前の既存リソースがあります。このコマンドは、**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 秒でリソースが同じノード上で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
  ...
```

11.4. リソースグループの設定

クラスターの最も一般的な設定要素の 1 つがリソースセットです。リソースセットはまとめて配置し、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。

11.4.1. 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、グループに追加す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別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** オプションを使用して、追加するリソースの位置を、そのグループにすでに含まれるリソースを基準にして指定できます。

以下のコマンドを使用して、リソースを作成するときに、既存のグループに新しいリソースを追加することもできます。以下のコマンドでは、作成するリソースが group_name 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。

- グループ内に、複数のリソースが置かれています。
- 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されたリソースは実行できません。
- 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** が含まれるリソースグループ **shortcut** 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。

11.4.2. 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...
```

11.4.3. 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リスト表示します。

```
pcs resource group list
```

11.4.4. グループオプション

リソースグループには、**priority** オプション、**target-role** オプション、または **is-managed** オプションを設定できます。このオプションは、1 つのリソースに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、リソースのメタオプションの設定 を参照してください。

11.4.5. 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。グループの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、グループ全体でスコアが 500 の現在の場所が優先されます。

11.5. リソース動作の決定

リソースの制約を設定して、クラスター内のそのリソースの動作を指定できます。以下の制約のカテゴリーを設定できます。

- **場所** の制約 - この制約は、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。
- **順序** の制約 - この制約は、リソースが実行する順序を決定します。順序の制約を設定する方法は、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。
- **コロケーション** の制約 - この制約は、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。コロケーションの制約の詳細は、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。

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。

第12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、たとえば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ことを決定できます。

場所の制約に加え、リソースが実行されるノードは、そのリソースの **resource-stickiness** 値に影響されます。これは、リソースが現在実行しているノードに留まることをどの程度優先するかを決定します。**resource-stickiness** 値の設定に関する詳細は、現在のノードを優先するようにリソースを設定 を参照してください。

12.1. 場所の制約の設定

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

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]] ...
```

次のコマンドは、リソースが指定ノード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]] ...
```

次の表は、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12.1 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

フィールド	説明
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正の整数値。**INFINITY** は、リソースの場所制約のデフォルト **score** 値です。 **pcs constraint location rsc prefers** 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、リソースがそのノードで優先的に実行します。ただし、そのノード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。 **pcs constraint location rsc avoids** コマンドで **score** に **INFINITY** を指定すると、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ないことを示します。これは、**-INFINITY** のスコアで **pcs constraint location add** コマンドを設定するのと同じです。 数値スコア (**INFINITY** 以外) は、その制約が任意で、それを上回る要因が他に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** される場合は、リソースがその場所に残されます。

以下のコマンドは、**Webserver** リソースが、**node1** ノード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関する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。

次のコマンドは、**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 で説明しているように、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
```

12.2. ノードのサブセットへのリソース検出を制限

Pacemaker がどこでリソースを開始しても、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監視操作 (プローブとも呼ばれています) を実行します。このリソース検出のプロセスは、監視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** オプションを指定して、指定したリソースに対して、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。物理的にリソースが稼働可能なノードのサブセットで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。このコマンドでは、基本的な場所の制約に対応します。score を正の値にすると、リソースが特定のノードで優先的に実行するように設定されます。score を負の値にすると、リソースがノードを回避するように設定されます。基本的な場所の制約と同様に、制約にリソースの正規表現を使用することもできます。

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

以下の表は、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12.2 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** の値を **INFINITY** に設定すると、そのノードが利用可能な場合は、リソースがそのノードで優先的に実行します。ただし、そのノード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。**score** の値を **-INFINITY** に設定すると、他のノードが利用できない場合でも、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、その制約が任意で、それを上回る要因が他に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** される場合は、リソースが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。これは、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行いません。 * **exclusive** - このノード (および同様に **exclusive** マークが付いているその他のノード) 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。複数のノードで同じリソースの **exclusive** 検出を使用する複数の場所制約により、**resource-discovery** が排他的であるノードのサブセットが作成されます。1つまたは複数のノードで、リソースが **exclusive** 検出用にマーク付けされている場合、そのリソースは、ノードのサブセット内にのみ配置できます。

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。

12.3. 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、どのノードでも実行できません。そして、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るクラスターを設定してから、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、優先する設定やクラスターの設定により異なります。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。

12.3.1. 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場合は、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、いずれのノードでもリソースの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** リソースでは **example-1** ノードが優先され、**Database** リソースでは **example-2** ノードが優先されるようにし、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** ノードにフェイ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
```

12.3.2. 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** が明示的に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。全ノードのスコアは暗黙で 0 になるため、優先ノードに障害が発生した場合はいずれのリソースも **example-3** ノードにフェイ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
```

INFINITY は、スコアのデフォルト値であるため、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。

12.4. 現在のノードを優先するリソースの設定

リソースには、リソースのメタオプションの設定 で説明されているように、リソースの作成時にメタ属性として設定できる **resource-stickiness** 値があります。**resource-stickiness** 値は、現在実行しているノード上にリソースが残す量を決定します。Pacemaker は、他の設定 (場所の制約の score 値など) とともに **resource-stickiness** 値を考慮して、リソースを別のノードに移動するか、そのまま残すかを決定します。

resource-stickiness の値を 0 にすると、クラスターは、必要に応じてリソースを移動して、ノード間でリソースのバランスを調整できます。これにより、関連のないリソースが起動または停止したときにリソースが移動する可能性があります。stickiness が高くなると、リソースは現在の場所に留まり、その他の状況が stickiness を上回る場合に限り移動するようになります。これにより、新しく追加したノードに割り当てられたリソースは、管理者の介入なしには利用できなくなる可能性があります。

RHEL 9 で新規に作成したクラスターでは、**resource-stickiness** のデフォルト値が 1 に設定されています。この小さい値は、作成する他の制約で簡単に上書きできますが、Pacemaker がクラスター全体で正常なリソースを不必要に移動しないようにするには十分です。**resource-stickiness** の値 0 によるクラスターの挙動が望ましい場合は、次のコマンドを使用して、**resource-stickiness** のデフォルト値を 0 に変更できます。

```
# pcs resource defaults update resource-stickiness=0
```

既存のクラスターを RHEL 9 にアップグレードし、**resource-stickiness** にデフォルト値を明示的に設定していない場合は、**resource-stickiness** の値が 0 のままになり、**pcs resource defaults** コマンドに stickiness の値が表示されなくなります。

resource-stickiness に正の値が設定されている場合、リソースは新たに追加されたノードに移動しません。この時点でリソースバランスが必要な場合は、**resource-stickiness** の値を一時的に 0 に設定できます。

場所の制約スコアが **resource-stickiness** の値よりも大きい場合には、クラスターは場所の制約が指定するノードに、正常なリソースを依然として移動する可能性があります。

Pacemaker がリソースを配置する場所を決定する方法の詳細は、ノード配置ストラテジーの設定 を参照してください。

12.5. PCS コマンドとしてのリソース制約のエクスポート

Red Hat Enterprise Linux 9.3 では、**pcs constraint** コマンドの **--output-format=cmd** オプションを使用して、設定済みのリソース制約を別のシステムで再作成するのに使用できる **pcs** コマンドを表示できます。

次のコマンドは、**IPaddr2** リソースと **apache** リソースを作成します。

```
# pcs resource create VirtualIP IPaddr2 ip=198.51.100.3 cidr_netmask=24
Assumed agent name 'ocf:heartbeat:IPaddr2' (deduced from 'IPaddr2')
# pcs resource create Website apache configfile="/etc/httpd/conf/httpd.conf"
statusurl="http://127.0.0.1/server-status"
Assumed agent name 'ocf:heartbeat:apache' (deduced from 'apache')
```

次のコマンドは、2 つのリソースの場所の制約、コロケーション制約、および順序制約を設定します。

```
# pcs constraint location Website avoids node1
# pcs constraint colocation add Website with VirtualIP
# pcs constraint order VirtualIP then Website
Adding VirtualIP Website (kind: Mandatory) (Options: first-action=start then-action=start)
```

リソースと制約を作成した後に次のコマンドを実行すると、別のシステムで制約を再作成するために使用できる **pcs** コマンドが表示されます。

```
# pcs constraint --output-format=cmd
pcs -- constraint location add location-Website-node1--INFINITY resource%Website node1 -
INFINITY;
pcs -- constraint colocation add Website with VirtualIP INFINITY \
  id=colocation-Website-VirtualIP-INFINITY;
pcs -- constraint order start VirtualIP then start Website \
  id=order-VirtualIP-Website-mandatory
```

第13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、順序の制約を設定できます。

次は、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]
```

以下の表では、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まとめています。

表13.1 順序の制約のプロパティー

フィールド	説明
resource_id	動作を行うリソースの名前。
action	リソースに対して指示されるアクション。**action** プロパティーでは、以下の値が使用できます。 * **start** - リソースの開始アクションを指示します。 * **stop** - リソースの停止アクションを指示します。 * **promote** - アンプロモートリソースから、プロモートしたリソースにリソースを昇格します。 * **demote** - プロモートリソースから、アンプロモートリソースまで、リソースを降格します。 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** - 常に制約を有効にします (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場合や起動できない場合は、2 番目に指定したリソースが停止します。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 番目および 2 番目のリソースは、いずれの順序でも起動できますが、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、リソースの起動によりホストの負荷が高くなる場合です。

フィールド	説明
symmetrical オプション	true の場合、逆の制約は逆のアクションに適用されます (たとえば、A の開始後に B が開始する場合は、B が A が停止前に停止する場合など)。**kind** が **Serialize** の順序制約を対称にすることはできません。デフォルト値は、**Mandatory** および **Optional** の場合は **true** で、**Serialize** の場合は **false** です。

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 resource1 [resourceN]...
```

13.1. 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限り、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** に設定されているか、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 オプションが **true** に設定されているか、デフォルトのままにすると、逆のアクションの命令は逆順になります。**start** と **stop** のアクションは対称になり、**demote** と **promote** は対称になります。たとえば、対称的に、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。対称順序は、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すること示しています。たとえば、A then B と設定した場合は、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がありません。

13.2. 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 番目に指定しているリソースの状態を変更しても、2 番目に指定しているリソースには影響しません。

次のコマンドは、**VirtualIP** 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
```

13.3. リソースセットへの順序の設定

一般的に、管理者は、複数のリソースの連鎖を作成する場合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、このような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、1 つまたは複数のリソース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。それ以外の場合は、制約を設定しても効果がありません。
- **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** に設定すると、次のセットに進む前に、セットの 1 つのリソースのみを開始する必要があります。**require-all** を **false** に設定しても、**sequential** が **false** に設定されている順序なしセットと併用しない限り、効果はありません。デフォルト値は **true** です。
- **action** - クラスターリソースの実行順序の決定 の表順序の制約のプロパティーで説明されているように、**start**、**promote**、**demote**、または **stop** に設定できます。
- **ロール** - **Stopped**、**Started**、**Promoted**、または **Unpromoted** 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** - クラスターリソースの実行順序の決定 の表順序の制約のプロパティーで説明されているように、制約を有効にする方法を指定します。
- **symmetrical** - クラスターリソースの実行順序の決定 の表順序の制約のプロパティーで説明しているように、逆の作用に逆の制約を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は、次のコマンドを実行すると、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、起動するリソースセットに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
```

13.4. PACEMAKER で管理されないリソース依存関係の起動順序の設定

クラスターは、クラスターが管理していない依存関係を持つリソースを含めることができます。この場合は、Pacemaker を起動する前にその依存関係を起動し、Pacemaker が停止した後に停止する必要があります。

systemd resource-agents-deps ターゲットを使用してこの条件を設定するために、スタートアップ順序を設定できます。このターゲットに対して **systemd** ドロップインユニットを作成すると、Pacemaker はこのターゲットに対して相対的な順序を適切に設定できます。

たとえば、クラスターが管理していない外部サービス **foo** に依存するリソースがクラスターに含まれている場合は、以下の手順を実行します。

1. 以下を含むドロップインユニット **/etc/systemd/system/resource-agents-deps.target.d/foo.conf** を作成します。

```
[Unit]
Requires=foo.service
After=foo.service
```

2. **systemctl daemon-reload** コマンドを実行します。

この方法で指定するクラスターの依存関係はサービス以外のものとなります。たとえば、**/srv** にファイルシステムをマウントする依存関係がある場合は、以下の手順を実行してください。

1. **/etc/fstab** ファイルに **/srv** が記載されていることを確認します。これは、システムマネージャーの設定が再読み込みされる際に、システムの起動時に **systemd** ファイルの **srv.mount** に自動的に変換されます。詳細は、man ページの **systemd.mount**(5) および **systemd-fstab-generator**(8) を参照してください。
2. ディスクのマウント後に Pacemaker が起動するようにするには、以下を含むドロップインユニット **/etc/systemd/system/resource-agents-deps.target.d/srv.conf** を作成します。

```
[Unit]
Requires=srv.mount
After=srv.mount
```

3. **systemctl daemon-reload** コマンドを実行します。

Pacemaker クラスターが使用する LVM ボリュームグループに、iSCSI ターゲットなど、リモートブロックストレージに存在する 1 つ以上の物理ボリュームが含まれている場合は、Pacemaker が起動する前にサービスが開始されるように、ターゲット用に **systemd resource-agents-deps** ターゲットと

systemd ドロップインユニットを設定することができます。

以下の手順では、**blk-availability.service** を依存関係として設定します。**blk-availability.service** サービスは、**iscsi.service** などのサービスが含まれるラッパーです。お使いのデプロイメントでこれが必要な場合は、**blk-availability** の代わりに **iscsi.service**(iSCSI のみ) または **remote-fs.target** を依存関係として設定できます。

1. 以下を含むドロップインユニット **/etc/systemd/system/resource-agents-deps.target.d/blk-availability.conf** を作成します。

```
[Unit]
Requires=blk-availability.service
After=blk-availability.service
```

2. **systemctl daemon-reload** コマンドを実行します。

第14章 クラスターリソースのコロケーション

あるリソースの場所を、別のリソースの場所に依存させるように指定する場合は、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制約を作成すると、リソースがノードに割り当てられる割り当て順序に重要な影響を及ぼす点に注意してください。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にリソース A を配置することができません。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を設定するのか、もしくはリソース B をリソース A に対してコロケーションを設定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を設定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に、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 [promoted|unpromoted] **source_resource** with [promoted|unpromoted] **target_resource** [**score**] [**options**]

以下の表は、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まとめています。

表14.1 コロケーション制約のパラメーター

パラメーター	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。クラスターは、このリソースの配置先を決定してから、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ます。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します。-**INFINITY** は、source_resource を target_resource と同じノードで実行してはならないことを示しています。

パラメーター	説明
influence オプション	従属リソースが移行のしきい値に達して失敗した場合に、クラスターで別のノードにプライマリーリソース (source_resource) と従属リソース (target_resource) を移行するか、クラスターでサーバーの切り替えなしに従属リソースをオフラインのままにするかを決定します。 **influence** コロケーション制約オプションには、**true** または **false** の値を指定できます。このオプションのデフォルト値は、従属リソースの **critical** リソースメタオプションの値で決定します。デフォルト値は **true** です。 このオプションの値が **true** の場合、Pacemaker は、プライマリーリソースと依存するリソース両方をアクティブに維持しようとします。依存するリソースが障害の移行しきい値に達すると、可能な場合は両方のリソースが別のノードに移行します。 このオプションの値が **false** の場合、Pacemaker は、依存するリソースのステータスが原因でプライマリーリソースを移行しないようにします。この場合、依存するリソースが障害の移行しきい値に達すると、プライマリーリソースがアクティブで、現在のノードに留まると停止します。

14.1. 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** には、target_resource がアクティブではないケースも含まれます。

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、(何らかの理由で) **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は、**myresource1** の実行が許可されません。

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
```

ここでも、**-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** がすでに実行されている場合は、いずれのノードでも **myresource1** を実行できなくなります。

14.2. リソースの勧告的な配置の指定

リソースの勧告的な配置は、リソースの配置は推奨ではあるものの、必須ではありません。制約のスコ

アが **-INFINITY** より大きく、**INFINITY** より小さい場合、クラスターは希望の設定に対応しようとしますが、クラスターリソースの一部を停止するという選択肢がある場合には、その設定が無視される場合があります。

14.3. 複数リソースのコロケーション

お使いの設定で、コロケーションと起動の順番を指定してリソースを作成する必要がある場合には、このような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、コロケーションを設定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もあります。

- リソースのセットにコロケーションを設定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 のいずれかに対してコロケーションを設定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 の両方に対してコロケーションを設定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に対してコロケーションを設定できます。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に限り有効です。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。
- **ロール** - **Stopped**、**Started**、**Promoted**、または **Unpromoted** 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、場所の制約の設定 の表の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に対してコロケーションが設定されます。たとえば、set A B は B が A の場所に配置されることを意味します。しかし、複数のセットをリストする場合、各セットはその後のメンバーと同じ場所に配置されます。たとえば、set C D sequential=false set A B は、C と D のセットが、A と B のセットと同じ場所に配置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 にはコロケーションが設定されています)。

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]]
```

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

pcs constraint colocation remove **source_resource target_resource**

第15章 リソース制約とリソース依存関係の表示

設定した制約を表示させるコマンドがいくつかあります。設定されたリソース制約をすべて表示するか、特定のタイプのリソース制約だけに表示を制限することもできます。また、設定したリソース依存関係を表示できます。

設定済みの全制約の表示

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。**--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 [list|show] [--full]
```

デフォルトでは、リソースの制約のリストには、期限切れの制約は表示されません。リストに期限切れの制約を含めるには、**pcs constraint** コマンドに **--all** オプションを使用します。これにより、期限切れの制約のリストが表示され、制約とそれに関連するルールが **(expired)** として表示されます。

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リスト表示します。

- **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]
```

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。

```
pcs constraint order [show]
```

コロケーション制約の表示

次のコマンドは、現在のコロケーション制約をリスト表示します。

```
pcs constraint colocation [show]
```

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リスト表示します。

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

リソースの依存関係の表示

次のコマンドは、クラスターリソース間の関係をツリー構造で表示します。

```
pcs resource relations resource [--full]
```

--full オプションを指定すると、制約 ID およびリソースタイプを含む追加情報が表示されます。

以下の例では、C、D、および E の 3 つのリソースが設定されています。

```
# pcs constraint order start C then start D
Adding C D (kind: Mandatory) (Options: first-action=start then-action=start)
# pcs constraint order start D then start E
Adding D E (kind: Mandatory) (Options: first-action=start then-action=start)

# pcs resource relations C
C
`- order
  |  start C then start D
  `- D
    `- order
      |  start D then start E
      `- E
# pcs resource relations D
D
|- order
|  |  start C then start D
|  `- C
`- order
  |  start D then start E
  `- E
# pcs resource relations E
E
`- order
  |  start D then start E
  `- D
    `- order
      |  start C then start D
      `- C
```

以下の例では、A および B のリソースが 2 つ設定されています。リソース A および B はリソースグループ G の一部です。

```
# pcs resource relations A
A
`- outer resource
  `- G
    `- inner resource(s)
      |  members: A B
      `- B
# pcs resource relations B
B
`- outer resource
  `- G
    `- inner resource(s)
      |  members: A B
      `- A
# pcs resource relations G
G
`- inner resource(s)
```

```
|  members: A B
|- A
`- B
```

第16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。

16.1. PACEMAKER ルール

Pacemaker ルールを使用すると、設定をより動的に作成できます。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、日付などの様々な式だけでなく、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16.1 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**、**Unpromoted**、および **Promotiond** です。注意: **role="Promoted"** 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ません。どのアクティブ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。

16.1.1. 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16.2 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指定します。指定できる値: **string**、**integer**、**number**、**version**。デフォルト値は **string** です。
operation	実行する比較動作。設定できる値は以下のとおりです。 * **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、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** と同等にならない場合は True * **defined** - ノードに指定属性がある場合は 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、以下の表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16.3 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ロール。昇格可能なクローンに対する場所の制約のルール内でのみ有効です。

16.1.2. 時刻と日付ベースの式

日付の式は、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16.4 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。設定できる値は以下のとおりです。 * **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 のような比較を実行します。

16.1.3. 日付の詳細

日付の詳細は、時間に関係する cron のような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、デフォルトを 0 に設定するのではなく、無視されます。

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 (両時間を含む) の時間と一致します。ただし、**weekdays="1,2"** または **weekdays="1-2,5-6"** には複数の範囲が含まれるため、指定することはできません。

表16.5 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)

16.2. 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** を省略すると、デフォルトの INFINITY に設定されます。**resource-discovery** を省略すると、デフォルトの **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のリソース検出を制限 を参照してください。

基本的な場所の制約と同様に、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は、**score** を正または負の値にすることができます。正の値は prefers を示し、負の値は avoids を示します。

```
pcs constraint location rsc rule [resource-discovery=option] [role=promoted|unpromoted]
[score=score | score-attribute=attribute] expression
```

expression オプションは、以下のいずれかに設定できます。ここで、**duration_options** および **date_spec_options** は、日付の詳細 の表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|number|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 (hour) 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 日の金曜日が満月になると 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17章 クラスターリソースの管理

クラスターリソースの表示、変更、および管理に使用できるコマンドは複数あります。

17.1. 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リストを表示する場合は、次のコマンドを使用します。

```
pcs resource status
```

例えば、**VirtualIP** と言う名前のリソースと **WebSite** という名前のリソースでシステムを設定していた場合、**pcs resource status** 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
```

リソースに設定されている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
```

個々のリソースのステータスを表示するのに、次のコマンドを使用します。

```
pcs resource status resource_id
```

たとえば、システムが **VirtualIP** という名前のリソースで設定されていると、**pcs resource status VirtualIP** コマンドは以下の出力を表示します。

```
# pcs resource status VirtualIP
 VirtualIP      (ocf::heartbeat:IPaddr2):       Started
```

特定のノードで実行されているリソースのステータスを表示するのに、以下のコマンドを使用します。このコマンドを使用すると、クラスターとリモートノードの両方でリソースのステータスを表示できます。

```
pcs resource status node=node_id
```

たとえば、**node-01** が **VirtualIP** および **WebSite** という名前のリソースを実行している場合、**pcs resource status node=node-01** コマンドは以下の出力を表示します。

```
# pcs resource status node=node-01
 VirtualIP      (ocf::heartbeat:IPaddr2):       Started
 WebSite        (ocf::heartbeat:apache):        Started
```

17.2. PCS コマンドとしてのクラスターリソースのエクスポート

Red Hat Enterprise Linux 9.1 では、**pcs resource config** コマンドの **--output-format=cmd** オプションを使用して、別のシステムに設定済みのクラスターデバイスを再作成するのに使用できる **pcs** コマンドを表示できます。

以下のコマンドは、Red Hat 高可用性クラスター内のアクティブ/パッシブ Apache HTTP サーバー用に作成された 4 つのリソース (**LVM-activate** リソース、**Filesystem** リソース、**IPaddr2** リソース、および **Apache** リソース) を作成します。

```
# pcs resource create my_lvm ocf:heartbeat:LVM-activate vgname=my_vg
vg_access_mode=system_id --group apachegroup
# pcs resource create my_fs Filesystem device="/dev/my_vg/my_lv" directory="/var/www"
fstype="xfs" --group apachegroup
# pcs resource create VirtualIP IPaddr2 ip=198.51.100.3 cidr_netmask=24 --group apachegroup
# pcs resource create Website apache configfile="/etc/httpd/conf/httpd.conf"
statusurl="http://127.0.0.1/server-status" --group apachegroup
```

リソースを作成した後、次のコマンドを実行すると、別のシステムでそれらのリソースを再作成するために使用できる **pcs** コマンドが表示されます。

```
# pcs resource config --output-format=cmd
pcs resource create --no-default-ops --force -- my_lvm ocf:heartbeat:LVM-activate \
  vg_access_mode=system_id vgname=my_vg \
  op \
    monitor interval=30s id=my_lvm-monitor-interval-30s timeout=90s \
    start interval=0s id=my_lvm-start-interval-0s timeout=90s \
    stop interval=0s id=my_lvm-stop-interval-0s timeout=90s;
pcs resource create --no-default-ops --force -- my_fs ocf:heartbeat:Filesystem \
  device=/dev/my_vg/my_lv directory=/var/www fstype=xfs \
  op \
    monitor interval=20s id=my_fs-monitor-interval-20s timeout=40s \
    start interval=0s id=my_fs-start-interval-0s timeout=60s \
    stop interval=0s id=my_fs-stop-interval-0s timeout=60s;
pcs resource create --no-default-ops --force -- VirtualIP ocf:heartbeat:IPaddr2 \
  cidr_netmask=24 ip=198.51.100.3 \
  op \
    monitor interval=10s id=VirtualIP-monitor-interval-10s timeout=20s \
    start interval=0s id=VirtualIP-start-interval-0s timeout=20s \
    stop interval=0s id=VirtualIP-stop-interval-0s timeout=20s;
pcs resource create --no-default-ops --force -- Website ocf:heartbeat:apache \
  configfile=/etc/httpd/conf/httpd.conf statusurl=http://127.0.0.1/server-status \
  op \
    monitor interval=10s id=Website-monitor-interval-10s timeout=20s \
    start interval=0s id=Website-start-interval-0s timeout=40s \
    stop interval=0s id=Website-stop-interval-0s timeout=60s;
pcs resource group add apachegroup \
  my_lvm my_fs VirtualIP Website
```

pcs コマンドまたは 1 つの設定済みリソースのみ再作成するために使用できるコマンドを表示するには、そのリソースのリソース ID を指定します。

```
# pcs resource config VirtualIP --output-format=cmd
pcs resource create --no-default-ops --force -- VirtualIP ocf:heartbeat:IPaddr2 \
  cidr_netmask=24 ip=198.51.100.3 \
```

```
  op \
    monitor interval=10s id=VirtualIP-monitor-interval-10s timeout=20s \
    start interval=0s id=VirtualIP-start-interval-0s timeout=20s \
    stop interval=0s id=VirtualIP-stop-interval-0s timeout=20s
```

17.3. 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、**VirtualIP** リソースに設定したパラメーターの初期値、**ip** パラメーターの値を変更するコマンド、変更されたパラメーター値を示しています。

```
# 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
```

注記

pcs resource update コマンドを使用してリソースの動作を更新すると、特に呼び出しのないオプションはデフォルト値にリセットされます。

17.4. 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した場合は、**pcs status** コマンドでクラスターの状態を表示すると失敗メッセージが表示されます。障害の原因を解決しようとしたら、**pcs status** コマンドを再度実行してリソースの更新されたステータスを確認できます。また、**pcs resource failcount show --full** コマンドを使用してクラスターリソースの障害数を確認できます。

pcs resource cleanup コマンドを使用して、リソースの障害ステータスをクリアできます。**pcs resource cleanup** コマンドは、リソースのステータスと、リソースの **failcount** 値をリセットします。このコマンドは、リソースの操作履歴も削除し、現在の状態を再検出します。

次のコマンドは、resource_id で指定されたリソースのリソースステータスと **failcount** 値をリセットします。

```
pcs resource cleanup resource_id
```

resource_id を指定しないと、**pcs resource cleanup** コマンドは失敗数ですべてのリソースのリソースステータスと **failcount** 値をリセットします。

pcs resource cleanup resource_id コマンドの他に、**pcs resource refresh resource_id** コマンドを使用して、リソースのステータスをリセットし、リソースの操作履歴を消去することもできます。**pcs resource cleanup** コマンドと同様に、オプションを指定せずに **pcs resource refresh** コマンドを実行し、すべてのリソースのリソースステータスと **failcount** 値をリセットできます。

pcs resource cleanup コマンドおよび **pcs resource refresh** コマンドの両方が、リソースの操作履歴を消去し、リソースの現在の状態を再検出します。**pcs resource cleanup** コマンドは、クラスターの状態に示されているように、アクションが失敗したリソースでのみ動作しますが、**pcs resource refresh** コマンドは、現在の状態に関係なくリソース上で動作します。

17.5. 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様々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無効化、有効化、および禁止 に従ってリソースを有効、無効、および禁止にしてクラスターリソースの挙動を制御することもできます。

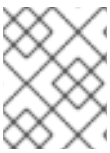
失敗した回数が、定義した値を超えると、新しいノードに移動し、外部接続が失われた時にリソースを移動するようにクラスターを設定できます。

17.5.1. 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** オプションを設定し、定義した回数だけ障害が発生した場合にリソースが新しいノードに移動されるように設定できます。このしきい値に一度到達すると、このノードでは、以下が行われるまで、障害が発生したリソースを実行できなくなります。

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

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な有限数として定義されます。0 にすると、**migration-threshold** 機能が無効になります。



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ようにリソースの移動を設定するのは同じではありません。

次の例では、**dummy_resource** 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 update 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) は、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、極めて重大となります。リソースの停止に失敗し STONITH

が有効になっていると、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フェンスが行われます。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、障害のタイムアウト後に再度停止が試行されます。

17.5.2. 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 アドレスによって指定されるリスト) 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、リソース場所制約を設定します。

以下の表には、**ping** リソースに設定できるプロパティーを紹介しています。

表17.1 ping リソースのプロパティー

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 (弱める) 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、クラスターでリソースがバウンスするのを防ぎます。
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。使用できる値には、解決可能な DNS ホスト名、IPv4 アドレス、および IPv6 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されるように、**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、既存のリソース **Webserver** に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 **gateway.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
```

17.6. 監視操作の無効化

定期的な監視を停止する最も簡単な方法は、監視を削除することです。ただし、一時的に無効にしたい場合もあります。このような場合は、操作の定義に **enabled="false"** を追加します。監視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。

pcs resource update コマンドを使用してリソースの動作を更新すると、特に呼び出しのないオプションはデフォルト値にリセットされます。たとえば、カスタムのタイムアウト値 600 を使用して監視操作を設定している場合に以下のコマンドを実行すると、タイムアウト値がデフォルト値の 20 にリセットされます (**pcs resource op default** コマンドを使用しても、デフォルト値を設定できます)。

```
# pcs resource update resourceXZY op monitor enabled=false
# pcs resource update resourceXZY op monitor enabled=true
```

このオプションの元の値 600 を維持するために、監視操作に戻す場合は、以下の例のように、その値を指定する必要があります。

```
# pcs resource update resourceXZY op monitor timeout=600 enabled=true
```

17.7. クラスターリソースタグの設定および管理

pcs を使用すると、クラスターリソースにタグを付けることができます。これにより、1 つのコマンドで、指定したリソースセットを有効化、無効化、マネージド化、または管理非対象化することができます。

17.7.1. カテゴリー別に管理するためのクラスターリソースのタグ付け

以下の手順では、リソースタグを使用して 2 つのリソースをタグ付けし、タグ付けされたリソースを無効にします。この例では、タグ付けする既存のリソースの名前は **d-01** および **d-02** です。

手順

1. **special-resources** という名前のタグ (**d-01** および **d-02**) を作成します。

```
[root@node-01]# pcs tag create special-resources d-01 d-02
```

2. リソースタグ設定を表示します。

```
[root@node-01]# pcs tag config
special-resources
  d-01
  d-02
```

3. **special-resources** タグが付けられた全リソース を無効にします。

```
[root@node-01]# pcs resource disable special-resources
```

4. リソースのステータスを表示して、**d-01** および **d-02** リソースが無効になっていることを確認します。

```
[root@node-01]# pcs resource
  * d-01        (ocf::pacemaker:Dummy): Stopped (disabled)
  * d-02        (ocf::pacemaker:Dummy): Stopped (disabled)
```

pcs resource disable コマンドに加え、**pcs resource enable**、**pcs resource manage** および **pcs resource unmanage** コマンドはタグ付けされたリソースの管理をサポートします。

リソースタグを作成したら、以下を実行します。

- **pcs tag delete** コマンドを使用して、リソースタグを削除できます。
- **pcs tag update** コマンドを使用して、既存のリソースタグのリソースタグ設定を変更できます。

17.7.2. タグ付けされたクラスターリソースの削除

pcs コマンドでは、タグ付けされたクラスターリソースを削除できません。タグ付けられたリソースを削除するには、以下の手順に従います。

手順

1. リソースタグを削除します。
   a. 以下のコマンドは、**special-resources** タグの付いたすべてのリソースからこのリソースタグを削除します。

```
[root@node-01]# pcs tag remove special-resources
[root@node-01]# pcs tag
 No tags defined
```

   b. 以下のコマンドは、リソース **d-01** からのみリソースタグ **special-resources** を削除します。

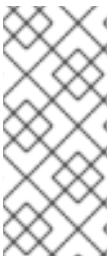
```
[root@node-01]# pcs tag update special-resources remove d-01
```

2. リソースを削除します。

```
[root@node-01]# pcs resource delete d-01
Attempting to stop: d-01... Stopped
```

第18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、ノードの分散のために、クローンとなるリソースを使用して、クラスター全体に分散させる IP リソースのインスタンスを複数設定できます。リソースエージェントが対応していれば、任意のリソースのクローンを作成できます。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

同時に複数のノードでアクティブにできるリソースのみがクローンに適しています。たとえば、共有メモリーデバイスから **ext4** などの非クラスター化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の操作には適していません。

18.1. クローンリソースの作成および削除

リソースと、そのリソースのクローンを同時に作成できます。

以下のコマンドを実行して、リソースと、そのリソースのクローンを作成します。

pcs resource create **resource_id** [**standard**:[**provider**:]]**type** [**resource options**] [meta **resource meta options**] clone [**clone_id**] [**clone options**]

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

デフォルトでは、クローンの名前は **resource_id-clone** となります。**clone_id** オプションの値を指定して、クローンのカスタム名を設定できます。

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を作成する場合は、次のコマンドを実行します。

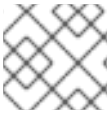
pcs resource clone **resource_id** | **group_id** [**clone_id**][**clone options**]...

pcs resource clone **resource_id** | **group_id** [**clone options**]...

デフォルトでは、クローンの名前は **resource_id-clone** または **group_name-clone** です。**clone_id** オプションの値を指定して、クローンのカスタム名を設定できます。

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、クローンのデフォルト名は、リソース名に **-clone** を付けた名前になります。次のコマンドは、タイプが **apache** のリソース **webfarm** と、そのクローンとなるリソース **webfarm-clone** を作成します。

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

注記

あるリソースまたはリソースグループのクローンを、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、依存されているクローンが同じノードで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るようになります。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 に依存していると、ノードがクラスターから離れてから戻ってきてから、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、全ノードで再起動します。これは、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場合は、そのリソースの全インスタンスが、そのリソースが依存してい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 | clone_id | group_name
```

以下の表には、クローンのリソースに指定できるオプションを示しています。

表18.1 リソースのクローンオプション

フィールド	説明
priority, target-role, is-managed	リソースのメタオプションの設定 の表リソースのメタオプションで説明されているように、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する時に、前もって、およびアクションが成功した時に、他のコピーに通知します。使用できる値は **false** および **true** です。デフォルト値は **false** です。

フィールド	説明
globally-unique	クローンの各コピーで異なる機能を実行させるかどうか。使用できる値は **false** および **true** です。 このオプションの値を **false** にすると、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を **true** にすると、任意のマシンで実行中のクローンのコピーが、別のインスタンスが別のノードまたは同じノードで実行されているかに関係なく、そのインスタンスと同等ではありません。**clone-node-max** の値が1より大きい場合にはデフォルト値が **true** になり、小さい場合は **false** がデフォルト値になります。
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	(クローン間の) 順序制約の動作を変更して、(2番目のクローンの全インスタンスが終了するまで待機する代わりに) 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できるようにします。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	このフィールドに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、元のクローンの後に順序付けされたクローンは、元のクローンに指定され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現するために、クローンは、デフォルトでわずかに固定 (sticky) されています。これは、クローンが実行しているノードにとどまることをわずかに優先することを示します。**resource-stickiness** の値を指定しないと、クローンが使用する値は1となります。値を小さくすることで他のリソースのスコア計算への阻害を最小限に抑えながら、Pacemaker によるクラスター内の不要なコピーの移動を阻止することができます。**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。

18.2. クローンリソース制約の表示

ほとんどの場合、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1つです。ただし、リソースクローンの **clone-max** には、そのクラスター内のノード合計より小さい数を設定できます。この場合は、リソースの場所の制約を使用して、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、クローンの ID を使用する必要があることを除いて、通常のリソースの制約と同じように記述されます。

次のコマンドは、クラスターがリソースのクローン **webfarm-clone** を **node1** に優先的に割り当てる場所の制約を作成します。

```
# 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、**interleave** クローンオプションをデフォルトの **false** のままにしているため、起動する必要がある **webfarm-clone** のすべてのインスタンスが起動するまで、**webfarm-stats** のインスタンスは起動しません。**webfarm-clone** のコピーを 1 つも起動できない場合にのみ、**webfarm-stats** が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) とクローンのコロケーションは、リソースを、クローンのアクティブコピーを持つ任意のマシンで実行できることを意味します。クラスターは、クローンが実行している場所と、リソース自体の場所の優先度に基づいてコピーを選択します。

クローン間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
```

18.3. 昇格可能なクローンリソース

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。インスタンスを 2 つの動作モード (**promoted** および **unpromoted**) のいずれかに設定できます。モードの名前には特別な意味はありませんが、インスタンスの起動時に、**Unpromoted** 状態で起動する必要があるという制限があります。注意: Promoted ロール名および Unpromoted ロール名は、以前の RHEL リリースの Master ロールおよび Slave Pacemaker ロールと機能的に同等です。

18.3.1. 昇格可能なクローンリソースの作成

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_id] [clone options]
```

デフォルトでは、昇格可能なクローンの名前は **resource_id-clone** となります。

clone_id オプションの値を指定して、クローンのカスタム名を設定できます。

また、次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。

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

デフォルトでは、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。

clone_id オプションの値を指定して、クローンのカスタム名を設定できます。

以下の表には、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示しています。

表18.2 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。

18.3.2. 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。

コロケーション制約を作成できます。この制約は、リソースがプロモートロールまたはアンプロモートロールのどちらで動作しているかを指定します。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ます。

```
pcs constraint colocation add [promoted|unpromoted] source_resource with [promoted|unpromoted] target_resource [score] [options]
```

コロケーションの制約の詳細は、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に、リソースに指定できるアクションに、リソースのアンプロモートロールからプロモートへのロールの昇格を指定する **promote** があります。また、**demote** の動作を指定して、リソースのプロモートロールからアンプロモートロールへの降格を指定することもできます。

順序制約を設定するコマンドは次のようになります。

```
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。

18.4. 障害時の昇格リソースの降格

プロモート または **監視** アクションが失敗した場合や、リソースを実行しているパーティションでクォーラムを失った場合に、リソースは降格されますが、完全に停止されないように、プロモート可能なリソースを設定できます。これにより、リソースを完全に停止したときに、手動で介入する必要がなくなります。

- 昇格可能なリソースを **promote** アクションが失敗したときに降格するように設定するには、以下の例のように **on-fail** 操作メタオプションを **demote** に設定します。

  ```
  # pcs resource op add my-rsc promote on-fail="demote"
  ```

- **monitor** アクションが失敗したときに昇格可能なリソースを降格するように設定するには、**interval** をゼロ以外の値に設定し、**on-fail** 操作メタオプションを **demote** に設定して、**role** を **Promoted** に設定します。

```
# pcs resource op add my-rsc monitor interval="10s" on-fail="demote"
role="Promoted"
```

- クラスターパーティションでクォーラムが失われると、昇格されたリソースが降格されますが、実行され続け、他のすべてのリソースが停止されるようにクラスターを設定するには、**no-quorum-policy** クラスタープロパティーを **demote** に設定します。

操作の **on-fail** メタ属性を **demote** に設定しても、リソースの昇格を決定する方法には影響しません。影響を受けるノードのプロモーションスコアが引き続き最高となっている場合は、再度昇格するように選択されます。

第19章 クラスターノードの管理

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に使用できる、さまざまな **pcs** コマンドがあります。

19.1. 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と、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、ローカルノードのクラスターサービスを強制的に停止できます。このコマンドは、**kill -9** コマンドを実行します。

```
pcs cluster kill
```

19.2. 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 つ以上のノードで起動時にクラスターサービスが実行されるように設定されます。

ノードがフェンスされた後にクラスターに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を有効にしていないと、クラスターサービスを手動で開始する前に、管理者が問題を調査できます。これにより、たとえばハードウェアに問題があるノードで再度問題が発生する可能性がある場合は、クラスターに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] [...]
```

指定した 1 つまたは複数のノードの起動時に、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。

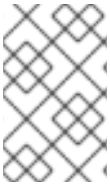
- **--all** オプションを指定すると、全ノードでクラスターサービスが無効になります。
- ノードを指定しないと、ローカルノードでのみクラスターサービスが無効になります。

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

19.3. クラスターノードの追加

次の手順で既存のクラスターに新しいノードを追加します。

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。



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。

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

手順

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、新しいノードにも手動でインストールする必要があります。

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

   クラスターパッケージに加えて、既存のクラスターノードにインストールした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および設定する必要があります。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確認する **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、次のコマンドを実行して、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 --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で、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 **hacluster** ユーザーを認証します。

```
[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) に同期させます。

```
[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して有効にします。

```
[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対して、フェンシングデバイスを設定してテストします。

19.4. 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、クラスター内の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** からそのノードを削除します。

```
pcs cluster node remove node
```

19.5. リンクが複数あるクラスターへのノードの追加

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。

以下の例では、ノード **rh80-node3** をクラスターに追加し、1 番目のリンクに IP アドレス 192.168.122.203 を、2 番目のリンクに 192.168.123.203 を指定します。

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

19.6. 既存のクラスターへのリンクの追加および修正

ほとんどの場合に、クラスターの再起動なしに既存クラスターのリンクを追加または変更できます。

19.6.1. 既存クラスターへのリンクの追加および削除

実行中のクラスターに新しいリンクを追加するには、**pcs cluster link add** コマンドを使用します。

- リンクの追加時に、各ノードのアドレスを指定する必要があります。
- リンクの追加および削除は、**knet** トランスポートプロトコルを使用している場合に限り可能です。
- クラスター内で常に 1 つはリンクを定義する必要があります。
- クラスター内のリンクの最大数は 8 で、指定番号は 0-7 です。3、6、7 のみを指定するなど、リンクはどれでも定義できます。

- リンク番号を指定せずにリンクを追加すると、**pcs** は利用可能なリンクで番号が一番小さいものを使用します。
- 現在設定されているリンクのリンク番号は、**corosync.conf** ファイルに含まれます。**corosync.conf** ファイルを表示するには、**pcs cluster corosync** コマンドまたは **pcs cluster config show** コマンドを実行します。

以下のコマンドは、リンク番号 5 を 3 つのノードクラスターに追加します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.11 node2=10.0.5.12 node3=10.0.5.31
options linknumber=5
```

既存のリンクを削除するには、**pcs cluster link delete** コマンドまたは **pcs cluster link remove** コマンドを使用します。以下のコマンドのいずれかを実行すると、クラスターからリンク番号 5 が削除されます。

```
[root@node1 ~] # pcs cluster link delete 5

[root@node1 ~] # pcs cluster link remove 5
```

19.6.2. リンクが複数あるクラスター内のリンクの変更

クラスターに複数のリンクがあり、そのいずれかを変更する場合は、以下の手順を実行します。

手順

1. 変更するリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 2
```

2. アドレスとオプションを更新して、クラスターにリンクを追加し直します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.11 node2=10.0.5.12
node3=10.0.5.31 options linknumber=2
```

19.6.3. 単一リンクを使用したクラスターのリンクアドレスの変更

クラスターで 1 つのみリンクを使用し、別のアドレスを使用するようにリンクを変更する必要がある場合は、以下の手順を実行します。この例では、元のリンクはリンク 1 です。

1. 新しいアドレスおよびオプションを指定して新規リンクを追加します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.11 node2=10.0.5.12
node3=10.0.5.31 options linknumber=2
```

2. 元のリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 1
```

クラスターへのリンクの追加時に、現在使用中のアドレスは指定できないことに注意してください。たとえば、リンクが 1 つある 2 ノードクラスターがあり、ノード 1 つだけでアドレスを変更する場合に上記の手順を使用して、新規アドレスと既存のアドレスを指定するリンクを新たに追加できません。代わ

りに、以下の例のように、既存のリンクを削除し、アドレスを更新したリンクを追加しなおすことができます。

この例では、以下のように設定されています。

- 既存クラスターのリンクはリンク 1 で、ノード 1 に 10.0.5.11 のアドレスを使用し、ノード 2 に 10.0.5.12 アドレスを使用します。
- ノード 2 のアドレスを 10.0.5.31 に変更します。

手順

リンクが 1 つである 2 ノードクラスターのアドレスのいずれかのみを更新するには、以下の手順に従います。

1. 現在使用されていないアドレスを使用して、既存のクラスターに新しい一時的なリンクを追加します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.13 node2=10.0.5.14 options
linknumber=2
```

2. 元のリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 1
```

3. 変更後の新しいリンクを追加します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.11 node2=10.0.5.31 options
linknumber=1
```

4. 作成した一時的なリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 2
```

19.6.4. リンクが 1 つのクラスター内のリンクオプションの変更

クラスターで使用されているリンクが 1 つのみで、そのリンクのオプションを変更しつつも、使用するアドレスを変更しない場合には、一時的なリンクを追加してからリンクを削除し、リンクを別のものに更新できます。

この例では、以下のように設定されています。

- 既存クラスターのリンクはリンク 1 で、ノード 1 に 10.0.5.11 のアドレスを使用し、ノード 2 に 10.0.5.12 アドレスを使用します。
- リンクオプション **link_priority** を 11 に変更します。

手順

次の手順で、1 つのリンクを持つクラスターでリンクオプションを変更します。

1. 現在使用されていないアドレスを使用して、既存のクラスターに新しい一時的なリンクを追加します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.13 node2=10.0.5.14 options
linknumber=2
```

2. 元のリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 1
```

3. 元のリンクのオプションを更新して追加し直します。

```
[root@node1 ~] # pcs cluster link add node1=10.0.5.11 node2=10.0.5.12 options
linknumber=1 link_priority=11
```

4. 一時的なリンクを削除します。

```
[root@node1 ~] # pcs cluster link remove 2
```

19.6.5. 新しいリンクの追加時にリンクの変更はできません。

設定で新しいリンクを追加することができない場合や、既存のリンクを 1 つ変更することが唯一のオプションである場合は、以下の手順を使用します。これにより、クラスターをシャットダウンする必要があります。

手順

以下の例では、クラスター内のリンク番号 1 を更新し、リンクの **link_priority** オプションを 11 に設定します。

1. クラスターのクラスターサービスを停止します。

```
[root@node1 ~] # pcs cluster stop --all
```

2. リンクアドレスとオプションを更新します。
pcs cluster link update コマンドでは、すべてのノードアドレスとオプションを指定する必要はありません。代わりに、変更するアドレスのみを指定できます。この例では、**node1** および **node3** のアドレスを変更し、**link_priority** オプションのみを変更します。

```
[root@node1 ~] # pcs cluster link update 1 node1=10.0.5.11 node3=10.0.5.31 options
link_priority=11
```

オプションを削除するには、 **option=** 形式で Null 値にオプションを設定します。

3. クラスターを再起動します。

```
[root@node1 ~] # pcs cluster start --all
```

19.7. ノードのヘルスストラテジーの設定

ノードは、そのクラスターメンバーシップを維持するためには十分に機能していても、別の側面では正常に機能しておらず、リソースにとって適切ではないロケーションになることがあります。たとえば、ディスクドライブが SMART エラーを報告していたり、CPU の負荷が高くなっている場合などがそうです。RHEL 9.1 では、Pacemaker のノードヘルスストラテジーを使用して、自動的にリソースを正常でないノードから移動できます。

次のヘルスノードリソースエージェントを使用して、ノードのヘルスを監視できます。このエージェントは、CPU とディスクのステータスに基づいてノードの属性を設定します。

- **ocf:pacemaker:HealthCPU**: CPU のアイドリングを監視
- **ocf:pacemaker:HealthIOWait**: CPU I/O 待機を監視
- **ocf:pacemaker:HealthSMART**: ディスクドライブの SMART ステータスを監視
- **ocf:pacemaker:SysInfo**: ローカルシステム情報を使用してさまざまなノード属性を設定し、ディスク領域の使用状況を監視するヘルスエージェントとしても機能

さらに、すべてのリソースエージェントがヘルスノードストラテジーの定義に使用できるノード属性を提供する可能性があります。

手順

次の手順では、CPU I/O 待機が 15% を超えるノードからリソースを移動するクラスターのヘルスノードストラテジーを設定します。

1. **health-node-strategy** クラスタープロパティーを設定して、Pacemaker がノードヘルスの変化に応答する方法を定義します。

```
# pcs property set node-health-strategy=migrate-on-red
```

2. ヘルスノードリソースエージェントを使用するクラスターリソースのクローンを作成し、**allow-unhealthy-nodes** リソースメタオプションを設定して、ノードのヘルスが回復したかどうかをクラスターが検出してリソースをノードに戻すかどうかを定義します。すべてのノードのヘルスを継続的にチェックするには、定期的な監視アクションを使用してこのリソースを設定します。
この例では、**HealthIOWait** リソースエージェントを作成して CPU I/O 待機を監視し、ノードからリソースを移動するための制限を 15% に設定します。このコマンドは、**allow-unhealthy-nodes** リソースメタオプションを **true** に設定し、繰り返しの監視間隔を 10 秒に設定します。

```
# pcs resource create io-monitor ocf:pacemaker:HealthIOWait red_limit=15 op monitor interval=10s meta allow-unhealthy-nodes=true clone
```

19.8. 多数のリソースを使用した大規模なクラスターの設定

デプロイするクラスターにノードとリソースが多数含まれる場合に、クラスターの以下のパラメーターのデフォルト値を変更する必要がある場合があります。

cluster-ipc-limit クラスタープロパティー

cluster-ipc-limit クラスタープロパティーは、あるクラスターデーモンが別のクラスターデーモンを切断するまでに対応できる IPC メッセージバックログの最大数になります。多数のリソースを消去するか、大規模なクラスターで同時にリソースを変更すると、多数の CIB 更新が一度に行われます。これが原因で、CIB イベントキューのしきい値に到達するまでに Pacemaker サービスで全設定の更新を処理する時間がない場合には、低速なクライアントがエビクトされる可能性がありました。
大規模なクラスターで使用するための **cluster-ipc-limit** の推奨値は、クラスターのリソース数にノード数を乗算した値です。この値は、クラスターデーモン PID の Evicting client メッセージがログに表示されると増える可能性があります。

pcs property set コマンドを使用して、**cluster-ipc-limit** の値をデフォルト値 500 から増やすことができます。たとえば、リソースが 200 個ある 10 ノードクラスターの場合には、以下のコマンドを使用して **cluster-ipc-limit** の値を 2000 に設定できます。

```
# pcs property set cluster-ipc-limit=2000
```

PCMK_ipc_buffer Pacemaker パラメーター

非常に大規模なデプロイメントでは、内部 Pacemaker メッセージがメッセージバッファーのサイズを超える可能性があります。バッファーサイズを超えると、以下の形式のシステムログにメッセージが表示されます。

```
Compressed message exceeds X% of configured IPC limit (X bytes); consider setting
PCMK_ipc_buffer to X or higher
```

このメッセージが表示されると、各ノードの **/etc/sysconfig/pacemaker** 設定ファイルで **PCMK_ipc_buffer** の値を増やしてください。たとえば、**PCMK_ipc_buffer** の値をデフォルト値から 13396332 バイトに増やすには、以下のようにクラスター内の各ノードの **/etc/sysconfig/pacemaker** ファイルのアンコメントされている **PCMK_ipc_buffer** フィールドを変更します。

```
PCMK_ipc_buffer=13396332
```

この変更を適用するには、以下のコマンドを実行します。

```
# systemctl restart pacemaker
```

第20章 PACEMAKER クラスターにユーザーパーミッションの設定

hacluster ユーザー以外の特定のユーザーに、Pacemaker クラスターを管理するパーミッションを付与できます。個々のユーザーに付与できるパーミッションには、以下の 2 つのセットがあります。

- 個々のユーザーが Web UI を介してクラスターを管理し、ネットワーク経由でノードに接続する **pcs** コマンドを実行できるパーミッション。ネットワーク経由でノードに接続するコマンドには、クラスターを設定するコマンド、またはクラスターからノードを追加または削除するためのコマンドが含まれます。

- クラスター設定の読み取り専用アクセスまたは読み書きアクセスを許可するためのローカルユーザーのパーミッションです。ネットワーク経由で接続する必要のないコマンドには、リソースの作成や制約の設定など、クラスター設定を編集するコマンドが含まれます。

両方のパーミッションセットが割り当てられている状況では、ネットワーク経由で接続するコマンドのパーミッションが最初に適用され、次にローカルノードのクラスター設定を編集するパーミッションが適用されます。ほとんどの **pcs** コマンドでは、ネットワークアクセスは必要ありません。その場合は、ネットワークパーミッションが適用されません。

20.1. ネットワーク上でノードにアクセスするためのパーミッションの設定

特定のユーザーに、Web UI でクラスターを管理し、ネットワーク経由でノードに接続する **pcs** コマンドを実行するパーミッションを付与するには、パーミッションを付与するユーザーをグループ **haclient** に追加します。これはクラスター内のすべてのノードで行う必要があります。

20.2. ACL でローカルパーミッションの設定

pcs acl コマンドを使用して、ローカルユーザーにアクセス制御リスト (ACL) を使用してクラスター設定への読み取り専用アクセスまたは読み取り/書き込みアクセスを許可するパーミッションを設定できます。

デフォルトでは、ACL は有効になっていません。ACL が有効になっていないと、すべてのノードの **haclient** グループのメンバーである任意のユーザーには、クラスター設定へのローカルの読み取り/書き込みに関する完全なパーミッションが付与されます。一方、**haclient** グループのメンバーではないユーザーには、パーミッションが付与されません。ただし、ACL が有効になっている場合は、**haclient** グループのメンバーであっても、ACL からそのユーザーに付与されたものにしかアクセスできません。ACL が有効になっている場合でも、root および **hacluster** ユーザーアカウントはクラスター設定に常にフルアクセスできます。

ローカルユーザーのパーミッションを設定するには、以下の 2 つの手順を実行します。

1. **pcs acl role create...** コマンドを実行して、そのロールのパーミッションを定義する **ロール** を作成します。

2. **pcs acl user create** コマンドで、作成したロールをユーザーに割り当てます。複数のロールを同じユーザーに割り当てると、**拒否** パーミッションが優先されてから、**書き込み**、**読み取り** の順に適用されます。

手順

以下の例では、**rouser** という名前のローカルユーザーに、クラスター設定に対する読み取り専用アクセスを提供します。設定の特定の部分のみにアクセスを制限することもできます。

警告

この手順を root として実行するか、すべての設定の更新を作業ファイルに保存して、完了したらアクティブな CIB にプッシュできるようにすることが重要です。それ以外の場合は、それ以上変更を加えられないようにすることができます。作業ファイルに設定の更新を保存する方法は、作業ファイルへの設定変更の保存 を参照してください。

1. この手順では、**rouser** ユーザーがローカルシステムに存在し、**rouser** ユーザーが **haclient** グループのメンバーであることが必要です。

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

2. **pcs acl enable** コマンドを使用して、Pacemaker ACL を有効にします。

```
# pcs acl enable
```

3. cib に対して読み取り専用のパーミッションが付与されている **read-only** という名前のロールを作成します。

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

4. pcs ACL システムで **rouser** ユーザーを作成し、そのユーザーに **read-only** ロールを割り当てます。

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

5. 現在の ACL を表示します。

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

6. **rouser** が **pcs** コマンドを実行する各ノードで **rouser** としてログインし、ローカルの **pcsd** サービスに対して認証します。これは、ACL ユーザーとして **pcs status** などの特定の **pcs** コマンドを実行する場合に必要になります。

```
[rouser ~]$ pcs client local-auth
```

第21章 リソースの監視操作

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。リソースの監視動作を指定しないと、デフォルトでは、**pcs** コマンドにより監視動作が作成されします。監視の間隔はリソースエージェントにより決定します。リソースエージェントでデフォルトの監視間隔が提供されない場合は、pcs コマンドにより 60 秒間隔の監視動作が作成されます。

以下の表には、リソースの監視動作のプロパティーをまとめています。

表21.1 動作のプロパティー

フィールド	説明
id	動作の一意の名前。システムは、操作を設定する際に、これを割り当てます。
name	実行する動作。一般的な値は、**monitor**、**start**、**stop** です。
interval	値をゼロ以外に設定すると、この周波数で繰り返される反復操作 (秒単位) が作成されます。ゼロ以外の値は、アクション **名** が **monitor** に設定されている場合に限り有効になります。監視の反復アクションは、リソースの起動が完了するとすぐに実行し、その後の監視アクションの開始は、前の監視アクションが完了した時点でスケジュールされます。たとえば、**interval=20s** の監視アクションを 01:00:00 に実行すると、次の監視アクションは 01:00:20 ではなく、最初の監視アクションが完了してから 20 秒後に発生します。 この値を、デフォルト値であるゼロに設定すると、このパラメーターで、クラスターが作成した操作に使用する値を指定できます。たとえば、**interval** をゼロに設定し、操作の **name** を **start** に設定し、**timeout** 値を 40 に設定すると、Pacemaker がこのリソースを開始する場合に 40 秒のタイムアウトを使用します。**monitor** 操作の間隔をゼロに設定した場合は、システムの起動時に Pacemaker が行うプローブの **timeout**/**on-fail**/**enabled** を設定して、デフォルトが望ましくない場合に、すべてのリソースの現在のステータスを取得できます。
timeout	このパラメーターで設定された時間内に操作が完了しないと、操作を中止し、失敗したと見なします。デフォルト値は、**pcs resource op defaults** コマンドで設定した場合は **timeout** 値、設定していない場合は 20 秒 です。システムで操作 (**start**、**stop**、**monitor** など) の実行に許可されている時間よりも長い時間が必要なリソースがシステムに含まれている場合は、原因を調査して、実行時間が長いと予想される場合は、この値を増やすことができます。 **timeout** 値はいかなる遅延でもありません。また、タイムアウト期間が完了する前に操作が戻ると、クラスターはタイムアウト期間が終わる前に待機を終了します。

フィールド	説明
on-fail	この動作が失敗した場合に実行する動作。設定できる値は以下のとおりです。 * **ignore** - リソースが失敗していなかったように振る舞う * **block** - リソースでこれ以上、一切の操作を行わない * **stop** - リソースを停止して別の場所で起動しない * **restart** - リソースを停止して、(おそらく別の場所で) 再起動する * **fence** - 失敗したリソースがあるノードを STONITH する * **standby** - 失敗したリソースがあるノード上の **すべて** のリソースを移行する * **demote** - リソースの **promote** アクションに失敗した場合にリソースは降格されますが、完全に停止されません。リソースの **monitor** アクションが失敗した場合、**interval** がゼロ以外の値に設定され、**role** が **Promoted** に設定されている場合、リソースは降格されますが、完全に停止されません。 STONITH が有効な場合、**stop** 動作のデフォルトは **fence** になります。そうでない場合は **block** となります。その他のすべての操作は、デフォルトで **restart** です。
enabled	**false** の場合、操作は存在しないものとして処理されます。使用できる値は **true** または **false** です。

21.1. リソースの監視動作の設定

次のコマンドでリソースを作成すると、モニタリング操作を設定できます。

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

たとえば、次のコマンドは、監視操作付きの **IPaddr2** リソースを作成します。新しいリソースには **VirtualIP** という名前が付けられ、**eth2** で IP アドレス 192.168.0.99、ネットマスク 24 になります。監視操作は、30 秒ごとに実施されます。

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

また、次のコマンドで既存のリソースに監視操作を追加することもできます。

> pcs resource op add **resource_id operation_action** [**operation_properties**]

設定されているリソース操作を削除する場合は、次のコマンドを使用します。

> pcs resource op remove **resource_id operation_name operation_properties**

注記

操作プロパティーを正しく指定して、既存の操作を適切に削除する必要があります。

監視オプションの値を変更する場合は、リソースを更新します。たとえば、以下のコマンドで **VirtualIP** を作成できます。

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

デフォルトでは、次の操作が作成されます。

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

stop の timeout 操作を変更するには、以下のコマンドを実行します。

```
# pcs resource update VirtualIP op stop interval=0s timeout=40s

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

21.2. グローバルリソース操作のデフォルトの設定

pcs resource op defaults update コマンドを使用すると、すべてのリソースでリソースオペレーションのデフォルト値を変更できます。

たとえば、次のコマンドは、すべての監視操作に対して、**timeout** 値のグローバルデフォルトを 240 秒に設定します。

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

以前のリリースのすべてのリソース操作のデフォルトを設定する元の **pcs resource defaults name=value** コマンドは、複数のデフォルトが設定されない限りサポートされます。ただし、**pcs resource op defaults update** が、コマンドの推奨されるバージョンになりました。

21.2.1. リソース固有の操作の値の上書き

クラスターリソース定義でオプションが指定されていない場合に限り、クラスターリソースがグローバルデフォルトを使用することに注意してください。デフォルトでは、リソースエージェントがすべての操作の **timeout** オプションを定義します。グローバル操作のタイムアウト値を有効にするには、**timeout** オプションを明示的に指定せずにクラスターリソースを作成するか、次のコマンドのように、クラスターリソースを更新して **timeout** オプションを削除する必要があります。

```
# pcs resource update VirtualIP op monitor interval=10s
```

たとえば、すべての監視操作に、グローバルデフォルトの **timeout** 値 240 秒を設定し、クラスターリソース **VirtualIP** を更新して、**monitor** 操作のタイムアウト値を削除すると、リソース **VirtualIP** に

は、**start**、**stop**、および **monitor** の操作のタイムアウト値が、それぞれ 20 秒、40 秒、240 秒に設定されます。タイムアウト操作のグローバルなデフォルト値は、ここでは **monitor** にのみ適用されます。ここでは、前のコマンドにより、デフォルトの **timeout** オプションが削除されています。

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

21.2.2. リソースセットのリソース操作のデフォルト値の変更

pcs resource op defaults set create コマンドを使用して、複数のリソース操作のデフォルトセットを作成できます。これにより、**resource** 式および操作式を含むルールを指定できます。Pacemaker が対応しているルール式はすべて使用できます。

このコンマでは、特定タイプの全リソースにデフォルトのリソース操作値を設定できます。たとえば、バンドルが使用されている場合に Pacemaker が作成した暗黙的な **podman** リソースを設定できるようになりました。

以下のコマンドは、すべての **podman** リソースの全操作に、デフォルトのタイムアウト値 90s を設定します。この例では、**::podman** は、クラスやプロバイダーは任意でタイプが **podman** を指定します。

リソース操作のデフォルトセット名を指定する **id** オプションは必須ではありません。このオプションを設定すると、**pcs** が自動的に ID を生成します。この値を設定すると、より分かりやすい名前に設定できます。

```
# pcs resource op defaults set create id=podman-timeout meta timeout=90s rule resource
::podman
```

以下のコマンドは、すべてのリソースの **stop** 操作に、デフォルトのタイムアウト値 120s を設定します。

```
# pcs resource op defaults set create id=stop-timeout meta timeout=120s rule op stop
```

特定タイプの全リソースの、特定の操作にデフォルトのタイムアウト値を設定できます。以下の例は、すべての **podman** リソースの **stop** 操作に、デフォルトのタイムアウト値 120 を設定します。

```
# pcs resource op defaults set create id=podman-stop-timeout meta timeout=120s rule
resource ::podman and op stop
```

21.2.3. 現在設定されているリソース操作のデフォルト値の表示

pcs resource op defaults コマンドは、指定したルールなど、現在設定されているリソース操作のデフォルト値のリストを表示します。

以下のコマンドは、全 **podman** リソースの全操作に、デフォルトのタイムアウト値 90s が設定されており、リソース操作のデフォルトセットの ID が **podman-timeout** として設定されているクラスターのデフォルトの操作値を表示します。

```
# pcs resource op defaults
Meta Attrs: podman-timeout
```

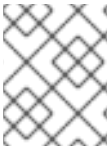
```
  timeout=90s
  Rule: boolean-op=and score=INFINITY
    Expression: resource ::podman
```

以下のコマンドは、全 **podman** リソースの **stop** 操作に、デフォルトのタイムアウト値 120s が設定されており、リソース操作のデフォルトセットの ID が **podman-stop-timeout** として設定されているクラスターのデフォルトの操作値を表示します。

```
# pcs resource op defaults]
Meta Attrs: podman-stop-timeout
  timeout=120s
  Rule: boolean-op=and score=INFINITY
    Expression: resource ::podman
    Expression: op stop
```

21.3. 複数の監視操作の設定

リソースエージェントが対応する範囲で、1 つのリソースに複数の監視操作を設定できます。これにより、1 分ごとに表面的なヘルスチェックを行い、徐々に頻度を上げてより正確なチェックを行うこともできます。



注記

複数の監視操作を設定する場合は、2 種類の操作が同じ間隔で実行されないように注意してください。

様々なレベルで行う詳細なヘルスチェックに対応する追加の監視操作をリソースに設定する場合は、**OCF_CHECK_LEVEL=n** オプションを追加します。

たとえば、以下のように **IPaddr2** リソースを設定すると、デフォルトでは 10 秒間隔でタイムアウト値が 20 秒の監視操作が作成されます。

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

仮想 IP が、深さ 10 の様々なチェックに対応する場合は、次のコマンドを実行すると、Pacemaker は、通常の 10 秒間隔の仮想 IP チェックに加えて、60 秒ごとに高度な監視チェックを実行します。なお、上述のとおり、追加の監視操作は 10 秒間隔にしないようにしてください。

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

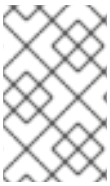
第22章 PACEMAKER クラスターのプロパティー

クラスターのプロパティーは、クラスター操作中に生じる可能性のある各状況でのクラスターの動作を制御します。

22.1. クラスタープロパティーおよびオプションの要約

以下の表には、Pacemaker クラスターのプロパティーのデフォルト値や、設定可能な値などをまとめています。

フェンス動作を決定するクラスタープロパティーがあります。これらのプロパティーの詳細は、フェンシングデバイスの一般的なプロパティー の表フェンスの動作を決定するクラスタープロパティーを参照してください。



注記

この表に記載しているプロパティー以外にも、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ーがあります。このようなプロパティーでは、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22.1 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と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)	クラスターが、ノードで並行に実行することが許可されている移行ジョブの数。

オプション	デフォルト	説明
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。設定できる値は以下のとおりです。 * ignore - 全リソースの管理を続行する * freeze - リソース管理は継続するが、影響を受けるパーティションに含まれない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フェンスする * demote - クラスターパーティションがクォーラムを失うと、プロモートされたリソースを降格し、その他のリソースを停止する
symmetric-cluster	true	リソースを、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。
cluster-delay	60s	(アクションの実行を除く) ネットワーク上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ーノードの速度と負荷によって異なります。
dc-deadtime	20s	起動時に他のノードからの応答を待つ時間。正しい値は、ネットワークの速度と負荷、および使用するスイッチの種類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。

オプション	デフォルト	説明
start-failure-is-fatal	true	特定のノードでリソースの起動に失敗した場合に、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。この理由により、**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、移行しきい値を低く設定することで軽減できます。これにより、何度も失敗してもその他のアクションを続行できます。
pe-error-series-max	-1 (すべて)	ERROR となるスケジューラー入力を保存する数。問題を報告する場合に使用されます。
pe-warn-series-max	-1 (すべて)	WARNING となるスケジューラー入力を保存する数。問題を報告する場合に使用されます。
pe-input-series-max	-1 (すべて)	normal となるスケジューラー入力を保存する数。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。

オプション	デフォルト	説明
cluster-recheck-interval	15 分	Pacemaker は基本的にイベント駆動型で、失敗によるタイムアウトやほとんどの時間ベースのルールについてクラスターを再確認するタイミングを予め把握します。Pacemaker は、このプロパティーで指定された非アクティブ期間の後にもクラスターを再確認します。このクラスターの再確認には 2 つの目的があります。**date-spec** のルールがこの頻繁で必ず確認されるようにすることと、一部のタイプのスケジューラーバグについてフェイルセーフとして機能することです。0 を値として指定すると、このポーリングが無効になります。正の値は時間間隔を示します。
maintenance-mode	false	メンテナンスモードでは、クラスターが干渉されないモードになり、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、開始します。
shutdown-escalation	20min	正常にシャットダウンして終了を試みるのをやめる時間。高度な使用のみ。
stop-all-resources	false	クラスターがすべてのリソースを停止します。
enable-acl	false	クラスターが、**pcs acl** コマンドで設定したアクセス制御リストを使用できるかどうかを示します。
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、クラスターが使用率属性を考慮にいれるかどうかと、どのように考慮するかを示します。

オプション	デフォルト	説明
node-health-strategy	none	ヘルスリソースエージェントと組み合わせて使用し、Pacemaker がノードヘルスの変化にどのように応答するかを制御します。設定できる値は以下のとおりです。 * **none** - ノードヘルスを追跡しません。 * **migrate-on-red** - リソースは、エージェントが監視するローカルの状態に基づき、ノードのステータスが **red** であるとヘルスエージェントが判断したノードから移動されます。 * **only-green** - リソースは、エージェントが監視するローカルの状態に基づき、ノードのステータスが **yellow** または **red** であるとヘルスエージェントが判断したノードから移動されます。 * **progressive**, **custom** - ヘルス属性の内部数値に従って、ヘルス状態に対するクラスターの応答をよりきめ細かく制御できる高度なノードヘルスストラテジー。

22.2. クラスターのプロパティーの設定と削除

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ーを削除する場合は、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、**symmetric-cluster** プロパティーを **false** に設定したことがある場合は、設定した値が次のコマンドにより削除され、**symmetric-cluster** の値がデフォルト値の **true** に戻されます。

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

22.3. クラスタープロパティー設定のクエリー

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** を同じように使用できます。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される形式で、**pcs show** は、特定のプロパティーの値を表示する場合に使用される形式です。

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
```

明示的に設定されていないプロパティー設定のデフォルト値など、クラスターのプロパティー設定の値をすべて表示する場合は、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
```

情報提供の目的で、次のコマンドを使用して、プロパティーがデフォルト以外の値に設定されているかどうかに関わらず、プロパティーのデフォルト値のリストを表示できます。

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

22.4. PCS コマンドとしてのクラスタープロパティーのエクスポート

Red Hat Enterprise Linux 9.3 では、**pcs property config** コマンドの **--output-format=cmd** オプションを使用して、設定済みのクラスタープロパティーを別のシステムに再作成するのに使用できる **pcs** コマンドを表示できます。

次のコマンドは、**migration-limit** クラスタープロパティーを 10 に設定します。

```
# pcs property set migration-limit=10
```

クラスタープロパティーを設定した後に次のコマンドを実行すると、別のシステムでクラスタープロパティーを設定するために使用できる **pcs** コマンドが表示されます。

```
# pcs property config --output-format=cmd
pcs property set --force -- \
 migration-limit=10 \
 placement-strategy=minimal
```

第23章 ノードの正常なシャットダウン時に停止したままになるようにリソースを設定

クラスターノードがシャットダウンしたときの Pacemaker のデフォルトの応答は、シャットダウンが正常なシャットダウンであっても、そのノードで実行中のすべてのリソースを停止し、別の場所でリソースを復元することです。ノードが正常にシャットダウンすると、ノードに接続されているリソースがノードにロックされ、シャットダウンしたノードがクラスターに再度参加するときに再び起動するまで、他の場所で起動できないように、Pacemaker を設定できます。これにより、ノードのリソースをクラスター内の他のノードにフェイルオーバーせずに、サービスの停止が許容できるメンテナンスウィンドウ中にノードの電源を切ることができます。

23.1. ノードの正常なシャットダウン時に停止したままになるようにリソースを設定するためのクラスタープロパティー

ノードの正常なシャットダウンでリソースがフェイルオーバーしないようにする機能は、以下のクラスタープロパティーで実装されます。

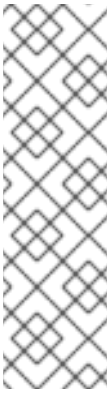
shutdown-lock

このクラスタープロパティーをデフォルト値の **false** に設定すると、クラスターは、ノードで適切にシャットダウンしているノードでアクティブなリソースを復旧します。このプロパティーを **true** に設定すると、適切にシャットダウンしているノードでアクティブなリソースは、クラスターに再参加した後にノードで再起動するまで別の場所で起動できなくなります。
shutdown-lock プロパティーはクラスターノードまたはリモートノードのいずれかで機能しますが、ゲストノードは機能しません。

shutdown-lock を **true** に設定すると、ノードがダウンした場合にクラスターリソースのロックを削除し、以下のコマンドを実行してノードで手動更新を実行してリソースを別の場所で起動できるようになります。

pcs resource refresh resource node=nodename

リソースのロックが解除されると、クラスターはリソースを別の場所に移動できるようになることに注意してください。リソースのスティッキネスの値または場所の設定を使用することにより、これが発生する可能性を抑制できます。



注記

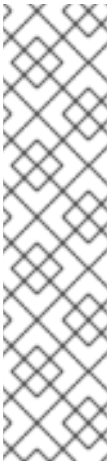
手動更新は、最初に次のコマンドを実行するとリモートノードで機能します。

1. リモートノードで **systemctl stop pacemaker_remote** コマンドを実行してノードを停止します。
2. **pcs resource disable remote-connection-resource** コマンドを実行します。

その後、リモートノードで手動更新を実行できます。

shutdown-lock-limit

このクラスタープロパティーをデフォルト値の 0 以外の値に設定すると、シャットダウンを開始してから指定した時間内にノードが再参加しない場合に、他のノードの復旧にリソースが利用可能になります。



注記

shutdown-lock-limit プロパティーは、以下のコマンドを最初に実行した場合に限りリモートノードで動作します。

1. リモートノードで **systemctl stop pacemaker_remote** コマンドを実行してノードを停止します。
2. **pcs resource disable remote-connection-resource** コマンドを実行します。

このコマンドの実行後、**shutdown-lock-limit** で指定した時間が経過すると、リモートノード上で実行しているリソースが他のノードの復旧に利用できます。

23.2. SHUTDOWN-LOCK クラスタープロパティーの設定

以下の例では、サンプルのクラスターで **shutdown-lock** クラスタープロパティーを **true** に設定し、ノードをシャットダウンして再起動したときの影響を示しています。この例のクラスターは、**z1.example.com**、**z2.example.com**、および **z3.example.com** の 3 つのノードで構成されます。

手順

1. **shutdown-lock** プロパティーを **true** に設定し、その値を確認します。この例では、**shutdown-lock-limit** プロパティーはデフォルト値 0 を維持します。

```
[root@z3 ~]# pcs property set shutdown-lock=true
[root@z3 ~]# pcs property list --all | grep shutdown-lock
 shutdown-lock: true
 shutdown-lock-limit: 0
```

2. クラスターのステータスを確認します。この例では、**3** 番目と **5** 番目のリソースが **z1.example.com** で実行しています。

```
[root@z3 ~]# pcs status
...
Full List of Resources:
...
 * first (ocf::pacemaker:Dummy): Started z3.example.com
 * second (ocf::pacemaker:Dummy): Started z2.example.com
 * third (ocf::pacemaker:Dummy): Started z1.example.com
 * fourth (ocf::pacemaker:Dummy): Started z2.example.com
 * fifth (ocf::pacemaker:Dummy): Started z1.example.com
...
```

3. **z1.example.com** をシャットダウンします。これにより、そのノードで実行中のリソースを停止します。

```
[root@z3 ~] # pcs cluster stop z1.example.com
Stopping Cluster (pacemaker)...
Stopping Cluster (corosync)...
```

4. **pcs status** コマンドを実行すると、ノードの **z1.example.com** がオフラインであることを示し、**z1.example.com** で実行していたリソースは、ノードの停止時に **LOCKED** になります。

```
[root@z3 ~]# pcs status
...

Node List:
 * Online: [ z2.example.com z3.example.com ]
 * OFFLINE: [ z1.example.com ]

Full List of Resources:
...
 * first (ocf::pacemaker:Dummy): Started z3.example.com
 * second (ocf::pacemaker:Dummy): Started z2.example.com
 * third (ocf::pacemaker:Dummy): Stopped z1.example.com (LOCKED)
 * fourth (ocf::pacemaker:Dummy): Started z3.example.com
 * fifth (ocf::pacemaker:Dummy): Stopped z1.example.com (LOCKED)

...
```

5. クラスターサービスを **z1.example.com** で再度起動し、クラスターに再参加できるようにします。ロックされたリソースは、そのノードで開始する必要がありますが、いったん起動すると、必ずしも同じノードに留まるわけではありません。

```
[root@z3 ~]# pcs cluster start z1.example.com
Starting Cluster...
```

6. この例では、**3** 番目および **5** 番目のリソースが、**z1.example.com** ノードで復元されます。

```
[root@z3 ~]# pcs status
...

Node List:
 * Online: [ z1.example.com z2.example.com z3.example.com ]

Full List of Resources:
..
 * first (ocf::pacemaker:Dummy): Started z3.example.com
 * second (ocf::pacemaker:Dummy): Started z2.example.com
 * third (ocf::pacemaker:Dummy): Started z1.example.com
 * fourth (ocf::pacemaker:Dummy): Started z3.example.com
 * fifth (ocf::pacemaker:Dummy): Started z1.example.com

...
```

第24章 ノード配置ストラテジーの設定

Pacemaker は、すべてのノードのリソース割り当てスコアに従って、リソースを配置する場所を決定します。このリソースは、リソースのスコアが最も高いノードに割り当てられます。この割り当てスコアは、リソースの制約、**resource-stickiness** の設定、各ノードにおけるリソースの過去の障害履歴、各ノードの使用率などの要因の組み合わせから導出されます。

すべてのノードでリソース割り当てスコアが等しい場合、デフォルトの配置ストラテジーにより、Pacemaker は、負荷を分散するために、割り当てられたリソースの数が最も少ないノードを選択します。各ノードのリソースの数が等しい場合は、CIB の最初の対象ノードがリソースを実行するのに選択されます。

ただし、多くの場合、リソースが使用するノードの容量 (メモリーや I/O など) の割合は、状況によって大きく異なります。そのため、ノードに割り当てられているリソースの数のみを考慮して、いつでも思想的な負荷分散が行われるとは限りません。さらに、組み合わせた要件が指定された容量を超えるように、リソースが配置されている場合、リソースが完全に起動しないか、パフォーマンスが低下して実行される可能性があります。このような要因を考慮するために、Pacemaker では次の要素を設定できます。

- 特定のノードの容量
- 特定のリソースが必要な容量
- リソースの配置の全体的なストラテジー

24.1. 使用率属性および配置ストラテジー

ノードが提供する容量、またはリソースが必要な容量を設定するには、ノードとリソースに **使用率属性** を使用します。これを行うには、リソースの使用率変数を設定し、その変数に値を割り当ててリソースに必要なものを示してから、同じ使用率変数をノードに設定し、その変数に値を割り当ててそのノードが提供するものを示します。

設定に応じて使用率属性に名前を付け、設定に必要な数だけ名前と値のペアを定義できます。使用率属性の値は整数である必要があります。

24.1.1. ノードおよびリソース容量の設定

以下の例では、2 つのノードに CPU 容量の使用率属性を設定し、この属性を変数 **cpu** として設定します。また、RAM 容量の使用率属性を設定し、この属性を変数 **memory** として設定します。この例では、以下のように設定されています。

- ノード 1 は、CPU 容量 2 と、RAM 容量 2048 を指定するように定義されています。
- ノード 2 は、CPU 容量 4 と、RAM 容量 2048 を指定するように定義されています。

```
# pcs node utilization node1 cpu=2 memory=2048
# pcs node utilization node2 cpu=4 memory=2048
```

次の例では、3 つの異なるリソースに必要な、同じ使用率属性を指定します。この例では、以下のように設定されています。

- **dummy-small** リソースには、CPU 容量 1 と、RAM 容量 1024 が必要です。
- **dummy-medium** リソースには、CPU 容量 2 と、RAM 容量 2048 が必要です。

- **dummy-large** リソースには、CPU 容量 1 と、RAM 容量 3072 が必要です。

```
# pcs resource utilization dummy-small cpu=1 memory=1024
# pcs resource utilization dummy-medium cpu=2 memory=2048
# pcs resource utilization dummy-large cpu=3 memory=3072
```

使用率属性で定義されているリソースの要件を満たすのに十分な空き容量があれば、ノードはリソースに適格と見なされます。

24.1.2. 配置ストラテジーの設定

ノードが提供する容量と、リソースが必要とする容量を設定したら、**placement-strategy** クラスタープロパティーを設定する必要があります。これを設定しないと、容量を設定しても効果がありません。

placement-strategy クラスタープロパティーには、4 つの値を使用できます。

- **default** - 使用率の値は全く考慮されません。リソースは、割り当てスコアに従って割り当てられます。スコアが同じであれば、リソースはノード間で均等に分散されます。
- **utilization** - 使用率の値は、ノードが適格と見なされるかどうか (つまり、リソースの要件を満たすのに十分な空き容量があるかどうか) を決定する場合にのみ考慮されます。負荷分散は、ノードに割り当てられたリソースの数に基づいて行われます。
- **balanced** - 使用率の値は、ノードがリソースを提供する資格があるかどうかを判断する際、および負荷分散する際に考慮されるため、リソースのパフォーマンスを最適化する方法でリソースを分散しようとします。
- **minimal** - 使用率の値は、ノードがリソースを提供する資格があるかどうかを決定する場合に限り考慮されます。負荷分散は、できるだけ少ないノードにリソースを集中させて、残りのノードで電力を節約できるようにします。

以下の例のコマンドでは、**placement-strategy** の値を **balanced** に設定します。このコマンドを実行すると、Pacemaker は、複雑な一連のコロケーション制約を使用せずに、リソースからの負荷がクラスター全体に均等に分散されるようにします。

```
# pcs property set placement-strategy=balanced
```

24.2. PACEMAKER リソースの割り当て

Pacemaker は、ノードの優先度、ノードの容量、およびリソース割り当ての優先度に従ってリソースを割り当てます。

24.2.1. ノード設定

Pacemaker は、以下のストラテジーに従ってリソースを割り当てる際に優先されるノードを決定します。

- 重みが最も高いノードが最初に消費されます。ノードの重みは、ノードの状態を表すためにクラスターによって維持されるスコアです。
- ノードの重みが、複数のノードで同じ場合は、以下のようになります。
  - **placement-strategy** クラスタープロパティーが **default** または **utilization** の場合は、以下のようになります。

        - 割り当てられているリソースの数が最も少ないノードが最初に使用されます。
        - 割り当てられているリソースの数が等しい場合は、CIB に登録されている最初の対象ノードが最初に使用されます。
    - **placement-strategy** クラスタープロパティーが **balanced** である場合は、以下のようになります。
        - 空き容量が最も多いノードが最初に使用されます。
        - ノードの空き容量が等しい場合は、割り当てられているリソースの数が最も少ないノードが最初に使用されます。
        - ノードの空き容量が等しく、割り当てられているリソースの数が等しい場合は、CIB に最初に登録されている対象ノードが最初に使用されます。
    - **placement-strategy** クラスタープロパティーが **minimal** の場合は、CIB に登録されている最初の対象ノードが最初に使用されます。

24.2.2. ノードの容量

Pacemaker は、以下のストラテジーに従って、どのノードに最も空き容量があるかを判断します。

- 使用率属性が 1 種類だけ定義されている場合、空き容量は単純な数値比較となります。
- 定義されている使用属性の種類が複数になる場合は、ほとんどの属性タイプで数値が最も高いノードの空き容量が、最も大きくなります。以下に例を示します。
    - NodeA の空き CPU が多く、NodeB の空きメモリーが多い場合は、互いの空き容量は等しくなります。
    - NodeA の空き CPU が多く、NodeB の空きメモリーとストレージが多い場合、NodeB の方が空き容量が多くなります。

24.2.3. リソースの割り当て設定

Pacemaker は、以下のストラテジーに従って、最初に割り当てられるリソースを決定します。

- 優先度の最も高いリソースが最初に割り当てられます。リソースの作成時に、リソースの優先度を設定できます。
- リソースの優先度が等しい場合は、リソースのシャッフルを防ぐために、実行中のノードで最も高いスコアを持つリソースが最初に割り当てられます。
- リソースが実行しているノードのリソーススコアが等しい場合や、リソースが実行していない場合は、優先ノードで最もスコアが高いリソースが最初に割り当てられます。この場合、優先ノードのリソーススコアが等しい場合は、CIB に最初に登録されている実行可能なリソースが最初に割り当てられます。

24.3. リソース配置ストラテジーのガイドライン

リソースに対する Pacemaker の配置ストラテジーが最も効果的に機能するようにするためにも、システムを設定するときに次の点を考慮する必要があります。

- 物理容量が十分にあることを確認します。

ノードの物理容量が、通常の状態でほぼ最大に使用されているとすると、フェイルオーバーの際に問題が発生する可能性があります。使用率機能がなくても、タイムアウトや二次障害が発生する可能性があります。

- ノードに設定する容量にバッファーをいくつか構築します。
  物理的に存在するよりもわずかに多くのノードリソースが通知されます。ここでは、Pacemaker リソースが、設定した CPU、メモリーなどを常に 100% 使用しないことが想定されます。このような状況は、オーバーコミットと呼ばれることもあります。

- リソースの優先度を指定します。
  クラスターがサービスを犠牲にする場合、犠牲にするサービスは一番重要でないことが理想的です。最も重要なリソースが最初にスケジュールされるように、リソースの優先度が適切に設定されていることを確認してください。

24.4. NODEUTILIZATION リソースエージェント

NodeUtilization エージェントは、使用可能な CPU、ホストメモリーの可用性、およびハイパーバイザーのメモリーの可用性に関するシステムパラメーターを検出し、このようなパラメーターを CIB に追加します。エージェントをクローンリソースとして実行して、各ノードにこのようなパラメーターを自動的に入力することができます。

NodeUtilization リソースエージェントと、このエージェントのリソースオプションの説明は、**pcs resource describe NodeUtilization** コマンドを実行してください。

第25章 仮想ドメインをリソースとして設定

pcs resource create コマンドを使用して、**VirtualDomain** をリソースタイプとして指定すると、**libvirt** 仮想化フレームワークが管理する仮想ドメインを、クラスターリソースとして設定できます。

仮想ドメインをリソースとして設定する場合は、以下の点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する必要があります。
- 仮想ドメインをクラスターリソースにすると、クラスターツールを使用しない限り、起動、停止、または移行を行うことができません。
- 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。
- 仮想ドメインの実行を許可するすべてのノードは、その仮想ドメインに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ようにする必要があります。

クラスターが仮想ドメイン内のサービスを管理できるようにする場合は、仮想ドメインをゲストノードとして設定できます。

25.1. 仮想ドメインリソースのオプション

以下の表は、**VirtualDomain** リソースに設定できるリソースオプションを説明します。

表25.1 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でのみ、強制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に対応していない場合に限り、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で設定されます。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 (.) の直前に接尾辞を挿入します。この設定されたホスト名がローカルで解決可能であり、関連する 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場合は、監視するスクリプトのリストとともに、このパラメーターを追加します。**注記**: 監視スクリプトを使用する場合、**start** および **migrate_from** の操作は、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。この遅延に対応できるように、この操作のタイムアウトを必ず設定してください。
autoset_utilization_cpu	**true**	**true** に設定すると、監視の実行時に、エージェントが **virsh** から **domainU** の **vCPU** 数を検出し、これをリソースの CPU 使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、エージェントが **virsh** から **Max memor** 数を検出し、これをリソースの **hv_memory** の使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へのパス。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。起動時にドメインのステータスファイルが存在すると、ドメインは、最後に停止する直前の状態に復元されます。このオプションは、**force_stop** オプションと互換性がありません。

VirtualDomain リソースオプションに加えて、**allow-migrate** メタデータオプションを設定して、リソースの別のノードへのライブ移行を許可できます。このオプションを **true** に設定すると、状態を失うことなくリソースを移行できます。このオプションがデフォルトの状態である **false** に設定されていると、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てから、2 番目のノードで再起動します。

25.2. 仮想ドメインリソースの作成

以下の手順では、以前に作成した仮想マシンのクラスターに **VirtualDomain** リソースを作成します。

手順

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、ディスクのファイルに、仮想マシンの **xml** 設定ファイルを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た場合は、ゲストを実行できるクラスターノードのいずれかにファイルに **xml** ファイルをダンプします。任意の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. 仮想マシンの **xml** 設定ファイルを、各ノードの同じ場所にあるゲストを実行できるその他のすべてのクラスターノードにコピーします。

3. 仮想ドメインの実行が許可されているすべてのノードが、その仮想ドメインに必要なストレージデバイスにアクセスできるようにします。

4. 仮想ドメインが、仮想ドメインを実行する各ノードで起動および停止できることを別途テストします。

5. 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。Pacemaker は、クラスターで設定されているノードを起動します。仮想マシンは、ホストの起動時に自動的に起動するように設定することはできません。

6. **pcs resource create** コマンドを使用して、**VirtualDoman** リソースを設定します。たとえば、次のコマンドは、**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 resource move VM nodeX** コマンドはライブ移行として実行されます。
   この例では、**migration_transport** が **ssh** に設定されます。SSH 移行が適切に機能するには、鍵を使用しないロギングがノード間で機能する必要があります。

```
# pcs resource create VM VirtualDomain config=/etc/pacemaker/guest1.xml
migration_transport=ssh meta allow-migrate=true
```

第26章 クラスタークォーラムの設定

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、**votequorum** サービスをフェンシングと併用します。クラスターの各システムには多くの投票数が割り当てられ、過半数の票を取得しているものだけがクラスターの操作を継続できます。サービスは、すべてのノードに読み込むか、いずれのノードにも読み込まないようにする必要があります。サービスをクラスターノードのサブセットに読み込むと、結果が予想できなくなります。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、結果が予想不可能になります。**votequorum** サービスの設定および操作の詳細は、**votequorum** (5) の man ページを参照してください。

26.1. 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。以下の表には、これらのオプションをまとめています。

表26.1 クォーラムオプション

オプション	説明
auto_tie_breaker	これを有効にすると、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**) と通信したままのノードのセットは、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、均等の分割でクラスターが動作を継続できるようになるため、主に偶数個のノードがあるクラスターで使用されます。複数の不均等な分割など、より複雑な失敗の場合には、クォーラムデバイスを使用することが推奨されます。 **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、初回だけ、クラスターが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 **lms** (last man standing) アルゴリズムを使用する 2 ノードクラスター、および偶数のノードで設定されるクラスターに使用されます。 **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。

オプション	説明
last_man_standing	有効にすると、クラスターは特定の状況で **expected_votes** とクォーラムを動的に再計算します。このオプションを有効にする場合は、**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。
last_man_standing_window	クラスターのノードが失われた後の、**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) です。

このオプションの設定および使用の詳細は、man ページの **votequorum**(5) を参照してください。

26.2. 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。稼働中のシステムでは、**quorum.two_node** および **quorum.expected_votes** オプションを変更できます。その他すべてのクォーラムオプションについては、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
```

26.3. クォーラム設定およびステータスの表示

クラスターを実行したら、以下のクラスタークォーラムコマンドを実行して、クォーラムの設定やステータスを表示できます。

次のコマンドは、クォーラムの設定を表示します。

```
pcs quorum [config]
```

次のコマンドは、クォーラムのランタイム状態を表示します。

```
pcs quorum status
```

26.4. クォーラムに達しないクラスターの実行

長時間クラスターでノードを使用しなかったためにクォーラムが失われた場合に、**pcs quorum expected-votes** コマンドを使用して、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、クォーラムがない場合でも、クラスターは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動で変更したために、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個別に起動して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。

次のコマンドは、ライブクラスターで期待される票数を、指定の値に設定します。これはライブクラスターにのみ影響し、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の電源を切り、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
```

第27章 クォーラムデバイスの設定

クラスター用のサードパーティー仲裁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で許容されるよりも多くのノード障害に耐えられるようにすることができます。クォーラムデバイスは、偶数のノードで設定されるクラスターに推奨されます。2 ノードクラスターでクォーラムデバイスを使用すると、スプリットブレインの状況で存続するノードをより適切に判別できます。

クォーラムデバイスを設定する場合は、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別の物理ネットワークで実行することが推奨されます。理想としては、クォーラムデバイスホストを、メインクラスターとは別のラックに置くか、少なくとも別の PSU に置くようにします。corosync リングと同じネットワークセグメントには置かないようにしてください。
- 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。ただし、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が、複数の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。

27.1. 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:~]# dnf install corosync-qdevice
[root@node2:~]# dnf install corosync-qdevice
```

- クォーラムデバイスホストに、**pcs** および **corosync-qnetd** をインストールします。

```
[root@qdevice:~]# dnf 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
```

27.2. クォーラムデバイスの設定

次の手順でクォーラムデバイスを設定し、クラスターに追加します。

この例では、以下のように設定されています。

- 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、これは現在対応し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に対応します。
  - **ffsplit** (fifty-fifty split) -これにより、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、1票が返されます。

警告

LMS アルゴリズムにより、ノードが1つしか残っていなくてもクラスターはクォーラムを維持できますが、number_of_nodes - 1と同じであるため、クォーラムデバイスの投票力が大きいことを意味します。クォーラムデバイスとの接続が失われると、number_of_nodes - 1の票が失われます。つまり、(クォーラムデバイスを無効にすることで) すべてのノードがアクティブなクラスターのみがクォーラムに達したままになります。他のクラスターは、クォーラムに達しなくなります。

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。

手順

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、システムの起動時にデバイスが開始するように設定します。

```
[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、そのステータスを確認できます。**corosync-qnetd** デーモンが実行中であり、この時点でクライアントが接続されていないことが分かります。**--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て、**pcsd** デーモンおよび **net** クォーラムデバイスで必要なファイアウォールのポートを有効にします。

```
[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 **hacluster** ユーザーを認証します。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、後で比較するために、クォーラムデバイスの現在の設定と状況を確認できます。このコマンドの出力から、クラスターがクォーラムデバイスを使用しておらず、各ノードの **Qdevice** メンバーシップのステータスが **NR** (登録されていない) であることが分かります。

```
[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、作成しておい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。ただし、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。以下のコマンド例では、**ffsplit** アルゴリズムを使用するようにクォーラムデバイスを設定します。クォーラムデバイスの設定オプションの詳細は、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のステータスを表示し、クォーラムデバイスが使用中であることを示します。各クラスターノードの **Qdevice** メンバーシップ情報のステータス値の意味は以下のとおりです。

- **A/NA** - クォーラムデバイスはが有効かどうか、つまり **qdevice** と **corosync** の間にハートビートがあるかどうかを示します。この値は常に、クォーラムデバイスが有効でなければなりません。

- **V/NV** - クォーラムデバイスがノードに投票した場合には **V** に設定されます。この例では、相互に通信できるため、両方のノードが **V** に設定されます。クラスターを 2 つの単一ノードクラスターに分割する場合には、ノードのいずれかが **V** に、他のノードは **NV** に設定されます。

- **MW/NMW** - 内部クォーラムデバイスフラグが設定されている (**MW**) か、設定されていない (**NMW**) かを示します。デフォルトでは、フラグは設定されず、値は **NMW** です。

```
[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次のコマンドを実行して、**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)
```

27.3. 定足数デバイスサービスの管理

次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
```

27.4. クラスターでのクォーラムデバイスの管理

クラスターのクォーラムデバイス設定の変更、クォーラムデバイスの無効化、クォーラムデバイスの削除にはさまざまな **pcs** コマンドを使用できます。

27.4.1. 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場合は、**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場合は、**pcs quorum device remove** コマンドおよび **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
```

27.4.2. クォーラムデバイスの削除

次のコマンドは、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すると、クォーラムデバイスの状態を表示するときに、次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
```

27.4.3. クォーラムデバイスの破棄

次のコマンドは、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にして停止し、設定ファイルをすべて削除します。

```
[root@qdevice:~]# pcs qdevice destroy net
Stopping quorum device...
quorum device stopped
quorum device disabled
Quorum device 'net' configuration files removed
```

第28章 クラスターイベントのスクリプトの実行 (トリガー)

Pacemaker クラスターはイベント駆動型のシステムで、イベントはリソースやノードの障害、設定の変更、またはリソースの開始や停止になります。Pacemaker クラスターアラートを設定して、アラートエージェントによりクラスターイベントが発生したときに何らかの外部アクションを実行できます。アラートエージェントは、クラスターがリソースエージェントを呼び出してリソースの設定と操作を処理するのと同じ方法で呼び出す外部プログラムです。

クラスターは、環境変数を用いてイベントの情報をエージェントに渡します。エージェントは、E メールメッセージの送信、ログのファイルへの記録、監視システムの更新など、この情報を自由に使用できます。

- Pacemaker は、デフォルトで **/usr/share/pacemaker/alerts** にインストールされるアラートエージェントのサンプルを複数提供します。これらのサンプルスクリプトは、コピーしてそのまま使用したり、目的に合わせて編集するテンプレートとして使用することもできます。対応する全属性は、サンプルエージェントのソースコードを参照してください。
- サンプルアラートエージェントがニーズを満たしていない場合は、Pacemaker アラートを呼び出すアラートエージェントを作成できます。

28.1. サンプルアラートエージェントのインストールおよび設定

サンプルアラートエージェントの1つを使用するとき、スクリプトがニーズにあっていることを確認してください。サンプルエージェントは、特定のクラスター環境用のカスタムスクリプトを作成するためのテンプレートとして提供されます。Red Hat は、Pacemaker との通信にアラートエージェントスクリプトが使用するインターフェイスをサポートしますが、カスタムエージェント自体にはサポートを提供していないことに注意してください。

サンプルアラートエージェントの1つ を使用するには、クラスターの各ノードにエージェントをインストールする必要があります。たとえば、次のコマンドは、**alert_file.sh.sample** スクリプトを **alert_file.sh** としてインストールします。

```
# install --mode=0755 /usr/share/pacemaker/alerts/alert_file.sh.sample
/var/lib/pacemaker/alert_file.sh
```

スクリプトをインストールしたら、スクリプトを使用するアラートを作成できます。

以下の例では、インストールした **alert_file.sh** アラートエージェントを使用してイベントのログをファイルに記録するアラートを設定します。アラートエージェントは、最低限のパーミッションを持つ **hacluster** ユーザーとして実行します。

この例では、イベントの記録に使用するログファイル **pcmk_alert_file.log** を作成します。また、アラートエージェントを作成し、その受信先としてログファイルへのパスを追加します。

```
# touch /var/log/pcmk_alert_file.log
# chown hacluster:haclient /var/log/pcmk_alert_file.log
# chmod 600 /var/log/pcmk_alert_file.log
# pcs alert create id=alert_file description="Log events to a file."
path=/var/lib/pacemaker/alert_file.sh
# pcs alert recipient add alert_file id=my-alert_logfile value=/var/log/pcmk_alert_file.log
```

以下の例では、**alert_snmp.sh.sample** スクリプトを **alert_snmp.sh** としてインストールし、インストールした **alert_snmp.sh** アラートエージェントを使用してクラスターイベントを NSMP トラップとして送信するアラートを設定します。デフォルトでは、正常な監視呼び出し以外のすべてのイベントを

SNMP サーバーに送信します。この例では、タイムスタンプの形式をメタオプションとして設定します。この例では、アラートの設定後にアラートの受信側が設定され、アラート設定が表示されます。

```
# install --mode=0755 /usr/share/pacemaker/alerts/alert_snmp.sh.sample
/var/lib/pacemaker/alert_snmp.sh
# pcs alert create id=snmp_alert path=/var/lib/pacemaker/alert_snmp.sh meta timestamp-
format="%Y-%m-%d,%H:%M:%S.%01N"
# pcs alert recipient add snmp_alert value=192.168.1.2
# pcs alert
Alerts:
 Alert: snmp_alert (path=/var/lib/pacemaker/alert_snmp.sh)
  Meta options: timestamp-format=%Y-%m-%d,%H:%M:%S.%01N.
  Recipients:
   Recipient: snmp_alert-recipient (value=192.168.1.2)
```

以下の例は、**alert_smtp.sh** エージェントをインストールし、インストールしたアラートエージェントを使用するアラートを設定して、クラスターイベントを E メールメッセージとして送信します。この例では、アラートの設定後に受信側が設定され、アラート設定が表示されます。

```
# install --mode=0755 /usr/share/pacemaker/alerts/alert_smtp.sh.sample
/var/lib/pacemaker/alert_smtp.sh
# pcs alert create id=smtp_alert path=/var/lib/pacemaker/alert_smtp.sh options
email_sender=donotreply@example.com
# pcs alert recipient add smtp_alert value=admin@example.com
# pcs alert
Alerts:
 Alert: smtp_alert (path=/var/lib/pacemaker/alert_smtp.sh)
  Options: email_sender=donotreply@example.com
  Recipients:
   Recipient: smtp_alert-recipient (value=admin@example.com)
```

28.2. クラスターアラートの作成

次のコマンドは、クラスターアラートを作成します。設定するオプションは、追加の環境変数として指定するパスで、アラートエージェントスクリプトに渡されるエージェント固有の設定値です。**id** の値を指定しないと、値が生成されます。

```
pcs alert create path=path [id=alert-id] [description=description] [options [option=value]...] [meta
[meta-option=value]...]
```

複数のアラートエージェントを設定できます。クラスターは、各イベントに対して、すべてのアラートエージェントを呼び出します。アラートエージェントはクラスターノードでのみ呼び出されます。アラートエージェントは、Pacemaker リモートノードが関係するイベントに対して呼び出されますが、このようなノードでは呼び出されません。

以下の例は、各イベントで **myscript.sh** を呼び出す簡単なアラートを作成します。

```
# pcs alert create id=my_alert path=/path/to/myscript.sh
```

28.3. クラスターアラートの表示、変更、および削除

クラスターアラートの表示、変更、および削除に使用できる **pcs** コマンドは複数あります。

次のコマンドは、設定されたすべてのアラートと、設定されたオプションの値を表示します。

```
pcs alert [config|show]
```

以下のコマンドは、指定した **alert-id** 値を持つ既存のアラートを更新します。

```
pcs alert update alert-id [path=path] [description=description] [options [option=value]...] [meta
[meta-option=value]...]
```

次のコマンドは、指定した **alert-id** 値を持つアラートを削除します。

```
pcs alert remove alert-id
```

代わりに **pcs alert delete** コマンドを実行できます。これは **pcs alert remove** コマンドと同じです。**pcs alert delete** コマンドおよび **pcs alert remove** コマンドの両方を使用すると、複数のアラートを削除できるようになります。

28.4. クラスターアラート受信側の設定

通常、アラートは受信側に送信されます。したがって、各アラートには、1 人以上の受信者を追加で設定できます。クラスターは、受信側ごとに別々にエージェントを呼び出します。

受信側は、IP アドレス、メールアドレス、ファイル名、特定のエージェントがサポートするものなど、アラートエージェントが認識できるものを設定します。

次のコマンドは、新しい受信側を指定のアラートに追加します。

```
pcs alert recipient add alert-id value=recipient-value [id=recipient-id] [description=description]
[options [option=value]...] [meta [meta-option=value]...]
```

次のコマンドは、既存のアラート受信側を更新します。

```
pcs alert recipient update recipient-id [value=recipient-value] [description=description] [options
[option=value]...] [meta [meta-option=value]...]
```

次のコマンドは、指定のアラート受信側を削除します。

```
pcs alert recipient remove recipient-id
```

代わりに、**pcs alert recipient delete** コマンドを実行できます。これは、**pcs alert recipient remove** コマンドと同じです。**pcs alert recipient remove** コマンドおよび **pcs alert recipient delete** コマンドの両方を使用すると、複数のアラート受信者を削除できます。

次のコマンド例は、受信者 ID が **my-recipient-id** のアラート受信側 **my-alert-recipient** を、アラート **my-alert** に追加します。これにより、クラスターが各イベントの **my-alert** 用に設定したアラートスクリプトを呼び出すように設定され、受信者 **some-address** が環境変数として渡されます。

```
# pcs alert recipient add my-alert value=my-alert-recipient id=my-recipient-id options
value=some-address
```

28.5. アラートメタオプション

リソースエージェントと同様に、メタオプションをアラートエージェントに対して設定すると、Pacemaker の呼び出し方法を調整できます。以下の表は、アラートメタオプションを示しています。メタオプションは、アラートエージェントごと、または受信側ごとに設定できます。

表28.1 アラートメタオプション

メタ属性	デフォルト	説明
enabled	**true**	(RHEL 9.3 以降) アラートに対して **false** と設定すると、アラートは使用されません。アラートに対して **true** と設定し、そのアラートの特定の受信者に対して **false** と設定している場合、その受信者は使用されません。
timestamp-format	%H:%M:%S.%06N	イベントのタイムスタンプをエージェントに送信するときにクラスターが使用する形式です。この文字列は **date**(1) コマンドで使用されます。
timeout	30s	アラートエージェントがこの時間内に完了しないと終了させられます。

以下の例は、**myscript.sh** スクリプトを呼び出すアラートを設定し、2 つの受信側をアラートに追加します。最初の受信側の ID は **my-alert-recipient1** で、2 つ目の受信側の ID は **my-alert-recipient2** です。スクリプトは各イベントで 2 回呼び出され、呼び出しのタイムアウト値はそれぞれ 15 秒です。呼び出しの 1 つは受信側 **someuser@example.com** に渡され、タイムスタンプの形式は %D %H:%M になります。もう 1 つの呼び出しは受信側 **otheruser@example.com** へ渡され、タイムスタンプの形式は %c になります。

```
# pcs alert create id=my-alert path=/path/to/myscript.sh meta timeout=15s
# pcs alert recipient add my-alert value=someuser@example.com id=my-alert-recipient1 meta timestamp-format="%D %H:%M"
# pcs alert recipient add my-alert value=otheruser@example.com id=my-alert-recipient2 meta timestamp-format="%c"
```

28.6. クラスターアラート設定コマンドの例

以下の例は、基本的なアラート設定コマンドの一部と、アラートの作成、受信側の追加、および設定されたアラートの表示に使用される形式を表しています。

クラスターの各ノードにアラートエージェント自体をインストールする必要がありますが、**pcs** コマンドを実行する必要があるのは 1 回だけです。

以下のコマンドは簡単なアラートを作成し、アラートに 2 つの受信側を追加した後、設定された値を表示します。

- アラート ID の値が指定されていないため、**alert** のアラート ID が作成されます。

- 最初の受信側作成コマンドは、**rec_value** の受信側を指定します。このコマンドには受信側 ID が指定されていないため、**alert-recipient** の値が受信側 ID として使用されます。
- 2 番目の受信側作成コマンドは、**rec_value2** の受信側を指定します。このコマンドは、**my-recipient** を受信側 ID として指定します。

```
# pcs alert create path=/my/path
# pcs alert recipient add alert value=rec_value
# pcs alert recipient add alert value=rec_value2 id=my-recipient
# pcs alert config
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
   Recipient: my-recipient (value=rec_value2)
```

以下のコマンドは、2 番目のアラートとそのアラートの受信側を追加します。2 番目のアラートのアラート ID は **my-alert** で、受信側の値は **my-other-recipient** です。受信側 ID が指定されていないため、**my-alert-recipient** が受信側 ID として使用されます。

```
# pcs alert create id=my-alert path=/path/to/script description=alert_description options
option1=value1 opt=val meta timeout=50s timestamp-format="%H%B%S"
# pcs alert recipient add my-alert value=my-other-recipient
# pcs alert
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
   Recipient: my-recipient (value=rec_value2)
 Alert: my-alert (path=/path/to/script)
  Description: alert_description
  Options: opt=val option1=value1
  Meta options: timestamp-format=%H%B%S timeout=50s
  Recipients:
   Recipient: my-alert-recipient (value=my-other-recipient)
```

以下のコマンドは、アラート **my-alert** と受信側 **my-alert-recipient** のアラート値を変更します。

```
# pcs alert update my-alert options option1=newvalue1 meta timestamp-format="%H%M%S"
# pcs alert recipient update my-alert-recipient options option1=new meta timeout=60s
# pcs alert
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
   Recipient: my-recipient (value=rec_value2)
 Alert: my-alert (path=/path/to/script)
  Description: alert_description
  Options: opt=val option1=newvalue1
  Meta options: timestamp-format=%H%M%S timeout=50s
  Recipients:
   Recipient: my-alert-recipient (value=my-other-recipient)
    Options: option1=new
    Meta options: timeout=60s
```

次のコマンドは、受信側 **my-alert-recipient** を **alert** から削除します。

```
# pcs alert recipient remove my-recipient
# pcs alert
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
 Alert: my-alert (path=/path/to/script)
  Description: alert_description
  Options: opt=val option1=newvalue1
  Meta options: timestamp-format="%M%B%S" timeout=50s
  Recipients:
   Recipient: my-alert-recipient (value=my-other-recipient)
    Options: option1=new
    Meta options: timeout=60s
```

次のコマンドは、設定から **myalert** を削除します。

```
# pcs alert remove myalert
# pcs alert
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
```

28.7. クラスターアラートエージェントの作成

Pacemaker クラスターアラートには、ノードアラート、フェンスアラート、およびリソースアラートの 3 種類があります。アラートエージェントに渡される環境変数は、アラートのタイプによって異なります。以下の表は、アラートエージェントに渡される環境変数を示し、環境変数が特定のアラートタイプに関連付けられるタイミングを指定します。

表28.2 アラートエージェントに渡される環境変数

環境変数	説明
CRM_alert_kind	アラートの種類 (ノード、フェンス、またはリソース)
CRM_alert_version	アラートを送信する Pacemaker のバージョン
CRM_alert_recipient	設定された送信側
CRM_alert_node_sequence	アラートがローカルノードで発行されるたびに増加するシーケンス番号。これは、Pacemaker によりアラートが発行された順序を参照するのに使用できます。後で発生したイベントのアラートは、先に発生したイベントのアラートよりもシーケンス番号が大きくなります。この番号は、クラスター全体を対象とする番号ではないことに注意してください。

環境変数	説明
CRM_alert_timestamp	**timestamp-format** メタオプションで指定された形式で、エージェントの実行前に作成されたタイムスタンプ。これにより、エージェントは、エージェント自体が呼び出されたタイミング (システムの負荷やその他の状況により遅延する可能性があります) に関係なく、信頼できる高精度のイベント発生時間を使用できます。
CRM_alert_node	影響を受けるノードの名前
CRM_alert_desc	イベントの詳細。ノードアラートの場合は、ノードの現在の状態 (番号または lost) になります。フェンスアラートの場合は、フェンス操作の要求元、ターゲット、フェンス操作のエラーコードなどを含む要求されたフェンス操作の概要になります。リソースアラートの場合は、**CRM_alert_status** と同等の読み取り可能な文字列になります。
CRM_alert_nodeid	状態が変更したノードの ID (ノードアラートの場合のみ提供)。
CRM_alert_task	要求されたフェンスまたはリソース操作 (フェンスおよびリソースアラートの場合のみ提供)。
CRM_alert_rc	フェンスまたはリソース操作の数値の戻りコード (フェンスおよびリソースアラートの場合のみ提供)。
CRM_alert_rsc	影響を受けるリソースの名前 (リソースアラートのみ)。
CRM_alert_interval	リソース操作の間隔 (リソースアラートのみ)
CRM_alert_target_rc	操作の予期される数値の戻りコード (リソースアラートのみ)。
CRM_alert_status	Pacemaker が、操作の結果を示すために使用する数値コード (リソースアラートのみ)。

アラートエージェントを記述する際は、以下を考慮する必要があります。

- アラートエージェントは受信者なしで呼び出されることがあります (受信者が設定されていない場合)。したがって、エージェントは、このような状況では終了しかしない場合でも、この状態に対応できなければなりません。設定を段階的に変更し、後で受信側を追加することもできます。
- 1 つのアラートに複数の受信側が設定されると、アラートエージェントは受信側ごとに 1 回呼び出されます。エージェントが同時に実行できない場合は、受信側を 1 つのみ設定する必要があります。エージェントは、受信側をリストとして解釈することができます。

- クラスターイベントの発生時、すべてのアラートは別々のプロセスとして同時に発生します。設定されているアラートと受信者の数、およびアラートエージェント内で行われている内容に応じて、負荷が急激に増加する可能性があります。たとえば、リソースを大量に消費するアクションを直接実行するのではなく、別のインスタンスのキューに追加することで、これを考慮に入れるようにエージェントを作成できます。

- アラートエージェントは、最低限のパーミッションを持つ **hacluster** ユーザーで実行します。アラートエージェントに追加のパーミッションが必要な場合は、適切な特権を持つ別のユーザーが、エージェントが必要なコマンドを実行できるように、**sudo** を設定することが推奨されます。

- **CRM_alert_timestamp** (このコンテンツはユーザー設定の **timestamp-format** によって指定)、**CRM_alert_recipient**、すべてのアラートオプションなど、ユーザー設定のパラメーターを検証およびサニタイズする場合は十分注意してください。これは、設定エラーから保護するために必要です。また、クラスターノードへの **hacluster** レベルのアクセスがなくても CIB を変更できるユーザーが存在する場合は、セキュリティーの問題が発生する可能性もあり、コードを挿入できないようにする必要があります。

- **onfail** パラメーターが **fence** に設定されている操作を持つリソースがクラスターに含まれる場合は、障害発生時に複数のフェンス通知 (このパラメーターが設定されているリソースごとに 1 つの通知と、追加の通知 1 つ) が送信されます。**pacemaker-fenced** および **pacemaker-controld** の両方が通知を送信します。この場合、送信される通知の数に関係なく、Pacemaker は 1 つのフェンス操作のみを実際に実行します。

注記

アラートインターフェイスは、**ocf:pacemaker:ClusterMon** リソースで使用される外部スクリプトインターフェイスと後方互換性を維持するよう設計されています。この互換性を維持するには、先頭に **CRM_notify_** および **CRM_alert_** が付けいたアラートエージェントに渡される環境変数を使用できます。互換性の問題の 1 つは、アラートエージェントが **hacluster** ユーザーで実行している最中に、**ClusterMon** リソースが root ユーザーで外部スクリプトを実行したことです。

第29章 マルチサイト PACEMAKER クラスター

クラスターが複数のサイトにまたがる場合は、サイト間のネットワーク接続の問題が原因でスプリットブレインが発生する可能性があります。接続が切断されたときに、別のサイトのノードで障害が発生したのか、サイト間の接続に失敗した状態で別サイトのノードが機能しているかどうかをノードが判断する方法がありません。さらに、同時を維持するには離れすぎている 2 つのサイト間で高可用性サービスを提供することが問題になることがあります。この問題に対応するため、Pacemaker は、Booth クラスターチケットマネージャーを使用して、複数のサイトにまたがる高可用性クラスターを設定する機能を完全にサポートしています。

29.1. BOOTH クラスターチケットマネージャーの概要

Booth **チケットマネージャー** は、特定サイトのクラスターノードを接続するネットワークとは異なる物理ネットワークで実行することを目的とした分散サービスです。それは、サイトの通常のクラスターにある別のゆるいクラスターである **Booth フォーメーション** を生成します。この集約通信層は、個別の Booth チケットに対して合意ベースの決定プロセスを促進します。

Booth **チケット** は、Booth フォーメーションのシングルトンで、時間に依存する移動可能な承認の単位を表します。実行には特定のチケットを要求するようにリソースを設定できます。これにより、1 つまたは複数のチケットが付与されているサイトで、リソースは一度に 1 つのサイトでのみ実行されるようになります。

Booth フォーメーションは、複数のサイトで実行しているクラスターで構成され、元のクラスターがすべて独立しているオーバーレイクラスターと考えることができます。チケット付与の有無についてクラスターと通信するのは Booth サービスで、Pacemaker のチケット制約に基づいてクラスターでリソースを実行するかどうかを判断するのは Pacemaker です。これは、チケットマネージャーを使用する場合に、各クラスターが独自のリソースと共有リソースを実行できることを示しています。たとえば、リソース A、B、および C は 1 つのクラスターでのみ実行され、リソース D、E、および F は別のクラスターでのみ実行されるとします。リソース G および H は、チケットによって決定されたこの 2 つのクラスターのいずれかで実行されます。また、別のチケットにより、この 2 つのクラスターのいずれかで実行できるリソース J を追加することもできます。

29.2. PACEMAKER を用いたマルチサイトクラスターの設定

次の手順で Booth チケットマネージャーを使用したマルチサイト設定を構築できます。

ここで使用するコマンド例は以下を前提とします。

- Cluster 1 は、ノード **cluster1-node1** および **cluster1-node2** で構成されます。
- Cluster 1 に割り当てられた Floating IP アドレスは 192.168.11.100 です。
- Cluster 2 は、**cluster2-node1** および **cluster2-node2** で構成されます。
- Cluster 2 に割り当てられた Floating IP アドレスは 192.168.22.100 です。
- 仲裁ノードは **arbitrator-node** で、IP アドレスは 192.168.99.100 です。
- この設定が使用する Booth チケットの名前は **apacheticket** です。

ここで使用するコマンド例は、Apache サービスのクラスターリソースが、各クラスターの **apachegroup** リソースグループの一部として設定されていることを前提としています。各クラスターの Pacemaker インスタンスは独立しているため、このリソースのチケット制約を設定するために、各クラスターでリソースとリソースグループが同じである必要はありませんが、これはフェイルオーバーの一般的な事例になります。

設定手順のどの時点でも、**pcs booth config** コマンドを実行すると、現在のノードまたはクラスターの Booth 設定を表示できます。また、**pcs booth status** コマンドを実行すると、ローカルノードの現在の Booth 状態を表示できます。

手順

1. **booth-site** Booth チケットマネージャーパッケージを、両方のクラスターの各ノードにインストールします。

```
[root@cluster1-node1 ~]# dnf install -y booth-site
[root@cluster1-node2 ~]# dnf install -y booth-site
[root@cluster2-node1 ~]# dnf install -y booth-site
[root@cluster2-node2 ~]# dnf install -y booth-site
```

2. **pcs** パッケージ、**booth-core** パッケージ、および **booth-arbitrator** パッケージを仲裁ノードにインストールします。

```
[root@arbitrator-node ~]# dnf install -y pcs booth-core booth-arbitrator
```

3. **firewalld** デーモンを実行している場合は、両方のクラスターの全ノードと、仲裁ノードで以下のコマンドを実行し、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 --add-service=high-availability
```

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。Red Hat High-Availability Add-On で必要なポートの詳細は、High Availability Add-On のポートの有効化 を参照してください。

4. 1 つのクラスターの 1 つのノードで Booth 設定を作成します。各クラスターおよび仲裁ノードに指定するアドレスは IP アドレスでなければなりません。各クラスターには Floating IP アドレスを指定します。

```
[cluster1-node1 ~] # pcs booth setup sites 192.168.11.100 192.168.22.100 arbitrators
192.168.99.100
```

このコマンドを実行すると、設定ファイルの **/etc/booth/booth.conf** および **/etc/booth/booth.key** がノードに作成されます。

5. Booth 設定のチケットを作成します。このチケットは、このチケットがクラスターに付与された場合のみリソースの実行を許可するリソース抑制を定義するのに使用します。
このフェイルオーバー設定手順は基本的な手順で、チケットを 1 つだけ使用します。各チケットが別の 1 つ以上のリソースに関連付けられる、より複雑な事例では追加のチケットを作成します。

```
[cluster1-node1 ~] # pcs booth ticket add apacheticket
```

6. 現在のクラスターのすべてのノードに対して Booth 設定を同期します。

```
[cluster1-node1 ~] # pcs booth sync
```

7. 仲裁ノードから、Booth 設定を仲裁者へプルします。この作業をこれまで行ったことがない場合は、最初に、設定をプルするノードに **pcs** を認証する必要があります。

```
[arbitrator-node ~] # pcs host auth cluster1-node1
[arbitrator-node ~] # pcs booth pull cluster1-node1
```

8. Booth 設定を別のクラスターにプルし、そのクラスターのすべてのノードを同期します。仲裁ノードでこの作業を行ったことがない場合は、最初に、設定をプルするノードに **pcs** を認証する必要があります。

```
[cluster2-node1 ~] # pcs host auth cluster1-node1
[cluster2-node1 ~] # pcs booth pull cluster1-node1
[cluster2-node1 ~] # pcs booth sync
```

9. 仲裁ノードで Booth を開始して有効にします。

注記

Booth はクラスターで Pacemaker リソースとして実行するため、クラスターのノードで Booth を手動で開始したり有効にしたりしないでください。

```
[arbitrator-node ~] # pcs booth start
[arbitrator-node ~] # pcs booth enable
```

10. 各クラスターに割り当てられた Floating IP アドレスを使用して、両方のクラスターサイトでクラスターリソースとして実行されるように Booth を設定します。**booth-ip** および **booth-service** をグループのメンバーとするリソースグループが作成されます。

```
[cluster1-node1 ~] # pcs booth create ip 192.168.11.100
[cluster2-node1 ~] # pcs booth create ip 192.168.22.100
```

11. 各クラスターに定義したリソースグループにチケット制約を追加します。

```
[cluster1-node1 ~] # pcs constraint ticket add apacheticket apachegroup
[cluster2-node1 ~] # pcs constraint ticket add apacheticket apachegroup
```

次のコマンドを実行すると、現在設定されているチケット制約を表示できます。

```
pcs constraint ticket [show]
```

12. この設定用に作成したチケットを最初のクラスターに付与します。
チケットを付与する前にチケット抑制を定義する必要はありません。最初にチケットをクラスターに付与した後、**pcs booth ticket revoke** コマンドで手動でオーバーライドしない限り、Booth はチケットの管理を引き継ぎます。**pcs booth** 管理コマンドの詳細は、**pcs booth** コマンドの PCS ヘルプ画面を参照してください。

```
[cluster1-node1 ~] # pcs booth ticket grant apacheticket
```

チケットは、いつでも (この手順の完了後でも) 追加および削除できます。ただし、チケットを追加または削除した後、この手順の説明どおりに、他のノード、クラスター、および仲裁ノードに対して設定ファイルを同期し、チケットを付与する必要があります。

Booth 設定ファイル、チケット、およびリソースのクリーンアップや削除に使用できるその他の Booth 管理コマンドに関する情報は、**pcs booth** コマンドの PCS ヘルプ画面を参照してください。

第30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、そのリソースが実際のクラスターノードであるかのように、クラスターがリソースを管理できます。

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、Red Hat サポート制限である 32 ノード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。

- **クラスターノード** - 高可用性サービスを実行しているノード (**pacemaker** および **corosync**)。
- **リモートノード** - **pacemaker_remote** を実行して、**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。
- **ゲストノード** - **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。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 KVM ゲストノード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
- フェンシング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、クラスター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。

このような制限事項以外に、リモートノードとゲストノードは、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、リモートノードとゲストノード自体をフェンスすることができます。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。このようなノードに制約を作成したり、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。リモートノード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。

30.1. PACEMAKER_REMOTE ノードのホストおよびゲストの認証

クラスターノードと pacemaker_remote の間の接続には、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 3121 ポートを使用) でセキュア化されます。そのため、クラスターノードと、**pacemaker_remote** を実行しているノードは、同じ秘密鍵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** を設定し、**pcs cluster node add-remote** コマンドは、リモートノードに **authkey** を設定します。

30.2. 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。

30.2.1. 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、仮想ドメインリソースのオプション の表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。以下の表は、これらのメタデータオプションについて説明しています。

表30.1 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**: この値を、リソースやノードの ID と重複させることはできません。
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

30.2.2. 仮想マシンのゲストノードとしての統合

以下の手順では、**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手順の概要を説明します。

手順

1. **VirtualDomain** リソースを設定します。
2. すべての仮想マシンで次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 3121 ポートを許可します。

```
# dnf 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。
4. ゲストノードとして統合しようとしているノードに **pcs** を認証していない場合は認証します。

```
# pcs host auth nodename
```

5. 次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、追加するゲストノードではなく、クラスターノードで実行する必要があります。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** デーモンを起動して有効にします。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場合があります。

```
# pcs cluster node add-guest nodename resource_id [options]
```

6. **VirtualDomain** リソースの作成後は、クラスターの他のノードと同じように、ゲストノードを扱うことができます。たとえば、クラスターノードから実行される次のコマンドのように、リソースを作成し、リソースにリソース制約を設定してゲストノードで実行できます。ゲストノードはグループに追加できます。これにより、ストレージデバイス、ファイルシステム、および仮想マシン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
```

30.3. 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。

30.3.1. リモートノードリソースのオプション

以下の表は、**remote** リソースに設定できるリソースオプションを示しています。

表30.2 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)。この待機期間は繰り返し発生します。待機期間の後に再接続に失敗した場合、待機期間の後に、新しい再接続が試行されます。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。

30.3.2. リモートノードの設定の概要

以下のセクションでは、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。

手順

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 のポート 2224 および 3121 を開きます。

2. リモートノードに、**pacemaker_remote** デーモンをインストールします。

```
# dnf 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. リモートノードとして追加するノードに **pcs** を認証していない場合は、認証します。

```
# 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、これをシステムの起動時に **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、追加するリモートノードではなく、クラスターノードで実行する必要があります。

```
# pcs cluster node add-remote remote1
```

6. **remote** リソースをクラスターに追加した後、リモートノードを、クラスター内の他のノードを処理するのと同じように処理できます。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じ方法でフェンスされます。クラスターノードと同様に、リモートノードで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シングアクションを開始できないことに注意してください。クラスターノードのみが、実際に別のノードに対してフェンシング操作を実行できます。

30.4. 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** 環境変数を設定できます。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で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)。

30.5. 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るようになりました。ただし、**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。リソースの監視タイムアウトが発生する前に **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。

手順

1. ノードからすべてのサービスを除去する **pcs resource disable resourcename** コマンドを使用して、ノードの接続リソースを停止します。接続リソースは、リモートノードの場合は **ocf:pacemaker:remote** リソース、通常はゲストノードの場合は **ocf:heartbeat:VirtualDomain** リソースになります。ゲストノードの場合、このコマンドは VM も停止するため、メンテナンスを実行するには、クラスターの外部で (たとえば、**virsh** を使用して) VM を起動する必要があります。

   ```
   pcs resource disable resourcename
   ```

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** コマンドでリソースを再度有効にします。

   ```
   pcs resource enable resourcename
   ```

第31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、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様々な方法を提供します。

- 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することができます。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、別のノードに移行するか、(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。スタンバイモードの詳細は、ノードをスタンバイモードに を参照してください。

- リソースを停止せずに、現在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** コマンドまたは **pcs constraint delete** コマンドを実行して制約を削除できます。ただし、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。その時点でリソースが実行できる場所は、リソースを最初に設定した方法によって異なるためです。**pcs resource relocate run** コマンドを使用すると、リソースを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無効化、有効化、および禁止 を参照してください。

- Pacemaker が、リソースに対して何らかのアクションを実行しないようにする場合 (たとえば、リソースのメンテナンス中に復元アクションを無効にする場合や、**/etc/sysconfig/pacemaker** 設定をリロードする必要がある場合) は、リソースの非管理モードへの設定 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、非管理モードにしないでください。

- クラスターを、サービスの開始や停止が行われ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非管理モードになります。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 を参照してください。

- RHEL High Availability Add-On および Resilient Storage Add-On に含まれるパッケージを更新する必要がある場合は、RHEL 高可用性クラスターの更新 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、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。

- RHEL クラスターで仮想マシンを移行する必要がある場合は、RHEL クラスターでの仮想マシンの移行 で説明するように、まず仮想マシンでクラスターサービスを停止してクラスターからノードを削除し、移行後にクラスターのバックアップを開始する必要があります。

31.1. 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になると、ノードが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リソースは、すべて別のノードに移行されます。

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** を指定すると、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。

このコマンドは、リソースのパッケージを更新する場合に使用できます。また、設定をテストして、ノードを実際にシャットダウンせずに復元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
```

次のコマンドは、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、指定されたノードでリソースが実行されないことに注意してください。**pcs node unstandby** コマンドを実行すると、指定されたノードでリソースを実行できます。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。その時点でリソースが実行できる場所は、リソースを最初に設定した方法によって異なります。

31.2. 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行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。

- ノードがメンテナンスで、そのノードで実行中の全リソースを別のノードに移行する必要がある
- 個別に指定したリソースを移行する必要がある

ノードで実行中の全リソースを別のノードに移行する場合は、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、以下のいずれかの方法で移行できます。

- **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、リソースを移行します。

31.2.1. 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て、リソースの **resource_id** を定義どおりに指定します。移行するリソースを実行する移行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] [--promoted] [--strict] [--wait[=n]]
```

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。デフォルトでは、リソースを移動すると、コマンドが作成する場所の制約が自動的に削除されます。リソースの **resource-stickiness** 値が 0 の場合のように、制約を削除するとリソースが元のノードに戻る場合、**pcs resource move** コマンドは失敗します。リソースを移動し、その制約を適用したままにする場合は、**pcs resource move-with-constraint** を使用します。

pcs resource move コマンドで **--promoted** パラメーターを指定すると、制約はリソースの昇格されたインスタンスにのみ適用されます。

pcs resource move コマンドに **--strict** パラメーターを指定した場合は、コマンドで指定したリソース以外のリソースに影響を与えるとコマンドが失敗します。

任意で、**pcs resource move** コマンドに **--wait[=n]** パラメーターを設定し、移行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 を指定しない場合は、デフォルト値の 60 分になります。

31.2.2. リソースを優先ノードへ移行

フェイルオーバーや管理者の手作業によるノードの移行により、リソースが移行した後、フェイルオーバーの原因となった状況が改善されたとしても、そのリソースが必ずしも元のノードに戻るとは限りません。リソースを優先ノードへ移行するには、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行します。

このコマンドは、リソースのスティッキネスを無視し、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行する場所の制約を作成します。リソースが移行すると、制約が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、リソースのスティッキネスを無視した最適なノードを表示する場合は、**pcs resource relocate show** コマンドを実行します。

31.3. クラスターリソースの無効化、有効化、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のに使用できる様々なコマンドがあります。

クラスターリソースの無効化
実行中のリソースを手動で停止し、クラスターが再起動しないようにする場合は、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 秒間待機します。その後、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 を指定しないと、デフォルトの 60 分に設定されます。

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

リソースを無効にしても、他のリソースに影響が及ばない場合に限り、リソースを無効にできます。これを確認することは、複雑なリソース関係が設定されている場合は手作業では不可能です。

- **pcs resource disable --simulate** コマンドは、クラスター設定を変更せずに、リソースを無効にする効果を表示します。
- **pcs resource disable --safe** コマンドは、あるノードから別のノードに移行されるなど、その他のリソースが何らかの影響を受けない場合にのみリソースを無効にします。**pcs resource safe-disable** コマンドは、**pcs resource disable --safe** コマンドのエイリアスです。

- **pcs resource disable --safe --no-strict** コマンドは、他のリソースが停止または降格されない場合に限りリソースを無効にします。

pcs resource disable --safe コマンドに **--brief** オプションを指定して、エラーのみを出力できます。安全な無効化操作に失敗した場合に **pcs resource disable --safe** コマンドが生成するエラーレポートには、影響を受けるリソース ID も含まれます。リソースの無効化によって影響を受けるリソースのリソース ID のみを把握する必要がある場合は、**--brief** オプションを使用します。これにより、詳細なシミュレーション結果は提供されません。

クラスターリソースの有効化

クラスターがリソースを起動できるようにするには、次のコマンドを使用します。他の設定によっては、リソースが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 はリソースが開始するまで最長で n 秒間待機します。その後、リソースが開始した場合には 0、リソースが開始しなかった場合には 1 を返します。n を指定しないと、デフォルトの 60 分に設定されます。

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

特定のノードでリソースが実行されないようにする

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

```
pcs resource ban resource_id [node] [--promoted] [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** 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。その時点でリソースが実行できる場所は、リソースを最初に設定した方法によって異なります。

pcs resource ban コマンドに **--promoted** パラメーターを指定すると、制約の有効範囲はプロモートロールに限定され、resource_id ではなく promotable_id を指定する必要があります。

任意で **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。

任意で、**pcs resource ban** コマンドに **--wait[=n]** パラメーターを設定し、移行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) 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のタイムアウト値が使用されます。

現在のノードでリソースを強制的に起動

指定したリソースを現在のノードで強制的に起動する場合は、**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。これは、主にデバッグリソースに使用されます。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原因は、大抵、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に制約が設定されているか、リソースが無効になっているかのいずれかになります。この場合は、次のコマンドを使用してリソースの設定をテストできます。ただし、通常は、クラスター内でリソースを起動するのに使用しないでください。

debug-start コマンドの形式は以下のようになります。

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

31.4. リソースの非管理モードへの設定

リソースが **非管理** モードの場合、リソースは引き続き設定に含まれますが、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 **非管理** 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 **管理** 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。このコマンドは、グループのすべてのリソースに対して実行されるため、1つのコマンドでグループ内の全リソースすべて **管理** または **非管理** モードに設定し、グループに含まれるリソースを個別に管理できます。

31.5. 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、サービスを開始したり、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、開始します。

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、次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、**symmetric-cluster** プロパティーを **false** に設定したことがある場合は、設定した値が次のコマンドにより削除され、**symmetric-cluster** の値がデフォルト値の **true** に戻されます。

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

31.6. RHEL 高可用性クラスターの更新

RHEL High Availability Add-On および Resilient Storage Add-On を設定するパッケージを、個別または一括で更新するには、以下に示す一般的な方法のいずれかを使用できます。

- **ローリング更新** - 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、そのノードをクラスターに戻します。これにより、各ノードの更新中も、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 の High Availability クラスターおよび Resilient Storage クラスターのソフトウェア更新手順を実行する場合は、更新を開始する前に、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 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更新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。

31.7. 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** サービスが停止すると、クラスターは、ノードを停止する前に、ノードからリソースを適切に移行します。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るようになりました。ただし、**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。リソースの監視タイムアウトが発生する前に **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。

手順

1. ノードからすべてのサービスを除去する **pcs resource disable resourcename** コマンドを使用して、ノードの接続リソースを停止します。接続リソースは、リモートノードの場合は **ocf:pacemaker:remote** リソース、通常はゲストノードの場合は **ocf:heartbeat:VirtualDomain** リソースになります。ゲストノードの場合、このコマンドは VM も停止するため、メンテナンスを実行するには、クラスターの外部で (たとえば、**virsh** を使用して) VM を起動する必要があります。

```
pcs resource disable resourcename
```

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** コマンドでリソースを再度有効にします。

```
pcs resource enable resourcename
```

31.8. RHEL クラスターでの仮想マシンの移行

Support Policies for RHEL High Availability Clusters - General Conditions with Virtualized Cluster Members の説明にあるように、Red Hat ではハイパーバイザーまたはホスト全体でのアクティブなクラスターノードのライブマイグレーションはサポートしていません。ライブマイグレーションを実行する必要がある場合は、まず仮想マシンでクラスターサービスを停止してクラスターからノードを削除し、移行後にクラスターのバックアップを開始する必要があります。以下の手順では、クラスターから仮想マシンを削除し、仮想マシンを移行し、クラスターに仮想マシンを復元する手順の概要を説明します。

以下の手順では、クラスターから仮想マシンを削除し、仮想マシンを移行し、クラスターに仮想マシンを復元する手順の概要を説明します。

以下の手順では、全クラスターノードとして使用する仮想マシンが対象で、特別な配慮なしでライブマイグレーションが可能なクラスターリソースとして管理される仮想マシン (例: ゲストノードとして使用する仮想マシン) は対象外です。RHEL High Availability および Resilient Storage Add-On を設定するパッケージを更新するのに必要な一般的な手順は、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更新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。

注記

この手順を実行する前に、クラスターノードの削除がクラスタークォーラム (定足数) に与える影響を考慮してください。たとえば、3 ノードクラスターがあり、1つのノードを削除すると、クラスターはノードの障害に耐えられなくなります。これは、3 ノードクラスターのノードがすでに1つダウンしている場合、2 つ目のノードを削除するとクォーラムが失われるためです。

手順

1. 移行する仮想マシンで実行しているリソースやソフトウェアの停止または移動を行う前に準備を行う必要がある場合は、以下の手順を実行します。
2. 仮想マシンで以下のコマンドを実行し、仮想マシン上のクラスターソフトウェアを停止します。

```
# pcs cluster stop
```

3. 仮想マシンのライブマイグレーションを実行します。
4. 仮想マシンでクラスターサービスを起動します。

```
# pcs cluster start
```

31.9. UUID によるクラスターの識別

Red Hat Enterprise Linux 9.1 では、作成されたクラスターには関連する UUID があります。クラスター名は一意のクラスター識別子ではないため、同じ名前の複数のクラスターを管理する設定管理データベースなどのサードパーティーツールは、その UUID によってクラスターを一意に識別できます。現在のクラスター UUID は、**pcs cluster config [show]** コマンドで表示できます。このコマンドの出力には、クラスター UUID が含まれています。

UUID を既存のクラスターに追加するには、次のコマンドを実行します。

```
# pcs cluster config uuid generate
```

既存の UUID でクラスターの UUID を再生成するには、次のコマンドを実行します。

```
# pcs cluster config uuid generate --force
```

第32章 障害復旧クラスターの設定

高可用性クラスターに障害復旧を提供する1つの方法は、2 つのクラスターを設定することです。次に、1つのクラスターをプライマリーサイトクラスターとして設定し、2 番目のクラスターを災害復旧クラスターとして設定できます。

通常の場合、プライマリークラスターは実稼働モードでリソースが実行されている状態です。障害復旧クラスターにはすべてのリソースが設定されており、それらを降格モードで実行するか、全く実行しません。たとえば、昇格モードでプライマリークラスターで実行され、降格モードで障害復旧クラスターで実行されているデータベースがあるとします。この設定のデータベースが、プライマリーから障害復旧サイトにデータが同期されるように設定されます。これは、**pcs** コマンドインターフェイスではなく、データベース設定自体で行われます。

プライマリークラスターがダウンした場合、ユーザーは **pcs** コマンドラインインターフェイスを使用して、手動でリソースを障害復旧サイトにフェイルオーバーできます。その後、障害のサイトにログインして、そのサイトを昇格し、そこでリソースを起動できます。プライマリークラスターが復元した後、**pcs** コマンドラインインターフェイスを使用してリソースをプライマリーサイトに手動で移動できます。

pcs コマンドを使用して、いずれかのサイトの1つのノードから、プライマリーおよび障害復旧サイトクラスターの両方のステータスを表示できます。

32.1. 障害復旧クラスターに関する考慮事項

pcs コマンドラインインターフェイスで管理および監視を行う障害復旧サイトの計画および設定を行うときは、次の考慮事項に注意してください。

- 障害復旧サイトはクラスターである必要があります。これにより、プライマリーサイトと同じツールや同様の手順で設定できるようになります。
- プライマリークラスターおよび障害復旧クラスターは、独立した **pcs cluster setup** コマンドで作成されます。
- クラスターとそのリソースは、データが同期され、フェイルオーバーが可能になるように設定する必要があります。
- 復旧サイトのクラスターノードには、プライマリーサイトのノードと同じ名前を持つことができません。
- **pcs** コマンドを実行するノードの両方のクラスターに対して、pcs ユーザー **hacluster** が認証されている必要があります。

32.2. 復旧クラスターの状態の表示

両方のクラスターのステータスを表示できるように、プライマリークラスターおよび障害復旧クラスターを設定するには、次の手順を実行します。

注記

障害復旧クラスターを設定すると、自動的にリソースを設定したり、データを複製したりしません。これらのアイテムはユーザーが手動で設定する必要があります。

この例では、以下のように設定されています。

- プライマリークラスターは **PrimarySite** という名前で、**z1.example.com** ノードと **z2.example.com** ノードで構成されます。
- 障害復旧サイトのクラスターは **DRsite** という名前で、**z3.example.com** ノードおよび **z4.example.com** ノードで構成されます。

この例では、リソースやフェンスが設定されていない基本的なクラスターを設定します。

手順

1. 両方のクラスターで使用されるすべてのノードを認証します。

```
[root@z1 ~]# pcs host auth z1.example.com z2.example.com z3.example.com
z4.example.com -u hacluster -p password
z1.example.com: Authorized
z2.example.com: Authorized
z3.example.com: Authorized
z4.example.com: Authorized
```

2. プライマリークラスターとして使用されるクラスターを作成し、クラスターのクラスターサービスを開始します。

```
[root@z1 ~]# pcs cluster setup PrimarySite z1.example.com z2.example.com --start
{...}
Cluster has been successfully set up.
Starting cluster on hosts: 'z1.example.com', 'z2.example.com'...
```

3. 障害復旧クラスターとして使用されるクラスターを作成し、クラスターのクラスターサービスを開始します。

```
[root@z1 ~]# pcs cluster setup DRSite z3.example.com z4.example.com --start
{...}
Cluster has been successfully set up.
Starting cluster on hosts: 'z3.example.com', 'z4.example.com'...
```

4. プライマリークラスターのノードから、2 つ目のクラスターを復旧サイトとして設定します。復旧サイトは、ノードのいずれかの名前により定義されます。

```
[root@z1 ~]# pcs dr set-recovery-site z3.example.com
Sending 'disaster-recovery config' to 'z3.example.com', 'z4.example.com'
z3.example.com: successful distribution of the file 'disaster-recovery config'
z4.example.com: successful distribution of the file 'disaster-recovery config'
Sending 'disaster-recovery config' to 'z1.example.com', 'z2.example.com'
z1.example.com: successful distribution of the file 'disaster-recovery config'
z2.example.com: successful distribution of the file 'disaster-recovery config'
```

5. 障害復旧設定を確認します。

```
[root@z1 ~]# pcs dr config
Local site:
  Role: Primary
Remote site:
  Role: Recovery
```

```
  Nodes:
    z3.example.com
    z4.example.com
```

6. プライマリークラスターのステータスと、プライマリークラスターのノードの障害復旧クラスターのステータスを確認します。

```
[root@z1 ~]# pcs dr status
--- Local cluster - Primary site ---
Cluster name: PrimarySite

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: z2.example.com (version 2.0.3-2.el8-2c9cea563e) - partition with quorum
  * Last updated: Mon Dec  9 04:10:31 2019
  * Last change:  Mon Dec  9 04:06:10 2019 by hacluster via crmd on z2.example.com
  * 2 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ z1.example.com z2.example.com ]

Full List of Resources:
  * No resources

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


--- Remote cluster - Recovery site ---
Cluster name: DRSite

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: z4.example.com (version 2.0.3-2.el8-2c9cea563e) - partition with quorum
  * Last updated: Mon Dec  9 04:10:34 2019
  * Last change:  Mon Dec  9 04:09:55 2019 by hacluster via crmd on z4.example.com
  * 2 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ z3.example.com z4.example.com ]

Full List of Resources:
  * No resources

Daemon Status:
```

```
corosync: active/disabled
pacemaker: active/disabled
pcsd: active/enabled
```

災害復旧設定の追加の表示オプションは、**pcs dr** コマンドのヘルプ画面を参照してください。

第33章 リソースエージェント OCF 戻りコードの解釈

Pacemaker リソースエージェントは Open Cluster Framework (OCF) Resource Agent API に準拠します。以下の表で、OCF 戻りコードと、Pacemaker がどのように解釈するかを説明します。

エージェントがコードを返したときに、クラスターがまず行うことは、期待されている結果通りにコードを返しているかどうかを確認します。そして、結果が期待されている値に一致しない場合、その操作が失敗したものとみなされ、復元操作が開始されます。

起動するには、起動した操作の結果の呼び出し元を通知する、定義した戻りコードでリソースエージェントを終了する必要があります。

以下の表で説明するように、障害回復には 3 つのタイプがあります。

表33.1 クラスターが行う復旧タイプ

タイプ	説明	クラスターが行った操作
軽度	一時的なエラーが発生しました。	リソースを再起動するか、新しい場所に移します。
重度	現在のノードに固有である可能性のある一時的ではないエラーが発生しました。	リソースを別の場所に移動し、現在のノードで再試行されないようにします。
致命的	すべてのクラスターノードに共有となる一時的でないエラーが発生しました (例: 指定された設定がよくありません)。	リソースを停止し、いかなるクラスターノードでも起動されないようにします。

次の表は、OCF 戻りコードと、失敗コードを受信したときにクラスターが開始するリカバリーのタイプを示しています。0 を返すアクション (OCF エイリアス **OCF_SUCCESS**) であっても、0 が予期された戻り値ではなかった場合には、失敗したと見なされる可能性があることに注意してください。

表33.2 OCF 戻りコード

戻りコード	OCF ラベル	説明
0	**OCF_SUCCESS**	* 操作が無事に完了しました。これは、起動、停止、昇格、降格コマンドに対して想定される戻りコードです。 * 予期しない場合のタイプ: ソフト
1	**OCF_ERR_GENERIC**	* この操作は、一般的なエラーを返しました。 * タイプ: ソフト * リソースマネージャーは、リソースの復元と、新しい場所への移動を試行します。

戻りコード	OCF ラベル	説明
2	**OCF_ERR_ARGS**	* このマシンのリソースの設定が正しくありません。たとえば、ノードで見つからない場所を参照しています。 * タイプ: 重度 * リソースマネージャーがリソースを別の場所に移動し、現在のノードで再試行されないようにします。
3	**OCF_ERR_UNIMPLEMENTED**	* 要求された操作が実装されていません。 * タイプ: 重度
4	**OCF_ERR_PERM**	* このリソースエージェントには、このタスクを完了するのに十分な特権がありません。これは、エージェントが特定のファイルを開けない場合や、特定のソケットでリッスンできない場合、ディレクトへの書き込みを行えない場合が考えられます。 * タイプ: 重度 * 特に設定されていない限り、リソースマネージャーは、別のノード (パーミッションが存在しない) でリソースを再起動することで、このエラーで失敗したリソースの復旧を試行します。
5	**OCF_ERR_INSTALLED**	* 操作が実行されたノードに、必要なコンポーネントが欠如しています。これは、必要なバイナリーが実行不可であるか、重要な設定ファイルが読み込み不可になっていることが原因の場合があります。 * タイプ: 重度 * 特に設定されていない限り、リソースマネージャーは、別のノード (必要なファイルまたはバイナリーが存在しない) でリソースを再起動することで、このエラーで失敗したリソースの復旧を試行します。
6	**OCF_ERR_CONFIGURED**	* ローカルノード上のリソースの設定が正しくありません。 * タイプ: 致命的 * このコードがかえされると、Pacemaker は、サービス設定がその他のノードで正しくても、クラスター内のノードでリソースが実行されないようにします。

戻りコード	OCF ラベル	説明
7	**OCF_NOT_RUNNING**	* このリソースは安全に停止します。これは、リソースが正常にシャットダウンされたか、起動されていないことを意味します。 * 予期しない場合のタイプ: ソフト * クラスターは、いかなる操作に対しても、これを返すリソースの停止を試行しません。
8	**OCF_RUNNING_PROMOTED**	* リソースは昇格されたロールで実行されています。 * 予期しない場合のタイプ: ソフト
9	**OCF_FAILED_PROMOTED**	* リソースは昇格されたロールにある (またはその可能性がある) が、失敗しています。 * タイプ: ソフト * リソースは降格され、停止して再起動されます (昇格される可能性があります)。
190		* サービスが適切にアクティブな状態であることが確認されましたが、今後障害が発生することが予想される状況です。
191		* リソースエージェントがロールをサポートし、サービスが昇格されたロールで適切にアクティブな状態であることが確認されましたが、今後障害が発生することが予想される状況です。
その他	該当なし	カスタムエラーコード

第34章 クラスターメンバーとして IBM Z/VM インスタンスを使用した RED HAT HIGH AVAILABILITY クラスターの設定

Red Hat は、z/VM 仮想マシンで実行する Red Hat High Availability クラスターの設計、設定、および管理に便利な記事を複数提供しています。

- Design Guidance for RHEL High Availability Clusters - IBM z/VM Instances as Cluster Members
- RHEL High Availability クラスターの管理手順 -RHEL7 または 8 IBM z Systems クラスターメンバー用の fence_zvmip を使用した z/VM SMAPI フェンシングの設定
- RHEL High Availability cluster nodes on IBM z Systems experience STONITH-device timeouts around midnight on a nightly basis
- Administrative Procedures for RHEL High Availability Clusters - Preparing a dasd Storage Device for Use by a Cluster of IBM z Systems Members

また、Red Hat High Availability クラスターの設計時に便利な、以下の記事も提供しています。

- RHEL 高可用性クラスターのサポートポリシー
- Exploring Concepts of RHEL High Availability Clusters - Fencing/STONITH