Red Hat Enterprise Linux 9

고가용성 클러스터 구성 및 관리

Red Hat High Availability Add-On을 사용하여 Pacemaker 클러스터 생성 및 유지 관리

Last Updated: 2024-07-19

Red Hat Enterprise Linux 9 고가용성 클러스터 구성 및 관리

Red Hat High Availability Add-On을 사용하여 Pacemaker 클러스터 생성 및 유지 관리

법적 공지

초록

Red Hat High Availability Add-On은 Pacemaker 클러스터 리소스 관리자를 사용하는 고가용성 클러스터를 구성합니다. 이 제목에서는 Pacemaker 클러스터 설정 및 활성/활성/수동/수동 절차를 숙지하는 절차를 제공합니다.

차례

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

Jira를 통해 피드백 제출 (등록 필요)

1. Jira 웹 사이트에 로그인합니다.
2. 상단 탐색 모음에서 **생성** 을 클릭합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. 대화 상자 하단에서 **생성** 을 클릭합니다.

1장. 고가용성 애드온 개요

고가용성 애드온은 중요한 프로덕션 서비스에 대한 안정성, 확장성 및 가용성을 제공하는 클러스터 기반 시스템입니다.

클러스터는 작업을 수행하기 위해 함께 작동하는 두 개 이상의 컴퓨터(노드 또는 *멤버*라고 함)입니다. 클러스터를 사용하여 고가용성 서비스 또는 리소스를 제공할 수 있습니다. 여러 머신의 중복은 여러 유형의 오류를 보호하는 데 사용됩니다.

고가용성 클러스터는 단일 장애 지점을 제거하고 노드가 작동하지 않는 경우 한 클러스터 노드에서 다른 클러스터 노드로 서비스를 장애 조치하여 고가용성 서비스를 제공합니다. 일반적으로 고가용성 클러스터의 서비스 읽기 및 쓰기 데이터(Read-write 마운트된 파일 시스템 등) 따라서 한 클러스터 노드에서 다른 클러스터 노드에서 서비스를 제어하므로 고가용성 클러스터는 데이터 무결성을 유지해야 합니다. 고가용성 클러스터의 노드 오류는 클러스터 외부의 클라이언트에서 표시되지 않습니다. (고가용성 클러스터를 장애 조치 클러스터라고도 합니다.) 고가용성 애드온은 **Pacemaker** 의 고가용성 서비스 관리 구성 요소를 통해 고가용성 클러스터링을 제공합니다.

Red Hat은 Red Hat 고가용성 클러스터를 계획, 구성 및 유지 관리하기 위한 다양한 문서를 제공합니다. Red Hat 클러스터 문서의 다양한 영역에 가이드 인덱스를 제공하는 문서 목록은 Red Hat High Availability Add-On 설명서 를 참조하십시오.

1.1. 고가용성 애드온 구성 요소

Red Hat High Availability Add-On은 고가용성 서비스를 제공하는 여러 구성 요소로 구성되어 있습니다.

고가용성 애드온의 주요 구성 요소는 다음과 같습니다.

- 클러스터 인프라 - 구성 파일 관리, 멤버십 관리, 잠금 관리 및 펜싱과 같은 클러스터로서 노드가 함께 작동할 수 있는 기본 기능을 제공합니다.
- 고가용성 서비스 관리 - 노드가 작동하는 경우 한 클러스터 노드에서 다른 클러스터 노드로의 서비스 장애 조치를 제공합니다.
- 클러스터 관리 툴 - 고가용성 애드온을 설정, 구성 및 관리하기 위한 구성 및 관리 툴입니다. 툴은 클러스터 인프라 구성 요소, 고가용성 및 서비스 관리 구성 요소, 스토리지와 함께 사용할 수 있습니다.

고가용성 애드온을 다음 구성 요소로 보완할 수 있습니다.

- Red Hat GFS2 (Global File System 2) - 복구 스토리지 애드온의 일부인 고가용성 애드온과 함께 사용할 클러스터 파일 시스템을 제공합니다. GFS2를 사용하면 스토리지가 각 클러스터 노드에 로컬로 연결된 것처럼 블록 수준에서 여러 노드가 스토리지를 공유할 수 있습니다. GFS2 클러스터 파일 시스템에는 클러스터 인프라가 필요합니다.
- LVM 잠금 데몬(**lvmlockd**) - 복구 스토리지 애드온의 일부로 클러스터 스토리지의 볼륨 관리 기능을 제공합니다. **lvmlockd** 지원도 클러스터 인프라가 필요합니다.
- HAProxy - 고가용성 로드 밸런싱 및 계층 4(TCP) 및 계층 7(HTTP, HTTPS) 서비스에서의 페일오버를 제공하는 라우팅 소프트웨어.

1.2. 고가용성 애드온 개념

Red Hat High Availability Add-On 클러스터의 주요 개념은 다음과 같습니다.

1.2.1. 펜싱

클러스터의 단일 노드와 통신하는 데 실패하는 경우 클러스터의 다른 노드는 실패한 클러스터 노드에서 액세스할 수 있는 리소스에 대한 액세스를 제한하거나 해제할 수 있어야 합니다. 클러스터 노드가 응답하지 않을 수 있으므로 클러스터 노드에 연결하여 수행할 수 없습니다. 대신 차단 에이전트를 사용한 펜싱이라는 외부 방법을 제공해야 합니다. 펜스 장치는 클러스터에서 잘못된 노드의 공유 리소스에 대한 액세스를 제한하거나 클러스터 노드에서 하드 재부팅을 발행하는 데 사용할 수 있는 외부 장치입니다.

펜스 장치를 구성하지 않으면 연결이 끊긴 클러스터 노드에서 이전에 사용한 리소스가 해제되어 다른 클러스터 노드에서 서비스가 실행되지 않을 수 있습니다. 반대로, 시스템은 클러스터 노드가 리소스를 해제한 것으로 잘못 가정하고 이로 인해 데이터 손상 및 데이터 손실이 발생할 수 있습니다. 펜스 장치가 구성된 데이터 무결성을 보장할 수 없으며 클러스터 구성은 지원되지 않습니다.

펜싱이 진행 중인 경우 다른 클러스터 작업을 실행할 수 없습니다. 클러스터 노드가 재부팅된 후 펜싱이 완료되거나 클러스터 노드가 클러스터에 다시 참여할 때까지 클러스터의 정상적인 작동을 재개할 수 없습니다.

펜싱에 대한 자세한 내용은 Red Hat High Availability Cluster에서의 펜싱을 참조하십시오.

1.2.2. 쿼럼

클러스터 시스템은 클러스터 무결성 및 가용성을 유지하기 위해 *쿼럼* 이라는 개념을 사용하여 데이터 손상 및 손실을 방지합니다. 클러스터 노드의 절반 이상이 온라인 상태이면 클러스터에 쿼럼이 있습니다. 오류로 인해 데이터 손상 가능성을 줄이기 위해 Pacemaker는 클러스터에 쿼럼이 없는 경우 기본적으로 모든 리소스를 중지합니다.

쿼럼은 투표 시스템을 사용하여 설정됩니다. 클러스터 노드가 나머지 클러스터와의 통신으로 작동하지 않거나 다른 클러스터와의 통신이 손실되면 대부분의 작업 노드에서 투표하여 서비스를 위해 노드를 펜싱할 수 있습니다.

예를 들어 6 노드 클러스터에서 쿼럼은 4개 이상의 클러스터 노드가 작동할 때 설정됩니다. 대다수의 노드가 오프라인 상태가 되거나 사용 불가능한 경우 클러스터에 쿼럼이 없으며 Pacemaker는 클러스터형 서비스를 중지합니다.

Pacemaker의 쿼럼 기능은 클러스터가 통신과 분리되는 경우도 있지만 각 파트는 동일한 데이터에 쓰기가 계속되어 손상이나 손실이 발생할 수 있는 경우도 있습니다. *분할 상태에 있다는 의미와 일반적으로 쿼럼 개념에 대한 자세한 내용은 RHEL High Availability Clusters - Quorum의 개념 탐색을 참조하십시오.*

*Red Hat Enterprise Linux High Availability Add-On 클러스터는 분리되는 문제를 방지하기 위해 펜싱과 함께 **votequorum** 서비스를 사용합니다. 클러스터의 각 시스템에 투표가 할당되고 대부분의 투표가 있는 경우에만 클러스터 작업을 진행할 수 있습니다.*

1.2.3. 클러스터 리소스

클러스터 리소스 는 클러스터 서비스에서 관리할 프로그램, 데이터 또는 애플리케이션의 인스턴스입니다. 이러한 리소스는 클러스터 환경에서 리소스를 관리하기 위한 표준 인터페이스를 제공하는 에이전트에 의해 추상화됩니다.

리소스가 정상 상태로 유지되도록 모니터링 작업을 리소스 정의에 추가할 수 있습니다. 리소스에 대한 모니터링 작업을 지정하지 않으면 기본적으로 하나씩 추가됩니다.

제약 조건을 구성하여 클러스터의 리소스 동작을 확인할 수 있습니다. 다음 유형의 제약 조건을 구성할 수 있습니다.

- *위치 제한 조건 - 리소스가 실행할 수 있는 노드를 결정합니다.*
- *순서 지정 제약 조건 - 순서 지정 제약 조건으로 인해 리소스가 실행되는 순서가 결정됩니다.*

- *공동 배치 제약 조건 - 공동 배치 제한 조건은 다른 리소스에 대해 리소스를 배치할 위치를 결정합니다.*

클러스터의 가장 일반적인 요소 중 하나는 함께 배치하고 순차적으로 시작하고 역방향 순서로 중지해야 하는 리소스 집합입니다. 이 구성을 단순화하기 위해 Pacemaker는 그룹 의 개념을 지원합니다.

1.3. PACEMAKER 개요

Pacemaker는 클러스터 리소스 관리자입니다. 이는 클러스터 인프라의 메시징 및 멤버십 기능을 사용하여 노드 및 리소스 수준 장애를 완화하여 클러스터 서비스 및 리소스에 대한 가용성을 극대화합니다.

1.3.1. Pacemaker 아키텍처 구성 요소

Pacemaker로 구성된 클러스터는 클러스터 멤버십을 모니터링하는 별도의 구성 요소 데몬, 서비스를 관리하는 스크립트, 분산된 리소스를 모니터링하는 리소스 관리 하위 시스템으로 구성됩니다.

다음 구성 요소는 Pacemaker 아키텍처를 형성합니다.

CIB(Cluster Information Base)

내부적으로 XML 을 사용하여 CIB를 통해 CIB - CIB를 통해 클러스터 상태 및 작업을 저장하고 배포하기 위해 Pacemaker에서 할당한 노드의 현재 구성 및 상태 정보를 배포하고 동기화합니다.

클러스터 리소스 관리 데몬(CRMd)

Pacemaker 클러스터 리소스 작업은 이 데몬을 통해 라우팅됩니다. CRMd에서 관리하는 리소스는 클라이언트 시스템에 의해 쿼리되고 이동, 인스턴스화 및 필요에 따라 변경할 수 있습니다.
각 클러스터 노드에는 CRMd와 리소스 간의 인터페이스 역할을 하는 로컬 리소스 관리자 데몬(LRMd)도 포함되어 있습니다. LRMd는 상태 정보 시작 및 정지 및 릴레이와 같은 CRMd에서 에이전트에 명령을 전달합니다.

제목에서 다른 노드 촬영 (STONITH)

STONITH는 Pacemaker 펜싱 구현입니다. Pacemaker에서 요청을 처리하고 노드에서 강제 종료한 후 클러스터에서 해당 리소스를 제거하여 데이터 무결성을 보장하는 클러스터 리소스의 역할을 합니다. STONITH는 CIB에서 구성되며 일반 클러스터 리소스로 모니터링할 수 있습니다.

Corosync

***Corosync는** 구성 요소이며 동일한 이름의 데몬으로, 고가용성 클러스터에 대한 핵심 멤버십 및 멤버 신뢰 요구 사항을 제공합니다. 고가용성 애드온이 작동해야 합니다.*
*이러한 멤버십 및 메시징 기능 외에도 **corosync** 도 마찬가지입니다.*

- *쿼럼 규칙을 관리하고 결정을 내립니다.*
- *클러스터의 여러 멤버에서 조정하거나 작동하는 애플리케이션에 대한 메시징 기능을 제공하므로 인스턴스 간 상태 저장 또는 기타 정보를 통신해야 합니다.*
- ***kronosnet** 라이브러리를 네트워크 전송으로 사용하여 여러 중복 링크 및 자동 장애 조치를 제공합니다.*

1.3.2. Pacemaker 구성 및 관리 툴

고가용성 애드온은 클러스터 배포, 모니터링 및 관리를 위한 두 가지 구성 툴을 제공합니다.

pcs

***pcs** 명령줄 인터페이스에서 Pacemaker 및 **corosync** 하트비트 데몬을 제어하고 구성합니다. 명령줄 기반 프로그램 **pcs** 는 다음 클러스터 관리 작업을 수행할 수 있습니다.*

- *Pacemaker/Corosync 클러스터 생성 및 구성*
- *실행 중인 클러스터 설정 수정*
- *Pacemaker와 Corosync 둘 다와 클러스터의 시작, 중지 및 표시 상태 정보를 원격으로 구성합니다.*

pcsd 웹 UI

Pacemaker/Corosync 클러스터를 생성하고 구성하는 그래픽 사용자 인터페이스입니다.

1.3.3. 클러스터 및 Pacemaker 구성 파일

*Red Hat High Availability Add-On의 구성 파일은 **corosync.conf** 및 **cib.xml** 입니다.*

***corosync.conf** 파일은 Pacemaker가 구축된 클러스터 관리자인 **corosync** 에서 사용하는 클러스터 매개 변수를 제공합니다. 일반적으로 **corosync.conf** 를 직접 편집하지 말고, 대신 pcs 또는 **pcs d** 인터페이스를 사용해야 합니다.*

***cib.xml** 파일은 클러스터의 구성과 클러스터의 모든 리소스의 현재 상태를 나타내는 XML 파일입니다. 이 파일은 Pacemaker의 CIB(Cluster Information Base)에서 사용됩니다. CIB의 내용은 전체 클러스터에서 자동으로 동기화됩니다. **cib.xml** 파일을 직접 편집하지 마십시오. 대신 **pcs** 또는 **pcsd** 인터페이스를 사용하십시오.*

1.4. RED HAT 고가용성 클러스터의 LVM 논리 볼륨

Red Hat High Availability Add-On은 두 개의 개별 클러스터 구성에서 LVM 볼륨에 대한 지원을 제공합니다.

선택할 수 있는 클러스터 구성은 다음과 같습니다.

- *클러스터의 단일 노드만 한 번에 스토리지에 액세스하는 활성/수동 장애 조치 구성의 HA-LVM(고가용성 LVM 볼륨) 입니다.*
- ***lvmlockd** 데몬을 사용하여 두 개 이상의 클러스터 노드가 동시에 스토리지에 액세스해야 하는 활성/활성 구성으로 스토리지 장치를 관리하는 LVM 볼륨입니다. **lvmlockd** 데몬은 복구 스토리지 애드온의 일부입니다.*

1.4.1. HA-LVM 또는 공유 볼륨 선택

***lvmlockd** 데몬에서 관리하는 HA-LVM 또는 공유 논리 볼륨은 배포 중인 애플리케이션 또는 서비스의 요구를 기반으로 해야 합니다.*

- *클러스터의 여러 노드에 활성/활성 시스템에서 LVM 볼륨에 대한 동시 읽기/쓰기 액세스 권한이 필요한 경우 **lvmlockd** 데몬을 사용하고 볼륨을 공유 볼륨으로 구성해야 합니다. **lvmlockd** 데몬은 클러스터의 노드 간에 LVM 볼륨의 활성화 및 변경을 동시에 조정하는 시스템을 제공합니다. 클러스터의 다양한 노드가 볼륨과 상호 작용하고 레이아웃을 변경할 수 있으므로 **lvm** 메타데이터의 잠금 서비스는 LVM 메타데이터를 보호합니다. 이러한 보호는 여러 클러스터 노드에서 동시에 활성화될 모든 볼륨 그룹을 공유 볼륨으로 구성하는 데 사용됩니다.*
- *고가용성 클러스터가 한 번에 지정된 LVM 볼륨에 액세스해야 하는 단일 멤버만 사용하여 활성/수동 방식으로 공유 리소스를 관리하도록 구성된 경우 **lvmlockd** 잠금 서비스 없이 HA-LVM을 사용할 수 있습니다.*

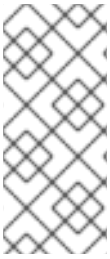
대부분의 애플리케이션은 다른 인스턴스와 동시에 실행하도록 설계되거나 최적화되지 않으므로 활성/수동 구성에서 더 잘 실행됩니다. 공유 논리 볼륨에서 클러스터 인식이 아닌 애플리케이션을 실행하도록 선

택하면 성능이 저하될 수 있습니다. 이는 이러한 인스턴스에 논리 볼륨 자체에 대한 클러스터 통신 오버헤드가 있기 때문입니다. 클러스터 인식 애플리케이션은 클러스터 파일 시스템에서 도입한 성능 손실과 클러스터 인식 논리 볼륨에서 성능상의 이점을 얻을 수 있어야 합니다. 이는 일부 애플리케이션 및 워크로드에서 다른 애플리케이션보다 더 쉽게 수행할 수 있습니다. 클러스터의 요구사항이 무엇인지, 활성/활성 클러스터의 최적화를 위한 추가 노력은 두 가지 LVM 변형 중에서 선택하는 방법입니다. 대부분의 사용자는 HA-LVM을 사용하여 최상의 HA 결과를 얻을 수 있습니다.

lvmlockd 를 사용하는 HA-LVM 및 공유 논리 볼륨은 LVM 메타데이터와 논리 볼륨의 손상을 방지한다는 점에서 비슷합니다. 이 경우 여러 머신이 중복되는 변경을 수행할 수 있는 경우 발생할 수 있습니다. HA-LVM은 논리 볼륨이 단독으로만 활성화할 수 있는 제한을 적용합니다. 즉, 한 번에 하나의 시스템에서만 활성화됩니다. 즉, 스토리지 드라이버의 로컬(비 클러스터) 구현만 사용됩니다. 이 방식으로 클러스터 조정 오버헤드가 발생하지 않도록 하면 성능이 향상됩니다. **lvmlockd** 를 사용하는 공유 볼륨은 이러한 제한이 적용되지 않으며 사용자는 클러스터의 모든 시스템에서 논리 볼륨을 활성화할 수 있습니다. 이로 인해 클러스터 인식 스토리지 드라이버를 사용해야 하므로 클러스터 인식 파일 시스템 및 애플리케이션을 맨 위에 배치할 수 있습니다.

1.4.2. 클러스터에서 LVM 볼륨 구성

클러스터는 Pacemaker를 통해 관리됩니다. HA-LVM 및 공유 논리 볼륨은 Pacemaker 클러스터와 함께 사용할 수 있으며 클러스터 리소스로 구성되어야 합니다.



참고

Pacemaker 클러스터에서 사용하는 LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우, Red Hat은 서비스가 시작되기 전에 서비스가 시작될 수 있도록 **systemd resource-agents-deps** 대상 및 **systemd** 드롭인 장치를 구성하는 것이 좋습니다. **systemd resource-agents-deps** 대상 구성에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스의 시작 순서 구성을 참조하십시오.

- HA-LVM 볼륨을 Pacemaker 클러스터의 일부로 구성하는 절차의 예는 Red Hat High Availability 클러스터에서 활성/수동 Apache HTTP 서버 구성 및 Red Hat High Availability 클러스터에서 활성/수동 NFS 서버 구성을 참조하십시오.
  이러한 절차에는 다음 단계가 포함됩니다.
  - 클러스터만 볼륨 그룹을 활성화할 수 있는지 확인
  - LVM 논리 볼륨 구성
  - LVM 볼륨을 클러스터 리소스로 구성
- **lvmlockd** 데몬을 사용하여 활성/활성 구성의 스토리지 장치를 관리하는 공유 LVM 볼륨을 구성하는 절차는 클러스터의 Cryostat2 파일 시스템 및 Red Hat High Availability 클러스터에서 활성/활성 Samba 서버 구성 을 참조하십시오.

2장. PACEMAKER 시작하기

Pacemaker 클러스터를 생성하는 데 사용하는 툴과 프로세스를 숙지하려면 다음 절차를 실행합니다. 작업 클러스터를 구성할 필요 없이 클러스터 소프트웨어가 어떻게 보이는지, 어떻게 관리되는지에 관심이 있는 사용자를 위한 것입니다.

참고

이 절차에서는 두 개 이상의 노드와 펜싱 장치 구성이 필요한 지원되는 Red Hat 클러스터를 생성하지 않습니다. RHEL High Availability 클러스터에 대한 Red Hat 지원 정책, 요구 사항 및 제한 사항에 대한 자세한 내용은 RHEL 고가용성 클러스터에 대한 지원 정책을 참조하십시오.

2.1. PACEMAKER 사용 학습

이 절차를 수행하면 Pacemaker를 사용하여 클러스터를 설정하는 방법, 클러스터 상태를 표시하는 방법 및 클러스터 서비스 구성 방법에 대해 알아봅니다. 이 예제에서는 Apache HTTP 서버를 클러스터 리소스로 생성하고 리소스가 실패할 때 클러스터가 응답하는 방법을 보여줍니다.

이 예제에서는 다음을 수행합니다.

- *노드는 **z1.example.com** 입니다.*
- *유동 IP 주소는 192.168.122.120 입니다.*

사전 요구 사항

- *RHEL 9를 실행하는 단일 노드*
- *노드의 정적으로 할당된 IP 주소 중 하나와 동일한 네트워크에 상주하는 유동 IP 주소*
- *실행 중인 노드의 이름이 **/etc/hosts** 파일에 있습니다.*

절차

1. *고가용성 채널에서 Red Hat High Availability Add-On 소프트웨어 패키지를 설치하고 **pcsd** 서비스를 시작 및 활성화합니다.*

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

   ***firewalld** 데몬을 실행하는 경우 Red Hat High Availability Add-On에 필요한 포트를 활성화합니다.*

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

2. *클러스터의 각 노드에 **hacluster** 사용자의 암호를 설정하고 **pcs** 명령을 실행할 노드의 클러스터의 각 노드에 대해 **hacluster** 사용자를 인증합니다. 이 예에서는 명령을 실행하는 노드인 단일 노드만 사용하지만 지원되는 Red Hat High Availability 다중 노드 클러스터를 구성하는 데 필요한 단계이기 때문에 이 단계는 여기에 포함됩니다.*

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

3. 하나의 멤버로 **my_cluster** 라는 클러스터를 생성하고 클러스터 상태를 확인합니다. 이 명령은 한 단계로 클러스터를 생성하고 시작합니다.

```
# 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
```

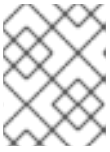
4. Red Hat High Availability 클러스터에는 클러스터의 펜싱을 구성해야 합니다. 이러한 요구 사항이 필요한 이유는 Red Hat High Availability Cluster의 Fencing에 설명되어 있습니다. 그러나 기본 Pacemaker 명령을 사용하는 방법만 표시하는 이 소개에서는 ston **ith 지원** 클러스터 옵션을 **false** 로 설정하여 펜싱을 비활성화합니다.

주의

stonith-enabled=false 를 사용하는 것은 프로덕션 클러스터에 완전히 부적절합니다. 실패한 노드가 안전하게 펜싱되도록 클러스터에 지시합니다.

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

5. 시스템에서 웹 브라우저를 구성하고 간단한 텍스트 메시지를 표시할 웹 페이지를 생성합니다. **firewalld** 데몬을 실행하는 경우 **httpd** 에 필요한 포트를 활성화합니다.



참고

systemctl enable 를 사용하여 클러스터가 관리할 서비스를 시스템 부팅 시 시작되도록 활성화하지 마십시오.

```
# 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 리소스 에이전트에서 Apache의 상태를 가져오려면 기존 구성에 다음을 추가하여 상태 서버 URL을 활성화합니다.

```
# 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. *클러스터에서 관리할* ***IPaddr2*** *및* ***apache*** *리소스를 생성합니다. 'IPaddr2' 리소스는 물리적 노드와 이미 연결되어 있지 않아야 하는 유동 IP 주소입니다. 'IPaddr2' 리소스의 NIC 장치가 지정되지 않은 경우 유동 IP는 노드에서 사용하는 정적으로 할당된 IP 주소와 동일한 네트워크에 있어야 합니다.*
   pcs resource list *명령을 사용하여 사용 가능한 모든 리소스 유형 목록을 표시할 수 있습니다.*
   pcs resource describe resourcetype *명령을 사용하여 지정된 리소스 유형에 설정할 수 있는 매개변수를 표시할 수 있습니다. 예를 들어 다음 명령은* ***apache*** *유형의 리소스에 대해 설정할 수 있는 매개변수를 표시합니다.*

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

이 예에서 IP 주소 리소스와 apache 리소스는 모두 ***apachegroup*** *이라는 그룹의 일부로 구성되어 있어 다중 노드 클러스터를 구성할 때 리소스가 동일한 노드에서 실행되도록 합니다.*

```
# 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
...
```

클러스터 리소스를 구성한 후 **pcs resource config** 명령을 사용하여 해당 리소스에 구성된 옵션을 표시할 수 있습니다.

```
# 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. 구성한 유동 IP 주소를 사용하여 생성한 웹 사이트를 브라우저에 지정합니다. 이렇게 하면 사용자가 정의한 텍스트 메시지가 표시됩니다.

8. apache 웹 서비스를 중지하고 클러스터 상태를 확인합니다. **killall -9** 를 사용하여 애플리케이션 수준 충돌을 시뮬레이션합니다.

```
# killall -9 httpd
```

클러스터 상태를 확인합니다. 웹 서비스를 중지하면 작업이 실패했지만 클러스터 소프트웨어가 서비스를 다시 시작했으며 웹 사이트에 계속 액세스할 수 있어야 합니다.

```
# 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
```

서비스가 가동되고 다시 실행되면 실패한 리소스에서 실패 상태를 지울 수 있으며 클러스터 상태를 볼 때 실패한 작업 알림이 더 이상 표시되지 않습니다.

```
# pcs resource cleanup WebSite
```

9. *클러스터 및 클러스터 상태를 확인하고 나면 노드에서 클러스터 서비스를 중지합니다. 이 소개를 위해 하나의 노드에서만 서비스를 시작했지만 실제 다중 노드 클러스터의 모든 노드에서 클러스터 서비스를 중지하므로* ***--all*** *매개변수가 포함됩니다.*

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

2.2. 장애 조치 구성 학습

다음 절차에서는 서비스를 실행하는 노드를 사용할 수 없게 되면 한 노드에서 다른 노드로 장애 조치되는 서비스를 실행하는 Pacemaker 클러스터를 생성하는 방법을 소개합니다. 이 절차를 통해 2노드 클러스터에서 서비스를 생성하는 방법을 배울 수 있으며 실행 중인 노드에서 실패할 때 해당 서비스에 어떤 일이 발생하는지 확인할 수 있습니다.

다음 예제 절차에서는 Apache HTTP 서버를 실행하는 2 노드 Pacemaker 클러스터를 구성합니다. 그런 다음 하나의 노드에서 Apache 서비스를 중지하여 서비스를 계속 사용할 수 있는 방법을 확인할 수 있습니다.

이 예제에서는 다음을 수행합니다.

- *노드는* ***z1.example.com*** *및* ***z2.example.com*** *입니다.*
- *유동 IP 주소는 192.168.122.120 입니다.*

사전 요구 사항

- *서로 통신할 수 있는 RHEL 9를 실행하는 두 개의 노드*
- *노드의 정적으로 할당된 IP 주소 중 하나와 동일한 네트워크에 상주하는 유동 IP 주소*
- *실행 중인 노드의 이름이* ***/etc/hosts*** *파일에 있습니다.*

절차

1. *두 노드 모두에서 고가용성 채널에서 Red Hat High Availability Add-On 소프트웨어 패키지를 설치하고* ***pcsd*** *서비스를 시작하고 활성화합니다.*

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

firewalld *데몬을 실행하는 경우 두 노드 모두에서 Red Hat High Availability Add-On에 필요한 포트를 활성화합니다.*

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

2. *클러스터의 두 노드에서* ***hacluster*** *사용자의 암호를 설정합니다.*

```
# passwd hacluster
```

3. ***pcs*** *명령을 실행할 노드의 클러스터의 각 노드에 대해* ***hacluster*** *사용자를 인증합니다.*

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

4. 두 노드가 모두 클러스터 구성원으로 **my_cluster** 라는 클러스터를 생성합니다. 이 명령은 한 단계로 클러스터를 생성하고 시작합니다. **pcs** 구성 명령이 전체 클러스터에 적용되므로 클러스터의 한 노드에서만 이 작업을 실행해야 합니다.
클러스터의 한 노드에서 다음 명령을 실행합니다.

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

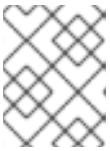
5. Red Hat High Availability 클러스터에는 클러스터의 펜싱을 구성해야 합니다. 이러한 요구 사항이 필요한 이유는 Red Hat High Availability Cluster의 Fencing에 설명되어 있습니다. 그러나 이 소개에서는 ston **ith 사용** 클러스터 옵션을 **false**로 설정하여 이 구성에서 장애 조치가 작동하는 방식만 표시하려면 펜싱을 비활성화합니다.

주의

stonith-enabled=false 를 사용하는 것은 프로덕션 클러스터에 완전히 부적절합니다. 실패한 노드가 안전하게 펜싱되도록 클러스터에 지시합니다.

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

6. 클러스터를 생성하고 펜싱을 비활성화한 후 클러스터 상태를 확인합니다.



참고

pcs cluster status 명령을 실행하면 시스템 구성 요소가 시작될 때 일시적으로 예제와 약간 다른 출력이 표시될 수 있습니다.

```
# 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. 두 노드 모두에서 웹 브라우저를 구성하고 간단한 텍스트 메시지를 표시할 웹 페이지를 생성합니다. **firewalld** 데몬을 실행하는 경우 **httpd** 에 필요한 포트를 활성화합니다.

참고

systemctl enable 를 사용하여 클러스터가 관리할 서비스를 시스템 부팅 시 시작되도록 활성화하지 마십시오.

```
# 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 리소스 에이전트가 Apache의 상태를 가져오려면 클러스터의 각 노드에서 상태 서버 URL을 활성화하기 위해 기존 구성 외에도 다음을 생성합니다.

```
# 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. *클러스터에서 관리할 **IPaddr2** 및 **apache** 리소스를 생성합니다. 'IPaddr2' 리소스는 물리적 노드와 이미 연결되어 있지 않아야 하는 유동 IP 주소입니다. 'IPaddr2' 리소스의 NIC 장치가 지정되지 않은 경우 유동 IP는 노드에서 사용하는 정적으로 할당된 IP 주소와 동일한 네트워크에 있어야 합니다.*
   ***pcs resource list** 명령을 사용하여 사용 가능한 모든 리소스 유형 목록을 표시할 수 있습니다.*
   ***pcs resource describe resourcetype** 명령을 사용하여 지정된 리소스 유형에 설정할 수 있는 매개변수를 표시할 수 있습니다. 예를 들어 다음 명령은 **apache** 유형의 리소스에 대해 설정할 수 있는 매개변수를 표시합니다.*

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

*이 예에서 IP 주소 리소스와 apache 리소스는 모두 **apachegroup** 이라는 그룹의 일부로 구성되어 있으므로 리소스가 동일한 노드에서 실행되도록 함께 유지됩니다.*

클러스터의 한 노드에서 다음 명령을 실행합니다.

```
# 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
...
```

이 인스턴스에서 **apachegroup** 서비스는 z1.example.com 노드에서 실행됩니다.

9. 생성한 웹 사이트에 액세스하여 실행 중인 노드에서 서비스를 중지한 후 두 번째 노드로 서비스가 실패하는 방식을 확인합니다.

   a. 구성한 유동 IP 주소를 사용하여 생성한 웹 사이트를 검색합니다. 이렇게 하면 사용자가 정의한 텍스트 메시지가 표시되고, 웹 사이트가 실행 중인 노드의 이름이 표시되어야 합니다.

   b. apache 웹 서비스를 중지합니다. **killall -9** 를 사용하여 애플리케이션 수준 충돌을 시뮬레이션합니다.

      ```
      # killall -9 httpd
      ```

      클러스터 상태를 확인합니다. 웹 서비스를 중지하면 작업이 실패했지만 클러스터 소프트웨어가 실행 중인 노드에서 서비스를 재시작하고 웹 브라우저에 계속 액세스할 수 있어야 합니다.

      ```
      # 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
      ```

서비스가 시작되고 다시 실행되면 실패 상태를 지웁니다.

```
# pcs resource cleanup WebSite
```

c. *서비스가 대기 모드로 실행되고 있는 노드를 대기 모드로 설정합니다. 펜싱을 사용하지 않도록 설정한 펜싱은 클러스터가 이러한 상황에서 복구하는 데 필요하므로 노드 수준 오류(예: 전원 케이블 가져오기)를 효과적으로 시뮬레이션할 수 없습니다.*

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

d. *클러스터 상태를 확인하고 서비스가 현재 실행 중인 위치를 확인합니다.*

```
# 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. *웹 사이트에 액세스합니다. 표시 메시지는 서비스가 현재 실행 중인 노드를 나타내지만 서비스 손실은 없어야 합니다.*

10. *클러스터 서비스를 첫 번째 노드로 복원하려면 노드를 대기 모드로 전환합니다. 이 경우 서비스를 해당 노드로 다시 이동할 필요는 없습니다.*

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

11. *최종 정리의 경우 두 노드 모두에서 클러스터 서비스를 중지합니다.*

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

3장. PCS 명령줄 인터페이스

pcs 명령줄 인터페이스에서 구성 파일에 더 쉬운 인터페이스를 제공하여 **corosync**,**pacemaker**,**booth** 및 **sbd** 와 같은 클러스터 서비스를 제어하고 구성합니다.

cib.xml 구성 파일을 직접 편집해서는 안 됩니다. 대부분의 경우 Pacemaker는 직접 수정된 **cib.xml** 파일을 거부합니다.

3.1. PCS HELP DISPLAY

pcs 의 **-h** 옵션을 사용하여 **pcs** 명령의 매개변수와 해당 매개변수에 대한 설명을 표시합니다.

다음 명령은 **pcs resource** 명령의 매개변수를 표시합니다.

```
# pcs resource -h
```

3.2. 원시 클러스터 구성 보기

클러스터 구성 파일을 직접 편집하지 않아야 하지만 **pcs cluster cib** 명령을 사용하여 원시 클러스터 구성을 볼 수 있습니다.

pcs cluster cib filename 명령을 사용하여 원시 클러스터 구성을 지정된 파일에 저장할 수 있습니다. 이전에 클러스터를 구성하고 활성 CIB가 있는 경우 다음 명령을 사용하여 원시 xml 파일을 저장합니다.

```
pcs cluster cib filename
```

예를 들어 다음 명령은 CIB의 원시 xml을 **testfile** 이라는 파일에 저장합니다.

```
# pcs cluster cib testfile
```

3.3. 작업 파일에 구성 변경 저장

클러스터를 구성할 때 활성 CIB에 영향을 주지 않고 지정된 파일에 구성 변경 사항을 저장할 수 있습니다. 이를 통해 현재 실행 중인 클러스터 구성을 각 개별 업데이트로 즉시 업데이트하지 않고 구성 업데이트를 지정할 수 있습니다.

CIB를 파일에 저장하는 방법에 대한 자세한 내용은 원시 클러스터 구성 보기를 참조하십시오. 해당 파일을 생성한 후에는 **pcs** 명령의 **-f** 옵션을 사용하여 활성 CIB가 아닌 해당 파일에 구성 변경 사항을 저장할 수 있습니다. 변경 사항을 완료하고 활성 CIB 파일을 업데이트할 준비가 되면 **pcs cluster cib-push** 명령을 사용하여 해당 파일 업데이트를 푸시할 수 있습니다.

절차

다음은 CIB 파일에 변경 사항을 푸시하는 데 권장되는 절차입니다. 이 절차에서는 저장된 원래의 CIB 파일의 사본을 생성하고 해당 사본을 변경합니다. 이러한 변경 사항을 활성 CIB로 푸시하는 경우 이 절차에서는 **pcs cluster cib-push** 명령의 **diff-against** 옵션을 지정하므로 원본 파일과 업데이트된 파일 간의 변경 사항만 CIB로 푸시됩니다. 이를 통해 사용자는 서로 덮어쓰지 않는 병렬로 변경할 수 있으며 전체 구성 파일을 구문 분석할 필요가 없는 Pacemaker의 로드를 줄일 수 있습니다.

1. 활성 CIB를 파일에 저장합니다. 이 예제에서는 CIB를 **original.xml**.xml이라는 파일에 저장합니다.

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

2. 저장된 파일을 구성 업데이트에 사용할 작업 파일에 복사합니다.

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

3. 필요에 따라 구성을 업데이트합니다. 다음 명령은 **updated.xml** 파일에 리소스를 생성하지만 현재 실행 중인 클러스터 구성에 해당 리소스를 추가하지 않습니다.

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

4. 업데이트된 파일을 활성 CIB로 내보내 원래 파일로 변경한 사항만 푸시하도록 지정합니다.

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

또는 다음 명령을 사용하여 CIB 파일의 현재 콘텐츠를 푸시할 수 있습니다.

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

전체 CIB 파일을 내보낼 때 Pacemaker는 버전을 확인하고 이미 클러스터에 있는 파일보다 오래된 CIB 파일을 푸시할 수 없습니다. 현재 클러스터에 있는 버전보다 오래된 버전으로 전체 CIB 파일을 업데이트해야 하는 경우 **pcs cluster cib-push** 명령의 **--config** 옵션을 사용할 수 있습니다.

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

3.4. 클러스터 상태 표시

클러스터 및 해당 구성 요소의 상태를 표시하는 데 사용할 수 있는 다양한 명령이 있습니다.

다음 명령을 사용하여 클러스터 및 클러스터 리소스의 상태를 표시할 수 있습니다.

```
# pcs status
```

리소스, 클러스터,**노드** 또는 **pcsd** 를 지정하여 **pcs status** 명령 의 command 매개변수를 사용하여 특정 **클러스터** 구성 요소의 상태를 표시할 수 있습니다.

```
pcs status commands
```

예를 들어 다음 명령은 클러스터 리소스의 상태를 표시합니다.

```
# pcs status resources
```

다음 명령은 클러스터 리소스가 아닌 클러스터 상태를 표시합니다.

```
# pcs cluster status
```

3.5. 전체 클러스터 구성 표시

다음 명령을 사용하여 전체 현재 클러스터 구성을 표시합니다.

```
# pcs config
```

3.6. PCS 명령을 사용하여 COROSYNC.CONF 파일 수정

pcs 명령을 사용하여 **corosync.conf** 파일에서 매개변수를 수정할 수 있습니다.

다음 명령은 **corosync.conf** 파일의 매개 변수를 수정합니다.

```
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]]
```

다음 예제 명령은 **knet_pmtud_interval** 전송 값과 **토큰** 을 입력하고 totem 값을 **조인** 합니다.

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

추가 리소스

- 기존 클러스터에서 노드를 추가 및 제거하는 방법에 대한 자세한 내용은 클러스터 노드 관리를 참조하십시오.
- 기존 클러스터에서 링크 추가 및 수정에 대한 자세한 내용은 기존 클러스터의 링크 추가 및 수정을 참조하십시오.
- ng 쿼럼 옵션 및 클러스터에서 쿼럼 장치 설정을 관리하는 방법에 대한 자세한 내용은 클러스터 쿼럼 및 쿼럼 장치 구성을 참조하십시오.

3.7. PCS 명령을 사용하여 COROSYNC.CONF 파일 표시

다음 명령은 **corosync.conf** 클러스터 구성 파일의 내용을 표시합니다.

```
# pcs cluster corosync
```

다음 예와 같이 **pcs cluster config** 명령을 사용하여 사용자가 읽을 수 있는 형식으로 **corosync.conf** 파일의 내용을 출력할 수 있습니다.

이 명령의 출력에는 클러스터가 RHEL 9.1 이상에서 생성된 경우 또는 UUID로 클러스터 ID에 설명된 대로 UUID가 수동으로 추가된 경우 클러스터의 UUID가 포함됩니다.

```
[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
```

다음 예제와 같이 기존 ***corosync.conf*** *파일을 다시 생성하는 데 사용할 수 있는* ***pcs*** *설정 명령을 표시하려면* ***--output-format=cmd*** *옵션과 함께* pcs ***cluster show*** *명령을 실행할 수 있습니다.*

```
[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
```

4 장. PACEMAKER 를 사용하여 RED HAT HIGH-AVAILABILITY 클러스터 생성

다음 절차에 **pcs** 명령줄 인터페이스를 사용하여 Red Hat High Availability 2-node 클러스터를 생성합니다.

이 예제에서 클러스터를 구성하려면 시스템에 다음 구성 요소가 포함되어야 합니다.

- 클러스터를 만드는 데 사용되는 노드 2개 이 예에서 사용된 노드는 **z1.example.com** 및 **z2.example.com** 입니다.
- 사설 네트워크의 네트워크 스위치. 클러스터 노드 간 통신과 네트워크 전원 스위치 및 파이버 채널 스위치와 같은 기타 클러스터 하드웨어 간의 통신을 위해 사설 네트워크가 필요하지는 않습니다.
- 클러스터의 각 노드에 대한 펜싱 장치입니다. 이 예에서는 호스트 이름이 **zapc.example.com** 인 APC 전원 스위치의 두 포트를 사용합니다.

4.1. 클러스터 소프트웨어 설치

다음 절차에 따라 클러스터 소프트웨어를 설치하고 클러스터 생성을 위해 시스템을 구성합니다.

절차

1. 클러스터의 각 노드에서 시스템 아키텍처에 해당하는 고가용성의 리포지토리를 활성화합니다. 예를 들어 x86_64 시스템의 고가용성 리포지토리를 활성화하려면 다음 **subscription-manager** 명령을 입력할 수 있습니다.

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

2. 클러스터의 각 노드에서 고가용성 채널에서 사용 가능한 모든 펜스 에이전트와 함께 Red Hat High Availability Add-On 소프트웨어 패키지를 설치합니다.

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

   또는 다음 명령과 함께 필요한 펜싱 에이전트와 함께 Red Hat High Availability Add-On 소프트웨어 패키지를 설치할 수 있습니다.

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

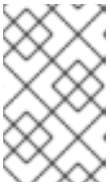
   다음 명령은 사용 가능한 펜스 에이전트 목록을 표시합니다.

   ```
   # 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
   ...
   ```

주의

Red Hat High Availability Add-On 패키지를 설치한 후에는 자동으로 설치되지 않도록 소프트웨어 업데이트 기본 설정이 설정되어 있는지 확인해야 합니다. 실행 중인 클러스터에 설치하면 예기치 않은 동작이 발생할 수 있습니다. 자세한 내용은 RHEL 고가용성 또는 복구 스토리지 클러스터에 소프트웨어 업데이트 적용 방법을 참조하십시오.

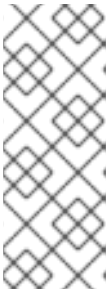
3. ***firewalld*** *데몬을 실행하는 경우 다음 명령을 실행하여 Red Hat High Availability Add-On에 필요한 포트를 활성화합니다.*



참고

rpm -q firewalld 명령을 사용하여 firewalld *데몬이 시스템에 설치되어 있는지 여부를 확인할 수 있습니다. 설치된 경우* ***firewall-cmd --state*** *명령을 사용하여 실행 중인지 확인할 수 있습니다.*

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



참고

클러스터 구성 요소에 대한 이상적인 방화벽 구성은 로컬 환경에 따라 달라집니다. 노드에 네트워크 인터페이스가 여러 개 있는지 또는 오프 호스트 방화벽이 있는지 여부와 같은 고려 사항이 있을 수 있습니다. Pacemaker 클러스터에 일반적으로 필요한 포트를 여는 예제는 로컬 조건에 맞게 수정해야 합니다. 고가용성 애드온 의 포트를 활성화하면 Red Hat High Availability Add-On 에서 사용할 수 있는 포트를 표시하고 각 포트에 대한 설명을 제공합니다.

4. ***pcs*** *를 사용하여 클러스터를 구성하고 노드 간에 통신하려면* ***pcs*** *관리 계정인 사용자 ID* ***hacluster*** *에 대해 각 노드에서 암호를 설정해야 합니다. 각 노드에서* ***hacluster*** *사용자의 암호를 동일하게 사용하는 것이 좋습니다.*

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

5. *클러스터를 구성하기 전에* ***pcsd*** *데몬을 시작하고 각 노드에서 부팅 시 시작되도록 활성화해야 합니다. 이 데몬은* ***pcs*** *명령과 함께 작동하여 클러스터의 노드 전체에서 구성을 관리합니다. 클러스터의 각 노드에서 다음 명령을 실행하여* ***pcsd*** *서비스를 시작하고 시스템을 시작할 때* ***pcsd*** *를 활성화합니다.*

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

4.2. PCP-ZEROCONF 패키지 설치(권장)

클러스터를 설정하면 Performance Co-dpdk(PCP) 툴에 대해 **pcp-zeroconf** 패키지를 설치하는 것이 좋습니다. PCP는 RHEL 시스템에 Red Hat이 권장하는 resource-monitoring 툴입니다. **pcp-zeroconf** 패키지를 설치하면 PCP가 클러스터를 방해하는 펜싱, 리소스 오류 및 기타 이벤트 조사의 이점을 위해 performance-monitoring 데이터를 실행하고 수집할 수 있습니다.

참고

PCP가 활성화된 클러스터 배포에서는 **/var/log/pcp/** 가 포함된 파일 시스템에서 PCP의 캡처된 데이터에 사용할 수 있는 충분한 공간이 필요합니다. PCP의 일반적인 공간 사용은 배포마다 다르지만 일반적으로 **pcp-zeroconf** 기본 설정을 사용할 때 10Gb이면 충분하며 일부 환경에는 덜 필요할 수 있습니다. 14일 간의 일반적인 활동에 걸쳐 이 디렉터리에서 사용량을 모니터링하면 보다 정확한 사용 예상을 제공할 수 있습니다.

절차

pcp-zeroconf 패키지를 설치하려면 다음 명령을 실행합니다.

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

이 패키지를 사용하면 **pmcd** 를 활성화하고 10초 간격으로 데이터 캡처를 설정합니다.

PCP 데이터 검토에 대한 자세한 내용은 RHEL High Availability 클러스터 노드를 재부팅한 이유 및 Red Hat 고객 포털에서 다시 발생하지 않도록 하는 방법을 참조하십시오.

4.3. 고가용성 클러스터 생성

다음 절차에 따라 Red Hat High Availability Add-On 클러스터를 생성합니다. 이 예제 절차에서는 **z1.example.com** 및 **z2.example.com** 노드로 구성된 클러스터를 생성합니다.

절차

1. **pcs** 를 실행할 노드의 클러스터의 각 노드에 대해 **pcs** 사용자 **hacluster** 를 인증합니다. 다음 명령은 **z1.example.com** 및 **z2.example.com** 으로 구성되는 2 노드 모두에 대해 **z1.example.com** 에서 **hacluster** 사용자를 인증합니다.

```
[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** 에서 다음 명령을 실행하여 z1.example.com 및 **z2.example.com** 노드로 구성된 2-노드 클러스터 **my_cluster** 를 생성합니다. 이렇게 하면 클러스터 구성 파일이 클러스터의 두 노드로 전파됩니다. 이 명령에는 클러스터의 두 노드에서 클러스터 서비스를 시작하는 **--start** 옵션이 포함되어 있습니다.

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

3. 노드가 부팅될 때 클러스터 서비스가 클러스터의 각 노드에서 실행되도록 합니다.

참고

특정 환경의 경우 이 단계를 건너뛰어 클러스터 서비스를 비활성화하도록 선택할 수 있습니다. 이를 통해 노드가 중단된 경우 노드가 클러스터에 다시 참여하기 전에 클러스터 또는 리소스 관련 문제를 해결할 수 있습니다. 클러스터 서비스가 비활성화된 경우 해당 노드에서 **pcs cluster start** 명령을 실행하여 노드를 재부팅할 때 서비스를 수동으로 시작해야 합니다.

```
[root@z1 ~]# pcs cluster enable --all
```

pcs cluster status 명령을 사용하여 클러스터의 현재 상태를 표시할 수 있습니다. **pcs cluster setup** 명령의 **--start** 옵션으로 클러스터 서비스를 시작하고 실행하기 전에 클러스터가 약간의 지연이 있을 수 있으므로 클러스터 및 해당 구성에서 후속 작업을 수행하기 전에 클러스터가 실행 중인지 확인해야 합니다.

```
[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. 여러 링크를 사용하여 고가용성 클러스터 생성

pcs cluster setup 명령을 사용하여 각 노드에 대한 모든 링크를 지정하여 여러 링크로 Red Hat High Availability 클러스터를 생성할 수 있습니다.

두 개의 링크를 사용하여 2-노드 클러스터를 생성하는 기본 명령의 형식은 다음과 같습니다.

```
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]
```

이 명령의 전체 구문은 **pcs**(8) 도움말 페이지를 참조하십시오.

여러 링크가 있는 클러스터를 생성할 때 다음 사항을 고려해야 합니다.

- **addr=주소** 매개변수의 순서가 중요합니다. 노드 이름 뒤에 지정된 첫 번째 주소는 **link0**, 두 번째 주소는 **link1** 의 경우 등입니다.
- 기본적으로 **link_priority가** 링크에 대해 지정되지 않은 경우 링크의 우선 순위는 링크 번호와 동일합니다. 링크 우선 순위는 지정된 순서에 따라 0, 1, 2, 3 등으로, 0이 링크 우선 순위입니다.
- 기본 링크 모드는 **passive** 이므로 번호가 가장 낮은 링크 우선 순위가 있는 활성 링크가 사용됩니다.
- **link_mode 및 link_ priority** 의 기본값을 사용하면 지정된 첫 번째 링크가 가장 높은 우선 순위 링크로 사용되며 해당 링크가 실패하면 다음 링크가 사용됩니다.
- 기본 전송 프로토콜인 **knet** 전송 프로토콜을 사용하여 최대 8개의 링크를 지정할 수 있습니다.

- *모든 노드에는 **addr=** 매개변수 수가 같아야 합니다.*
- *pcs cluster link add, pcs cluster link remove, pcs cluster link delete, **pcs cluster link delete, pcs cluster link update 명령을 사용하여 기존 클러스터에 링크를 추가, 제거**, 변경할 수 있습니다.*
- *단일 링크 클러스터와 마찬가지로 하나의 링크에 IPv4 및 IPv6 주소를 혼합하지 마십시오. 하나의 링크와 기타 IPv6를 실행하는 링크가 있을 수 있습니다.*
- *단일 링크 클러스터에서는 IPv4 및 IPv6 주소가 혼합되지 않는 IPv4 또는 IPv6 주소로 확인되는 경우 주소를 IP 주소 또는 이름으로 지정할 수 있습니다.*

*다음 예제에서는 두 개의 노드**rh80-node1** 및 **rh80-node2**. rh80-node2 . **rh80-node1** 이라는 두 개의 노드 클러스터를 만듭니다. rh80-node1에는 **link0** 으로 IP 주소 192.168.122.201이 link1로 . rh80-node2에는 두 개의 인터페이스가 있습니다. rh80-node2에는 두 개의 인터페이스가 있습니다. rh80-node1에는 link1이라는 두 개의 인터페이스가 있습니다. rh80-node1에는 **link1** 로 두 개의 인터페이스가 있습니다. **rh80-node2 IP 주소 192.168.122.202는 link0 으로, 192.168.123.202는 link1 로 .***

```
# 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
```

*링크 우선 순위를 기본값인 링크 번호와 다른 값으로 설정하려면 **pcs cluster setup** 명령의 **link _priority** 옵션을 사용하여 **link** 우선 순위를 설정할 수 있습니다. 다음 두 가지 예제의 각 명령은 두 개의 인터페이스를 사용하여 **2-**노드 클러스터를 생성합니다. 링크 **0**은 링크 우선 순위가 **1**이고 두 번째 링크인 **link 1**은 링크 우선 순위가 **0**입니다. 링크 **1**이 첫 번째로 사용되며 링크 **0**은 장애 조치**(failover)** 링크 역할을 합니다. 링크 모드를 지정하지 않기 때문에 기본값은 **passive**입니다.*

*이 두 명령은 동일합니다. **link** 키워드 다음에 링크 번호를 지정하지 않으면 **pcs** 인터페이스는 사용하지 않는 가장 낮은 링크 번호부터 자동으로 링크 번호를 추가합니다.*

```
# 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
```

*다음 예제와 같이 **pcs cluster setup** 명령의 **link_mode** 옵션을 사용하여 **link** 모드를 **passive** 의 기본값과 다른 값으로 설정할 수 있습니다.*

```
# 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
```

다음 예제에서는 링크 모드와 링크 우선 순위를 둘 다 설정합니다.

```
# 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
```

여러 링크가 있는 기존 클러스터에 노드를 추가하는 방법에 대한 자세한 내용은 여러 링크가 있는 클러스터에 노드 추가를 참조하십시오.

여러 링크를 사용하여 기존 클러스터의 링크를 변경하는 방법에 대한 자세한 내용은 기존 클러스터의 링크 추가 및 수정을 참조하십시오.

4.5. 펜싱 구성

클러스터의 각 노드에 대해 펜싱 장치를 구성해야 합니다. 펜스 구성 명령 및 옵션에 대한 자세한 내용은 **Red Hat High Availability** 클러스터에서 펜싱 구성을 참조하십시오.

Red Hat High Availability 클러스터에서 펜싱 및 중요성에 대한 일반적인 정보는 **Red Hat High Availability Cluster**의 **Fencing**을 참조하십시오.

참고

펜싱 장치를 구성할 때 해당 장치가 클러스터의 노드 또는 장치를 사용하여 전원을 공유하는지 여부에 주의하십시오. 노드와 해당 펜스 장치가 전원을 공유하는 경우 클러스터가 해당 노드의 전원을 펜싱할 수 없고 해당 펜스 장치를 손실해야 할 위험이 있을 수 있습니다. 이러한 클러스터에는 펜스 장치 및 노드에 대한 중복 전원 공급 장치 또는 전원을 공유하지 않는 중복 펜싱 장치가 있어야 합니다. **SBD** 또는 스토리지 펜싱과 같은 다른 펜싱 방법도 격리된 정전 시 중복성을 가져올 수 있습니다.

절차

이 예에서는 호스트 이름이 **zapc.example.com** 인 **APC** 전원 스위치를 사용하여 노드를 펜싱하고, **fence_apc_snmp** 펜싱 에이전트를 사용합니다. 두 노드가 동일한 펜싱 에이전트로 펜싱되므로 **pcs mk_host_map** 옵션을 사용하여 두 펜싱 장치를 단일 리소스로 구성할 수 있습니다.

pcs stonith create 명령을 사용하여 장치를 **stonith** 리소스로 구성하여 펜싱 장치를 생성합니다. 다음 명령은 **z1.example.com** 및 **z2.example.com** 노드에 **fence_apc_snmp** 펜싱 에이전트를 사용하는 **myapc** 라는 **stonith** 리소스를 구성합니다. **pcs mk_host_map** 옵션은 **z1.example.com** 을 포트 **1**에 매핑하고 **z2.example.com** 은 포트 **2**에 매핑합니다. **APC** 장치의 로그인 값과 암호는 모두 **apc** 입니다. 기본적으로 이 장치는 각 노드에 대해 **60**초의 모니터 간격을 사용합니다.

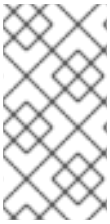
노드의 호스트 이름을 지정할 때 **IP** 주소를 사용할 수 있습니다.

```
[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"
```

다음 명령은 기존 펜싱 장치의 매개 변수를 표시합니다.

```
[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)
```

펜스 장치를 구성한 후 장치를 테스트해야 합니다. 펜스 장치를 테스트하는 방법에 대한 자세한 내용은 *차단 장치 테스트를* 참조하십시오.



참고

펜싱을 제대로 테스트하지 않으므로 네트워크 인터페이스를 비활성화하여 펜스 장치를 테스트하지 마십시오.

참고

펜싱이 설정되고 클러스터를 시작하면 다시 시작하면 시간 초과가 초과되지 않은 경우에도 네트워크를 다시 시작하는 노드의 펜싱이 트리거됩니다. 따라서 노드에서 의도하지 않은 펜싱을 트리거하므로 클러스터 서비스가 실행되는 동안 **network** 서비스를 재시작하지 마십시오.

4.6. 클러스터 구성 백업 및 복원

다음 명령은 **tar** 아카이브의 클러스터 구성을 백업하고 백업의 모든 노드에 클러스터 구성 파일을 복원합니다.

절차

tar 아카이브에서 클러스터 구성을 백업하려면 다음 명령을 사용합니다. 파일 이름을 지정하지 않으면 표준 출력이 사용됩니다.

```
pcs config backup filename
```

참고

pcs config backup 명령은 **CIB**에 구성된 대로 클러스터 구성 자체만 백업합니다. 리소스 데몬의 구성은 이 명령의 범위를 벗어납니다. 예를 들어, 클러스터에 **Apache** 리소스를 구성한 경우 리소스 설정(**CIB**에 있음)이 백업되는 반면 **Apache** 데몬 설정('/etc/httpd') 및 서비스하는 파일이 백업되지 않습니다. 마찬가지로 클러스터에 구성된 데이터베이스 리소스가 있는 경우 데이터베이스 자체는 백업되지 않고 **CIB**(데이터베이스 리소스 구성)가 수행됩니다.

다음 명령을 사용하여 백업의 모든 클러스터 노드에서 클러스터 구성 파일을 복원합니다. **--local** 옵션을 지정하면 이 명령을 실행하는 노드에서만 클러스터 구성 파일이 복원됩니다. 파일 이름을 지정하지 않으면 표준 입력이 사용됩니다.

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

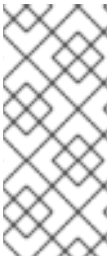
4.7. 고가용성 애드온의 포트 활성화

클러스터 구성 요소에 대한 이상적인 방화벽 구성은 로컬 환경에 따라 달라집니다. 노드에 네트워크 인터페이스가 여러 개 있는지 또는 오프 호스트 방화벽이 있는지 여부와 같은 고려 사항이 있을 수 있습니다.

firewalld 데몬을 실행하는 경우 다음 명령을 실행하여 **Red Hat High Availability Add-On**에 필요한 포트를 활성화합니다.

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

로컬 조건에 맞게 열려 있는 포트를 수정해야 할 수 있습니다.



참고

rpm -q firewalld 명령을 사용하여 **firewalld** 데몬이 시스템에 설치되어 있는지 여부를 확인할 수 있습니다. **firewalld** 데몬이 설치된 경우 **firewall-cmd --state** 명령을 사용하여 실행 중인지 여부를 확인할 수 있습니다.

다음 표에서는 **Red Hat High Availability Add-On**에서 사용할 수 있는 포트를 보여주고 해당 포트가 어떤 용도로 사용되는지에 대한 설명을 제공합니다.

표 **4.1.** 고가용성 애드온 사용을 위한 포트

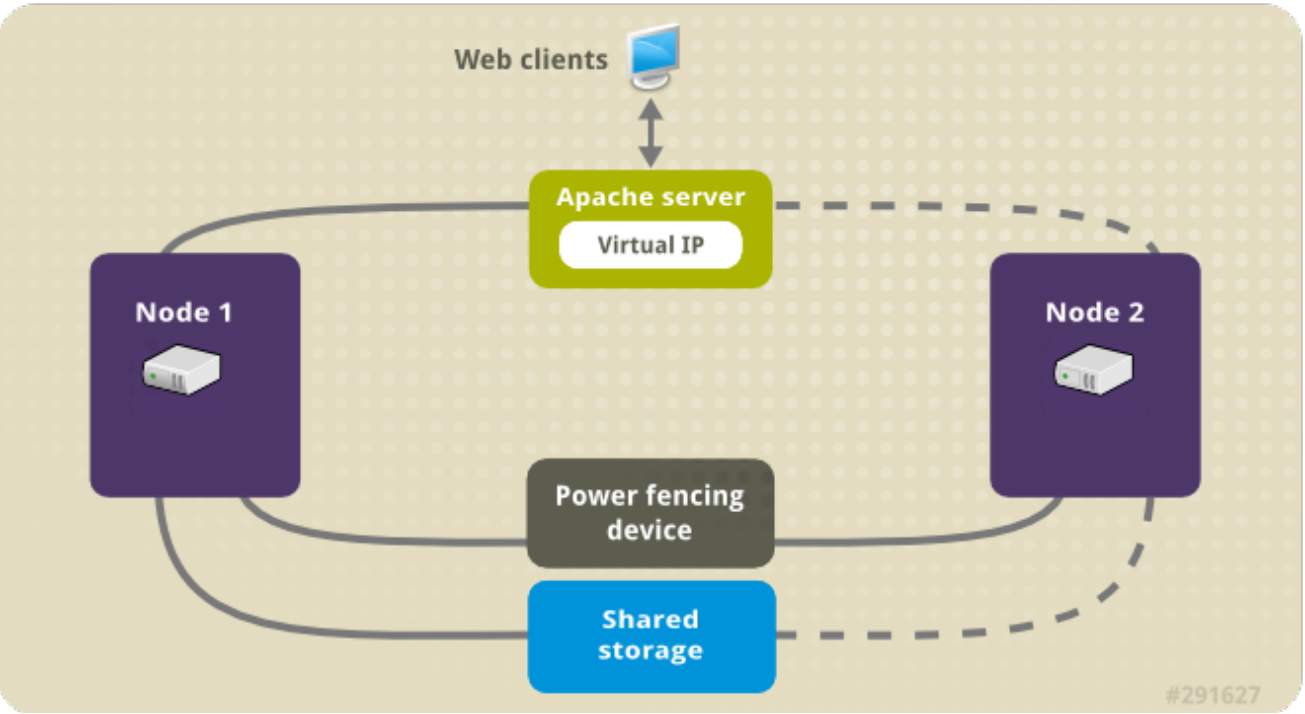
포트	필요한 경우
TCP 2224	모든 노드에 필요한 기본 **pcsd** 포트(pcsd Web UI에 필요하며 노드 간 통신에 필요합니다). **/etc/sysconfig/ pcsd 파일에서 PCSD_PORT 매개변수를 사용하여 pcsd** 포트를 구성할 수 있습니다. 2224 포트를 여는 것이 매우 중요합니다. 이러한 방식으로 모든 노드의 **pcs** 가 자체를 포함하여 클러스터의 모든 노드와 통신할 수 있습니다. Booth 클러스터 티켓 관리자 또는 쿼럼 장치를 사용하는 경우 Booth 중재자 또는 쿼럼 장치 호스트와 같은 모든 관련 호스트에서 포트 2224를 열어야 합니다.
TCP 3121	클러스터에 Pacemaker 원격 노드가 있는 경우 모든 노드에 필요합니다. 전체 클러스터 노드의 Pacemaker **기반** 데몬이 포트 3121에서 Pacemaker 원격 노드의 **pacemaker_remoted** 데몬에 연결됩니다. 클러스터 통신에 별도의 인터페이스를 사용하는 경우 해당 인터페이스에서만 포트를 열어야 합니다. 최소 포트는 Pacemaker 원격 노드에서 를 열어 전체 클러스터 노드로 열어야 합니다. 사용자는 전체 노드와 원격 노드 간에 호스트를 변환하거나 호스트의 네트워크를 사용하여 컨테이너 내에서 원격 노드를 실행할 수 있으므로 모든 노드에 포트를 여는 것이 유용할 수 있습니다. 노드가 아닌 다른 호스트에 포트를 열 필요는 없습니다.
TCP 5403	**corosync-qnetd** 에서 쿼럼 장치를 사용할 때 쿼럼 장치 호스트에 필요합니다. **corosync-qnetd** 명령의 **-p** 옵션을 사용하여 기본값을 변경할 수 있습니다.
UDP 5404-5412	노드 간 통신을 용이하게 하기 위해 corosync 노드에서 필요합니다. 포트 5404-5412를 여는 것이 중요합니다. 이러한 방식으로 해당 포트를 포함하여 클러스터의 모든 노드와 **corosync** 가 통신할 수 있습니다.
TCP 21064	클러스터에 DLM(예: **GFS2)이 필요한 리소스가 포함된 경우 모든 노드에 필요합니다**.
TCP 9929, UDP 9929	Booth 티켓 관리자가 다중 사이트 클러스터를 설정하는 데 사용될 때 동일한 노드의 연결에 모든 클러스터 노드와 Booth 중재자 노드에서 열려 있어야 합니다.

5장. RED HAT HIGH AVAILABILITY 클러스터에서 활성/수동 APACHE HTTP 서버 구성

다음 절차에 따라 2노드 Red Hat Enterprise Linux High Availability Add-On 클러스터에서 활성/수동 Apache HTTP 서버를 구성합니다. 이 사용 사례에서는 클라이언트가 유동 IP 주소를 통해 Apache HTTP 서버에 액세스합니다. 웹 서버는 클러스터의 두 노드 중 하나에서 실행됩니다. 웹 서버가 실행 중인 노드가 작동하지 않으면 서비스 중단이 최소화된 클러스터의 두 번째 노드에서 웹 서버가 다시 시작됩니다.

다음 그림에서는 클러스터가 네트워크 전원 스위치와 공유 스토리지로 구성된 2노드 Red Hat High Availability 클러스터인 클러스터에 대한 간략한 개요를 보여줍니다. 클러스터 노드는 가상 IP를 통해 Apache HTTP 서버에 대한 클라이언트 액세스를 위해 공용 네트워크에 연결됩니다. Apache 서버는 각각 Apache 데이터가 보관되는 스토리지에 액세스할 수 있는 노드 1 또는 노드 2에서 실행됩니다. 이 그림에서 웹 서버는 노드 1이 작동하는 경우 노드 2를 실행하는 데 사용할 수 있는 반면, 노드 2는 노드 1에서 실행됩니다.

그림 5.1. Red Hat High Availability Two-Node 클러스터의 Apache



이 사용 사례에서는 시스템에 다음 구성 요소가 포함되어야 합니다.

- 각 노드에 대해 전원 펜싱이 구성된 2노드 Red Hat High Availability 클러스터입니다. 개인 네트워크가 필요하지는 않지만 권장되지 않습니다. 이 절차에서는 Pacemaker를 사용하여 Red Hat High-Availability 클러스터를 생성하는 데 제공된 클러스터 예제를 사용합니다.
- Apache에 필요한 공용 가상 IP 주소입니다.
-

iSCSI, 파이버 채널 또는 기타 공유 네트워크 블록 장치를 사용하여 클러스터의 노드에 대한 공유 스토리지입니다.

클러스터는 웹 서버에 필요한 클러스터 구성 요소(예: LVM 리소스, 파일 시스템 리소스, IP 주소 리소스, 웹 서버 리소스)를 포함하는 Apache 리소스 그룹으로 구성됩니다. 이 리소스 그룹은 클러스터의 한 노드에서 다른 노드로 장애 조치할 수 있으므로 두 노드 중 하나가 웹 서버를 실행할 수 있습니다. 이 클러스터에 대한 리소스 그룹을 생성하기 전에 다음 절차를 수행합니다.

1. *논리 볼륨 my_lv 에 XFS 파일 시스템을 구성합니다.*
2. *웹 서버를 구성합니다.*

이러한 단계를 수행한 후 리소스 그룹과 포함된 리소스를 생성합니다.

5.1. PACEMAKER 클러스터에서 XFS 파일 시스템을 사용하여 LVM 볼륨 구성

다음 절차에 따라 클러스터 노드 간에 공유되는 LVM 논리 볼륨을 스토리지에 생성합니다.

참고

클러스터 노드에서 사용하는 LVM 볼륨 및 해당 파티션 및 장치를 클러스터 노드에만 연결해야 합니다.

다음 절차에서는 LVM 논리 볼륨을 생성한 다음 Pacemaker 클러스터에서 사용할 해당 볼륨에 XFS 파일 시스템을 생성합니다. 이 예에서 공유 파티션 /dev/sdb1 은 LVM 논리 볼륨이 생성될 LVM 물리 볼륨을 저장하는 데 사용됩니다.

절차

1. *클러스터의 두 노드에서 다음 단계를 수행하여 LVM 시스템 ID의 값을 시스템의 uname 식별자 값으로 설정합니다. LVM 시스템 ID는 클러스터만 볼륨 그룹을 활성화할 수 있도록 하는 데 사용됩니다.*
    a. */etc/lvm/lvm.conf 구성 파일의 system_id_source 구성 옵션을 uname 으로 설정합니다.*

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

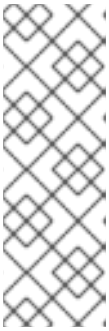
b.

노드의 LVM 시스템 ID가 노드의 uname 과 일치하는지 확인합니다.

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

2.

LVM 볼륨을 만들고 해당 볼륨에 XFS 파일 시스템을 만듭니다. /dev/sdb1 파티션은 공유되는 스토리지이므로 하나의 노드에서만 절차의 일부를 수행합니다.



참고

LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker를 시작하기 전에 서비스를 시작하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성을 참조하십시오.

a.

/dev/sdb1 파티션에 LVM 물리 볼륨을 만듭니다.

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

참고

LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker를 시작하기 전에 서비스를 시작하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성을 참조하십시오.

b.

물리 볼륨 /dev/sdb1 로 구성된 볼륨 그룹 my_vg 를 생성합니다.

시작할 때 Pacemaker에서 관리하는 볼륨 그룹이 자동으로 활성화되지 않도록 --setautoactivation n 플래그를 지정합니다. 생성 중인 LVM 볼륨에 기존 볼륨 그룹을 사용하는 경우 볼륨 그룹에 vgchange --setautoactivation n 명령을 사용하여 이 플래그를 재설정할 수 있습니다.

```
[root@z1 ~]# vgcreate --setautoactivation n my_vg /dev/sdb1
  Volume group "my_vg" successfully created
```

c.

새 볼륨 그룹에 실행 중인 노드 및 볼륨 그룹을 생성한 노드의 시스템 ID가 있는지 확인합니다.

```
[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.

볼륨 그룹 my_vg 를 사용하여 논리 볼륨을 생성합니다.

```
[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 명령을 사용하여 논리 볼륨을 표시할 수 있습니다.

```
[root@z1 ~]# lvs
  LV      VG      Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
  my_lv   my_vg   -wi-a---- 452.00m
  ...
```

e.

논리 볼륨 my_lv 에 XFS 파일 시스템을 생성합니다.

```
[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 파일에서 use_devicesfile = 1 매개변수를 사용하여 장치 파일을 사용하는 경우 클러스터의 두 번째 노드의 장치 파일에 공유 장치를 추가합니다. 이 기능은 기본적으로 활성화되어 있습니다.

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

5.2. APACHE HTTP SERVER 구성

다음 절차에 따라 Apache HTTP 서버를 구성합니다.

절차

1. *Apache HTTP Server가 클러스터의 각 노드에 설치되어 있는지 확인합니다. Apache HTTP 서버의 상태를 확인하려면 클러스터에 wget 툴을 설치해야 합니다.*

   각 노드에서 다음 명령을 실행합니다.

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

   firewalld 데몬을 실행하는 경우 클러스터의 각 노드에서 Red Hat High Availability Add-On에 필요한 포트를 활성화하고 httpd 를 실행하는 데 필요한 포트를 활성화합니다. 이 예제에서는 공용 액세스에 대해 httpd 포트를 활성화하지만 httpd 에 대해 활성화할 특정 포트는 프로덕션 사용량에 따라 다를 수 있습니다.

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

2. *Apache 리소스 에이전트가 Apache의 상태를 가져오려면 클러스터의 각 노드에서 상태 서버 URL을 활성화하기 위해 기존 구성 외에도 다음을 생성합니다.*

   ```
   # cat <<-END > /etc/httpd/conf.d/status.conf
   <Location /server-status>
      SetHandler server-status
      Require local
   </Location>
   END
   ```

3. *제공할 Apache의 웹 페이지를 만듭니다.*

   클러스터의 한 노드에서 XFS 파일 시스템으로 LVM 볼륨 구성에서 생성한 논리 볼륨이 활성화되었는지 확인하고, 해당 논리 볼륨에서 생성한 파일 시스템을 마운트하고, 해당 파일 시스템에 index.html 파일을 생성한 다음 파일 시스템을 마운트 해제합니다.

   ```
   # 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. 리소스 및 리소스 그룹 생성

다음 절차에 따라 클러스터의 리소스를 생성합니다. 이러한 리소스가 모두 동일한 노드에서 실행되도록 하려면 **apachegroup** 리소스 그룹의 일부로 구성됩니다. 생성할 리소스는 다음과 같이 나열되며, 시작하는 순서대로 나열됩니다.

1. **XFS** 파일 시스템으로 **LVM** 볼륨 구성에서 생성한 **LVM** 볼륨 그룹을 사용하는 **my_lvm** 이라는 **LVM**을 활성화합니다.

2. **XFS** 파일 시스템을 사용하여 **LVM** 볼륨 구성에서 생성한 파일 시스템 장치 **/dev/my_vg/my_lv** 를 사용하는 **my_fs** 라는 파일 시스템 리소스입니다.

3. **apachegroup** 리소스 그룹의 유동 **IP** 주소인 **IPaddr2** 리소스입니다. **IP** 주소는 물리적 노드와 이미 연결되어 있지 않아야 합니다. **IPaddr2** 리소스의 **NIC** 장치가 지정되지 않은 경우 유동 **IP**는 노드의 정적으로 할당된 **IP** 주소 중 하나와 동일한 네트워크에 있어야 합니다. 그렇지 않으면 유동 **IP** 주소를 할당하기 위해 **NIC** 장치를 올바르게 탐지할 수 없습니다.

4. **Apache HTTP** 서버 구성에 정의된 **index.html** 파일 및 **Apache** 설정을 사용하는 **website** 라는 **apache** 리소스입니다.

다음 절차에서는 **apachegroup** 리소스 그룹과 그룹에 포함된 리소스를 생성합니다. 리소스는 그룹에 추가하는 순서대로 시작하고 그룹에 추가된 반대 순서로 중지됩니다. 클러스터의 노드 중 하나에서만 이 절차를 실행합니다.

절차

1. 다음 명령은 **LVM** 활성화 리소스 **my_lvm** 을 생성합니다. **apachegroup** 리소스 그룹이 아직 존재하지 않기 때문에 이 명령은 리소스 그룹을 생성합니다.

참고

활성/수동 **HA** 구성에서 동일한 **LVM** 볼륨 그룹을 사용하는 **LVM** 활성화 리소스를 두 개 이상 구성하지 마십시오. 이로 인해 데이터 손상이 발생할 수 있습니다. 또한 활성/수동 **HA** 구성에서 **LVM** 활성화 리소스를 복제 리소스로 구성하지 마십시오.

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

리소스를 생성하면 리소스가 자동으로 시작됩니다. 다음 명령을 사용하여 리소스가 생성되었으며 시작되었는지 확인할 수 있습니다.

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

pcs resource disable 및 **pcs resource enable** 명령을 사용하여 개별 리소스를 수동으로 중지하고 시작할 수 있습니다.

2. 다음 명령은 구성에 대한 나머지 리소스를 생성하여 기존 리소스 그룹 **apachegroup** 에 추가합니다.

```
[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. 리소스 및 리소스를 포함하는 리소스 그룹을 생성한 후 클러스터 상태를 확인할 수 있습니다. **4**개의 리소스가 모두 동일한 노드에서 실행되고 있습니다.

```
[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
```

클러스터의 펜싱 장치를 구성하지 않은 경우 기본적으로 리소스가 시작되지 않습니다.

4. 클러스터가 가동되어 실행되면 브라우저에서 **IPaddr2** 리소스로 정의한 **IP** 주소를 가리키어 샘플 디스플레이를 볼 수 있으며 간단한 단어 **"Hello"**로 구성됩니다.

```
Hello
```

구성한 리소스가 실행 중이 아닌지 확인하면 **pcs resource debug-start resource** 명령을 실행하여 리소스 구성을 테스트할 수 있습니다.

5. **apache** 리소스 에이전트를 사용하여 **Apache**를 관리하는 경우 **systemd** 를 사용하지 않습니다. 이로 인해 **systemctl** 을 사용하여 **Apache**를 다시 로드하지 않도록 **Apache**와 함께 제공된 **logrotate** 스크립트를 편집해야 합니다.

클러스터의 각 노드의 **/etc/logrotate.d/httpd** 파일에서 다음 행을 제거합니다.

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

/var/run/httpd-website.pid를 **website** 가 **Apache** 리소스의 이름인 **PID** 파일 경로로 지정하여 제거한 행을 다음 세 줄로 바꿉니다. 이 예에서 **Apache** 리소스 이름은 **website** 입니다.

```
/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. 리소스 구성 테스트

다음 절차에 따라 클러스터에서 리소스 구성을 테스트합니다.

리소스 및 리소스 그룹 생성에 표시된 클러스터 상태 표시에서 모든 리소스 가 노드 **z1.example.com** 에서 실행됩니다. 다음 절차에 사용하여 첫 번째 노드를 **standby** 모드가 되게 하여 리소스 그룹이 **z2.example.com** 노드로 실패했는지 여부를 테스트할 수 있습니다. 그러면 노드가 더 이상 리소스를 호스팅할 수 없습니다.

절차

1. *다음 명령을 실행하면 **z1.example.com** 노드를 **standby** 모드가 됩니다.*

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

2. *노드 **z1** 을 **standby** 모드로 전환한 후 클러스터 상태를 확인합니다. 이제 리소스는 모두 **z2** 에서 실행되어야 합니다.*

```
[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
```

*정의된 **IP** 주소의 웹 사이트는 중단 없이 계속 표시되어야 합니다.*

3. ***z1** 을 대기 모드에서 제거하려면 다음 명령을 입력합니다.*

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

참고

*대기 모드에서 노드를 제거해도 리소스가 해당 노드로 되돌아가지 않습니다. 이는 리소스의 **resource-stickiness** 값에 따라 달라집니다. **resource-stickiness** **meta** 속성에 대한 자세한 내용은 현재 노드를 선호하도록 리소스 구성을 참조하십시오.*

6장. RED HAT HIGH AVAILABILITY 클러스터에서 활성/수동 NFS 서버 구성

Red Hat High Availability Add-On은 공유 스토리지를 사용하여 Red Hat Enterprise Linux High Availability Add-On 클러스터에서 고가용성 NFS 서버를 실행할 수 있도록 지원합니다. 다음 예제에서는 클라이언트가 유동 IP 주소를 통해 NFS 파일 시스템에 액세스하는 2-노드 클러스터를 구성하고 있습니다. NFS 서버는 클러스터의 두 노드 중 하나에서 실행됩니다. NFS 서버가 실행되는 노드가 작동하지 않는 경우 서비스 중단을 최소화하여 클러스터의 두 번째 노드에서 NFS 서버가 다시 시작됩니다.

이 사용 사례에서는 시스템에 다음 구성 요소가 포함되어야 합니다.

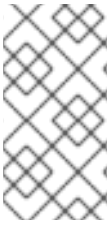
- 각 노드에 대해 전원 펜싱이 구성된 2노드 Red Hat High Availability 클러스터입니다. 개인 네트워크가 필요하지는 않지만 권장되지 않습니다. 이 절차에서는 Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성에 제공되는 클러스터 예제를 사용합니다.
- NFS 서버에 필요한 공용 가상 IP 주소입니다.
- iSCSI, 파이버 채널 또는 기타 공유 네트워크 블록 장치를 사용하여 클러스터의 노드에 대한 공유 스토리지입니다.

기존 2 노드 Red Hat Enterprise Linux High Availability 클러스터에 고가용성 활성/수동 NFS 서버를 구성하려면 다음 단계를 수행해야 합니다.

1. 클러스터의 노드의 공유 스토리지에 LVM 논리 볼륨에 파일 시스템을 구성합니다.
2. LVM 논리 볼륨의 공유 스토리지에 NFS 공유를 구성합니다.
3. 클러스터 리소스를 생성합니다.
4. 구성한 NFS 서버를 테스트합니다.

6.1. PACEMAKER 클러스터에서 XFS 파일 시스템을 사용하여 LVM 볼륨 구성

다음 절차에 따라 클러스터 노드 간에 공유되는 LVM 논리 볼륨을 스토리지에 생성합니다.



참고

클러스터 노드에서 사용하는 LVM 볼륨 및 해당 파티션 및 장치를 클러스터 노드에만 연결해야 합니다.

다음 절차에서는 LVM 논리 볼륨을 생성한 다음 Pacemaker 클러스터에서 사용할 해당 볼륨에 XFS 파일 시스템을 생성합니다. 이 예에서 공유 파티션 /dev/sdb1 은 LVM 논리 볼륨이 생성될 LVM 물리 볼륨을 저장하는 데 사용됩니다.

절차

1. 클러스터의 두 노드에서 다음 단계를 수행하여 LVM 시스템 ID의 값을 시스템의 uname 식별자 값으로 설정합니다. LVM 시스템 ID는 클러스터만 볼륨 그룹을 활성화할 수 있도록 하는 데 사용됩니다.

   a. /etc/lvm/lvm.conf 구성 파일의 system_id_source 구성 옵션을 uname 으로 설정합니다.

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

   b. 노드의 LVM 시스템 ID가 노드의 uname 과 일치하는지 확인합니다.

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

2. LVM 볼륨을 만들고 해당 볼륨에 XFS 파일 시스템을 만듭니다. /dev/sdb1 파티션은 공유되는 스토리지이므로 하나의 노드에서만 절차의 일부를 수행합니다.

참고

LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker를 시작하기 전에 서비스를 시작하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성을 참조하십시오.

a.

/dev/sdb1 파티션에 LVM 물리 볼륨을 만듭니다.

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

참고

LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker를 시작하기 전에 서비스를 시작하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성을 참조하십시오.

b.

물리 볼륨 /dev/sdb1 로 구성된 볼륨 그룹 my_vg 를 생성합니다.

시작할 때 Pacemaker에서 관리하는 볼륨 그룹이 자동으로 활성화되지 않도록 --setautoactivation n 플래그를 지정합니다. 생성 중인 LVM 볼륨에 기존 볼륨 그룹을 사용하는 경우 볼륨 그룹에 vgchange --setautoactivation n 명령을 사용하여 이 플래그를 재설정할 수 있습니다.

```
[root@z1 ~]# vgcreate --setautoactivation n my_vg /dev/sdb1
  Volume group "my_vg" successfully created
```

c.

새 볼륨 그룹에 실행 중인 노드 및 볼륨 그룹을 생성한 노드의 시스템 ID가 있는지 확인합니다.

```
[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.

볼륨 그룹 my_vg 를 사용하여 논리 볼륨을 생성합니다.

```
[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 명령을 사용하여 논리 볼륨을 표시할 수 있습니다.

```
[root@z1 ~]# lvs
  LV      VG      Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
```

```
  my_lv   my_vg   -wi-a---- 452.00m
  ...
```

e.

논리 볼륨 **my_lv** 에 **XFS** 파일 시스템을 생성합니다.

```
[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 파일에서 **use_devicesfile = 1** 매개변수를 사용하여 장치 파일을 사용하는 경우 클러스터의 두 번째 노드의 장치 파일에 공유 장치를 추가합니다. 이 기능은 기본적으로 활성화되어 있습니다.

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

6.2. NFS 공유 구성

다음 절차에 따라 **NFS** 서비스 장애 조치의 **NFS** 공유를 구성합니다.

절차

1.

클러스터의 두 노드에서 **/ tekton** 디렉터리를 만듭니다.

```
# mkdir /nfsshare
```

2.

클러스터의 한 노드에서 다음 절차를 수행합니다.

a.

XFS 파일 시스템을 사용하여 **LVM** 볼륨 구성에서 생성한 논리 볼륨이 활성화되었는지 확인한 다음 **/nfsshare** 디렉터리의 논리 볼륨에 생성한 파일 시스템을 마운트합니다.

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

b.

/ tekton 디렉터리에 내보내기 디렉터리 트리를 생성합니다.

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

c.

NFS 클라이언트가 액세스할 내보내기 디렉터리에 파일을 배치합니다. 이 예제에서는 clientdatafile1 및 clientdatafile2 라는 테스트 파일을 만들고 있습니다.

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

d.

파일 시스템을 마운트 해제하고 LVM 볼륨 그룹을 비활성화합니다.

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

6.3. 클러스터에서 NFS 서버의 리소스 및 리소스 그룹 구성

다음 절차에 따라 클러스터에서 NFS 서버에 대한 클러스터 리소스를 구성합니다.

참고

클러스터의 펜싱 장치를 구성하지 않은 경우 기본적으로 리소스가 시작되지 않습니다.

구성한 리소스가 실행 중이 아닌지 확인하면 pcs resource debug-start resource 명령을 실행하여 리소스 구성을 테스트할 수 있습니다. 그러면 클러스터의 제어 및 지식 외부에서 서비스가 시작됩니다. 구성된 리소스가 다시 실행 중인 시점에서 pcs resourcecleanup 리소스 를 실행하여 클러스터에서 업데이트를 인식할 수 있도록 합니다.

절차

다음 절차에서는 시스템 리소스를 구성합니다. 이러한 리소스가 모두 동일한 노드에서 실행되도록 하려면 리소스 그룹 nfsgroup 의 일부로 구성됩니다. 리소스는 그룹에 추가하는 순서대로 시작하고 그룹에 추가된 반대 순서로 중지됩니다. 클러스터의 노드 중 하나에서만 이 절차를 실행합니다.

1.

my_lvm 이라는 LVM 활성화 리소스를 만듭니다. 리소스 그룹 nfsgroup 이 아직 존재하지 않기 때문에 이 명령은 리소스 그룹을 생성합니다.

주의

활성/수동 HA 구성에서 동일한 LVM 볼륨 그룹을 사용하는 LVM 활성화 리소스를 두 개 이상 구성하지 마십시오. 이러한 위험으로 데이터 손상이 발생할 수 있습니다. 또한 활성/수동 HA 구성에서 LVM 활성화 리소스를 복제 리소스로 구성하지 마십시오.

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

2. *클러스터 상태를 확인하여 리소스가 실행 중인지 확인합니다.*

```
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. *클러스터의 Filesystem 리소스를 구성합니다.*

다음 명령은 nfsgroup 리소스 그룹의 일부로 nfsshare 라는 XFS Filesystem 리소스를 구성합니다. 이 파일 시스템은 XFS 파일 시스템으로 LVM 볼륨 구성에서 생성한 LVM 볼륨 그룹과 XFS 파일 시스템을 사용하며, NFS 공유 구성에서 생성한 /nfsshare 디렉터리에 마운트됩니다.

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

options = 매개 변수를 사용하여 **Filesystem** 리소스의 리소스 구성의 일부로 마운트 옵션을 지정할 수있습니다. 전체 구성 옵션에 대해 **pcs resource describe Filesystem** 명령을 실행합니다.

4. **my_lvm** 및 **tekton** 리소스가 실행 중인지 확인합니다.

```
[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. 리소스 그룹 **nfsgroup** 의 일부로 **nfs-daemon** 이라는 **nfsserver** 리소스를 생성합니다.

참고

nfsserver 리소스를 사용하면 **NFS** 서버에서 **NFS** 관련 상태 정보를 저장하는 데 사용하는 디렉터리인 **nfs_shared_infodir** 매개변수를 지정할 수 있습니다.

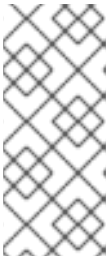
이 속성은 이 내보내기 컬렉션에서 생성한 **Filesystem** 리소스 중 하나의 하위 디렉터리로 설정하는 것이 좋습니다. 이렇게 하면 이 리소스 그룹을 재배치해야 하는 경우 **NFS** 서버에서 다른 노드에서 사용할 수 있게 되는 장치에 상태 저장 정보를 저장합니다. 예에서는 다음을 수행합니다.

- **/octets**는 **Filesystem** 리소스에서 관리하는 **shared-storage** 디렉터리입니다.
- **/tekton/exports/export1** 및 **/tekton/exports/export2** 는 내보내기 디렉토리입니다.
- **/octets/nfsinfo** 는 **nfsserver** 리소스의 **shared-octets** 디렉터리입니다.

```
[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 리소스를 추가하여 /tekton/exports 디렉터리를 내보냅니다. 이러한 리소스는 리소스 그룹 nfsgroup 의 일부입니다. 이렇게 하면 NFSv4 클라이언트용 가상 디렉터리가 빌드됩니다. NFSv3 클라이언트는 이러한 내보내기에도 액세스할 수 있습니다.*



참고

fsid=0 옵션은 NFSv4 클라이언트에 대한 가상 디렉터리를 생성하려는 경우에만 필요합니다. 자세한 내용은 NFS 서버의 /etc/exports 파일에서 fsid 옵션을 구성하는 방법을 참조하십시오.

```
[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 클라이언트가 NFS 공유에 액세스하는 데 사용할 유동 IP 주소 리소스를 추가합니다. 이 리소스는 리소스 그룹 nfsgroup 의 일부입니다. 이 예제 배포 예에서는 유동 IP 주소로 192.168.122.200을 사용하고 있습니다.*

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

8. *전체 NFS 배포가 초기화되면 NFSv3 재부팅 알림을 전송하기 위한 nfsnotify 리소스를 추가합니다. 이 리소스는 리소스 그룹 nfsgroup 의 일부입니다.*

참고

NFS 알림을 올바르게 처리하려면 유동 IP 주소에 NFS 서버와 NFS 클라이언트에서 일관성 있는 호스트 이름이 연결되어 있어야 합니다.

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

9. 리소스 및 리소스 제약 조건을 생성한 후 클러스터 상태를 확인할 수 있습니다. 모든 리소스가 동일한 노드에서 실행되고 있습니다.

```
[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 리소스 구성 테스트

다음 절차에 따라 고가용성 클러스터에서 NFS 리소스 구성을 검증할 수 있습니다. 내보낸 파일 시스템을 NFSv3 또는 NFSv4로 마운트할 수 있습니다.

6.4.1. NFS 내보내기 테스트

1. 클러스터 노드에서 firewalld 데몬을 실행하는 경우 시스템이 모든 노드에서 NFS 액세스에 필요한 포트가 활성화되어 있는지 확인합니다.

2. 클러스터 외부의 노드에서 배포와 동일한 네트워크에 있으며 NFS 공유를 마운트하여 NFS 공유를 볼 수 있는지 확인합니다. 이 예에서는 192.168.122.0/24 네트워크를 사용합니다.

```
# 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. NFSv4로 NFS 공유를 마운트할 수 있는지 확인하려면 NFS 공유를 클라이언트 노드의 디렉터리에 마운트합니다. 마운트 후 내보내기 디렉터리의 콘텐츠가 표시되는지 확인합니다. 테스트 후 공유를 마운트 해제합니다.

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

4. *NFSv3로 NFS 공유를 마운트할 수 있는지 확인합니다. 마운트 후 테스트 파일 clientdatafile1 이 표시되는지 확인합니다. NFSv4와 달리 NFSv3는 가상 파일 시스템을 사용하지 않으므로 특정 내보내기를 마운트해야 합니다. 테스트 후 공유를 마운트 해제합니다.*

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

6.4.2. 장애 조치(failover) 테스트

1. *클러스터 외부의 노드에서 NFS 공유를 마운트하고 NFS 공유 구성에서 생성한 clientdatafile1 파일에 대한 액세스 권한을 확인합니다.*
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_high_availability_clusters/assembly_configuring-active-passive-nfs-server-in-a-cluster-configuring-and-managing-high-availability-clusters#proc_configuring-nfs-share-configuring-ha-nfs

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

2. *클러스터 내의 노드에서 클러스터에서 nfsgroup 을 실행 중인 노드를 확인합니다. 이 예에서 nfsgroup 은 z1.example.com 에서 실행 중입니다.*

```
[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.

클러스터 내의 노드에서 **nfsgroup** 을 실행 중인 노드를 **standby** 모드에서 배치합니다.

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

4.

nfsgroup 이 다른 클러스터 노드에서 성공적으로 시작되었는지 확인합니다.

```
[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.

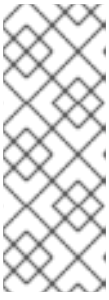
NFS 공유를 마운트한 클러스터 외부의 노드에서 이 외부 노드가 **NFS** 마운트 내의 테스트 파일에 계속 액세스할 수 있는지 확인합니다.

```
# ls nfsshare
clientdatafile1
```

장애 조치(**failover**) 중에 클라이언트에 대해 서비스가 잠시 손실되지만 클라이언트는 사용자 개입 없이 복구해야 합니다. 기본적으로 **NFSv4**를 사용하는 클라이언트는 마운트를 복구하는 데 최대 **90**초가 걸릴 수 있습니다. 이 **90**초는 시작 시 서버에서 관찰한 **NFSv4** 파일 리스 유예 기간을 나타냅니다. **NFSv3** 클라이언트는 몇 초 안에 마운트에 대한 액세스를 복구해야 합니다.

6.

클러스터 내의 노드에서 초기 대기 모드에서 **nfsgroup** 을 실행한 노드를 제거합니다.



참고

대기 모드에서 노드를 제거해도 리소스가 해당 노드로 되돌아가지 않습니다. 이는 리소스의 **resource-stickiness** 값에 따라 달라집니다. **resource-stickiness meta** 속성에 대한 자세한 내용은 현재 노드를 선호하도록 리소스 구성을 참조하십시오.

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

7장. 클러스터의 GFS2 파일 시스템

다음 관리 절차를 사용하여 Red Hat 고가용성 클러스터에서 VMDK2 파일 시스템을 구성합니다.

7.1. 클러스터에서 GFS2 파일 시스템 구성

다음 절차에 따라 Alertmanager2 파일 시스템을 포함하는 Pacemaker 클러스터를 설정할 수 있습니다. 이 예제에서는 2-노드 클러스터의 논리 볼륨 3개에 status2 파일 시스템을 생성합니다.

사전 요구 사항

- 클러스터 노드 모두에서 클러스터 소프트웨어를 설치 및 시작하고 기본 2-노드 클러스터를 생성합니다.
- 클러스터의 펜싱을 구성합니다.

Pacemaker 클러스터를 생성하고 클러스터의 펜싱 구성에 대한 자세한 내용은 Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성을 참조하십시오.

절차

1. 클러스터의 두 노드 모두에서 시스템 아키텍처에 해당하는 복구 스토리지용 리포지토리를 활성화합니다. 예를 들어 x86_64 시스템의 Resilient Storage 리포지토리를 활성화하려면 다음 subscription-manager 명령을 입력할 수 있습니다.

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

복구 스토리지 리포지토리는 High Availability 리포지토리의 상위 세트입니다. 복구 스토리지 리포지토리를 활성화하는 경우 고가용성 리포지토리도 활성화할 필요가 없습니다.

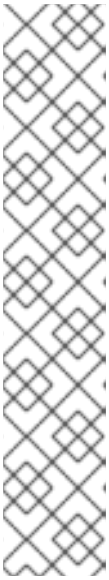
2. 클러스터의 두 노드에서 lvm2-lockd,gfs2-utils, dlm 패키지를 설치합니다. 이러한 패키지를 지원하려면 AppStream 채널 및 복구 스토리지 채널을 구독해야 합니다.

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

3. 클러스터의 두 노드 모두에서 /etc/lvm/ lvm.conf 파일의 use_lvm lockd 구성 옵션을 use_lvmlockd=1 로 설정합니다.

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

4. 글로벌 **Pacemaker** 매개 변수 **no-quorum-policy** 를 **freeze** 로 설정합니다.



참고

기본적으로 **no-quorum-policy** 의 값은 **stop** 으로 설정됩니다. 쿼럼이 유실되면 나머지 파티션의 모든 리소스가 즉시 중지됨을 나타냅니다. 일반적으로 이 기본값은 가장 안전하고 최적의 옵션이지만 대부분의 리소스와 달리, **GFS2**를 사용하려면 쿼럼이 필요합니다. 쿼럼이 **GFS2** 마운트 및 **GFS2** 마운트 자체를 사용하는 애플리케이션 모두 손실되면 제대로 중지할 수 없습니다. 쿼럼 없이 이러한 리소스를 중지하려고 하면 결국 쿼럼이 손실될 때마다 전체 클러스터가 펜싱됩니다.

이 문제를 해결하려면 **GFS2**가 사용 중인 경우 **no-quorum-policy** 를 **freeze** 로 설정합니다. 즉, 쿼럼이 손실되면 쿼럼이 복구될 때까지 나머지 파티션은 아무 작업도 수행하지 않습니다.

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

5. **dlm** 리소스를 설정합니다. 이는 클러스터에서 **GFS2** 파일 시스템을 구성하는 데 필요한 종속 항목입니다. 이 예제에서는 **locking** 라는 리소스 그룹의 일부로 **dlm** 리소스를 생성합니다.

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

6. 클러스터의 두 노드에서 리소스 그룹을 활성화할 수 있도록 잠금 리소스 그룹을 복제합니다.

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

7. **lvmlockd** 리소스를 잠금 리소스 그룹의 일부로 설정합니다.

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

8. 클러스터 상태를 확인하여 클러스터의 두 노드에서 잠금 리소스 그룹이 시작되었는지 확인합니다.

```
[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. 클러스터의 한 노드에서 두 개의 공유 볼륨 그룹을 만듭니다. 하나의 볼륨 그룹에는 두 개의 GFS2 파일 시스템이 포함되며 다른 볼륨 그룹에는 하나의 GFS2 파일 시스템이 포함됩니다.

참고

LVM 볼륨 그룹에 iSCSI 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 Pacemaker를 시작하기 전에 서비스를 시작하는 것이 좋습니다. Pacemaker 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 Pacemaker에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성을 참조하십시오.

다음 명령은 /dev/vdb 에 shared 볼륨 그룹 shared_vg1 을 생성합니다.

```
[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...
```

다음 명령은 /dev/ hiera 에 shared 볼륨 그룹 shared_vg2 를 생성합니다.

```
[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.

클러스터의 두 번째 노드에서 다음을 수행합니다.

a.

lvm.conf** 파일에서 **use_devicesfile = 1** 매개 변수를 사용하여 장치 파일을 사용하는 경우 공유 장치를 장치 파일에 추가합니다 **This feature is enabled by default.

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

b.

각 공유 볼륨 그룹에 대해 잠금 관리자를 시작합니다.

```
[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.

*클러스터의 한 노드에서 공유 논리 볼륨을 생성하고 **GFS2** 파일 시스템으로 볼륨을 포맷합니다. 파일 시스템을 마운트하는 각 노드에 저널 **1**개가 필요합니다. 클러스터의 각 노드에 충분한 저널을 생성해야 합니다. 잠금 테이블 이름의 형식은 **ClusterName:FSName** 입니다. 여기서 **ClusterName** 은 **GFS2** 파일 시스템이 생성되는 클러스터의 이름이고 **FSName** 은 클러스터 전체에서 모든 **lock_dlm** 파일 시스템에 대해 고유해야 하는 파일 시스템 이름입니다.*

```
[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.

*각 논리 볼륨에 대한 **LVM** 활성화 리소스를 생성하여 모든 노드에서 논리 볼륨을 자동으로 활성화합니다.*

a.

***shared_vg1** 볼륨 그룹에 대해 **shared_lv1** 논리 볼륨에 대해 **sharedlv1** 이라는 **LVM** 활성화 리소스를 만듭니다. 또한 이 명령은 리소스를 포함하는 **shared_vg1** 리소스 그룹도 생성합니다. 이 예에서 리소스 그룹의 이름은 논리 볼륨을 포함하는 공유 볼륨 그룹과 동일합니다.*

-

```
[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.

shared_vg1 볼륨 그룹에서 shared_lv2 에 대해 sharedlv2 라는 LVM 활성화 리소스를 만듭니다. 이 리소스는 shared_vg1 리소스 그룹의 일부이기도 합니다.

```
[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.

shared_vg2 볼륨 그룹에서 shared_lv1 에 대해 sharedlv3 이라는 LVM 활성화 리소스를 만듭니다. 또한 이 명령은 리소스를 포함하는 shared_vg2 리소스 그룹도 생성합니다.

```
[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. *두 개의 새 리소스 그룹을 복제합니다.*

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

14. *dlm 및 lvmlockd 리소스가 포함된 잠금 리소스 그룹을 먼저 시작하도록 순서 제한 조건을 구성합니다.*

```
[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. *tekton 1 및 pxe 2 리소스 그룹이 잠금 리소스 그룹과 동일한 노드에서 시작하도록 공동 배치 제한 조건을 구성합니다.*

```
[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. *클러스터의 두 노드에서 논리 볼륨이 활성화되어 있는지 확인합니다. 몇 초 정도 지연될 수 있습니다.*

-

```
[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. *파일 시스템 리소스를 생성하여 각 **GFS2** 파일 시스템을 모든 노드에 자동으로 마운트합니다.*

__Pacemaker__ 클러스터 리소스로 관리되므로 파일 시스템을 __/etc/fstab__ 파일에 추가하지 않아야 합니다. 마운트 옵션은 __options=__ 옵션을 사용하여 리소스 구성의 일부로 지정할 수 있습니다. __pcs resource describe Filesystem__ 명령을 실행하여 전체 구성 옵션을 표시합니다.

다음 명령은 파일 시스템 리소스를 생성합니다. 이러한 명령은 해당 파일 시스템의 논리 볼륨 리소스를 포함하는 각 리소스를 리소스 그룹에 추가합니다.

```
[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
```

검증 단계

1. *__GFS2__ 파일 시스템이 클러스터의 두 노드에 마운트되었는지 확인합니다.*

```
[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. *클러스터 상태를 확인합니다.*

```
[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 ]

...
```

추가 리소스

- ***GFS2 파일 시스템 구성***
- ***Microsoft Azure에서 Red Hat High Availability 클러스터 구성***
- ***AWS에서 Red Hat High Availability 클러스터 구성***

- *Google Cloud Platform에서 Red Hat High Availability Cluster 구성*

7.2. 클러스터에서 암호화된 GFS2 파일 시스템 구성

다음 절차에 따라 **LUKS** 암호화된 **VMDK2** 파일 시스템을 포함하는 **Pacemaker** 클러스터를 생성할 수 있습니다. 이 예에서는 논리 볼륨에 하나의 **VMDK2** 파일 시스템을 생성하고 파일 시스템을 암호화합니다. 암호화된 **GFS2** 파일 시스템은 **LUKS** 암호화를 지원하는 **crypt** 리소스 에이전트를 사용하여 지원됩니다.

이 과정에는 세 가지 부분이 있습니다.

- **Pacemaker** 클러스터에서 공유 논리 볼륨 구성
- 논리 볼륨 암호화 및 **crypt** 리소스 생성
- 암호화된 논리 볼륨을 **GFS2** 파일 시스템으로 포맷하고 클러스터용 파일 시스템 리소스 생성

7.2.1. Pacemaker 클러스터에서 공유 논리 볼륨 구성

사전 요구 사항

- 두 클러스터 노드에서 클러스터 소프트웨어를 설치하고 시작하고 기본 **2**-노드 클러스터를 생성합니다.
- 클러스터의 펜싱을 구성합니다.

Pacemaker 클러스터를 생성하고 클러스터의 펜싱 구성에 대한 자세한 내용은 *Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성을* 참조하십시오.

절차

1. 클러스터의 두 노드 모두에서 시스템 아키텍처에 해당하는 복구 스토리지용 리포지토리를 활성화합니다. 예를 들어 **x86_64** 시스템의 **Resilient Storage** 리포지토리를 활성화하려면 다음 **subscription-manager** 명령을 입력할 수 있습니다.

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

복구 스토리지 리포지토리는 **High Availability** 리포지토리의 상위 세트입니다. 복구 스토리지 리포지토리를 활성화하는 경우 고가용성 리포지토리도 활성화할 필요가 없습니다.

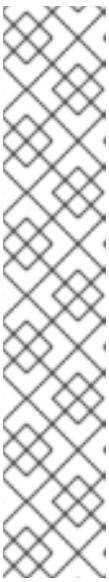
2. 클러스터의 두 노드에서 **lvm2-lockd,gfs2-utils, dlm** 패키지를 설치합니다. 이러한 패키지를 지원하려면 **AppStream** 채널 및 복구 스토리지 채널을 구독해야 합니다.

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

3. 클러스터의 두 노드 모두에서 **/etc/lvm/ lvm.conf** 파일의 **use_lvm lockd** 구성 옵션을 **use_lvmlockd=1** 로 설정합니다.

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

4. 글로벌 **Pacemaker** 매개 변수 **no-quorum-policy** 를 **freeze** 로 설정합니다.



참고

기본적으로 **no-quorum-policy** 값은 **stop** 으로 설정되어 쿼럼이 손실되면 나머지 파티션의 모든 리소스가 즉시 중지됨을 나타냅니다. 일반적으로 이 기본값은 가장 안전하고 최적의 옵션이지만 대부분의 리소스와 달리, **GFS2**를 사용하려면 쿼럼이 필요합니다. 쿼럼이 **GFS2** 마운트 및 **GFS2** 마운트 자체를 사용하는 애플리케이션 모두 손실되면 제대로 중지할 수 없습니다. 쿼럼 없이 이러한 리소스를 중지하려고 하면 결국 쿼럼이 손실될 때마다 전체 클러스터가 펜싱됩니다.

이 문제를 해결하려면 **GFS2**가 사용 중인 경우 **no-quorum-policy** 를 **freeze** 로 설정합니다. 즉, 쿼럼이 손실되면 쿼럼이 복구될 때까지 나머지 파티션은 아무 작업도 수행하지 않습니다.

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

5. **dlm** 리소스를 설정합니다. 이는 클러스터에서 **GFS2** 파일 시스템을 구성하는 데 필요한 종속 항목입니다. 이 예제에서는 **locking** 라는 리소스 그룹의 일부로 **dlm** 리소스를 생성합니다.

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

6. 클러스터의 두 노드에서 리소스 그룹을 활성화할 수 있도록 잠금 리소스 그룹을 복제합니다.

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

7. 그룹 잠금 의 일부로 **lvmlockd** 리소스를 설정합니다.

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

8. 클러스터 상태를 확인하여 클러스터의 두 노드에서 잠금 리소스 그룹이 시작되었는지 확인합니다.

```
[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. 클러스터의 한 노드에서 공유 볼륨 그룹을 생성합니다.

참고

LVM 볼륨 그룹에 **iSCSI** 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 **Pacemaker**를 시작하기 전에 서비스를 시작하는 것이 좋습니다. **Pacemaker** 클러스터에서 사용하는 원격 물리 볼륨의 시작 순서를 구성하는 방법에 대한 자세한 내용은 **Pacemaker**에서 *관리하지 않는 리소스 종속 항목의 시작 순서* 구성을 참조하십시오.

다음 명령은 **/dev/sda1** 에 공유 볼륨 그룹 **shared_vg1** 을 생성합니다.

```
[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. 클러스터의 두 번째 노드에서 다음을 수행합니다.

    a. lvm.conf 파일에서 use_devicesfile = 1 매개변수를 사용하여 장치 파일을 사용하는 경우 클러스터의 두 번째 노드의 장치 파일에 공유 장치를 추가합니다. 이 기능은 기본적으로 활성화되어 있습니다.

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

    b. 공유 볼륨 그룹의 잠금 관리자를 시작합니다.

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

11. 클러스터의 한 노드에서 공유 논리 볼륨을 생성합니다.

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

12. 논리 볼륨에 대한 LVM 활성화 리소스를 생성하여 모든 노드에서 논리 볼륨을 자동으로 활성화합니다.

    다음 명령은 shared_vg1 볼륨 그룹 shared_lv1 에 대해 sharedlv1 이라는 LVM 활성화 리소스를 생성합니다. 또한 이 명령은 리소스를 포함하는 shared_vg1 리소스 그룹도 생성합니다. 이 예에서 리소스 그룹의 이름은 논리 볼륨을 포함하는 공유 볼륨 그룹과 동일합니다.

```
[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. 새 리소스 그룹을 복제합니다.

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

14. *dlm 및 lvmlockd 리소스가 포함된 잠금 리소스 그룹을 먼저 시작하도록 순서 제한 조건을 구성합니다.*

```
[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. *tekton 1 및 pxe 2 리소스 그룹이 잠금 리소스 그룹과 동일한 노드에서 시작하도록 공동 배치 제한 조건을 구성합니다.*

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

검증 단계

클러스터의 두 노드에서 논리 볼륨이 활성 상태인지 확인합니다. 몇 초 정도 지연될 수 있습니다.

```
[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. 논리 볼륨을 암호화하고 crypt 리소스 생성

사전 요구 사항

- *Pacemaker 클러스터에서 공유 논리 볼륨을 구성했습니다.*

절차

1. *클러스터의 한 노드에서 crypt 키를 포함할 새 파일을 생성하고 root에서만 읽을 수 있도록 파일에 대한 권한을 설정합니다.*

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

2. *crypt 키를 만듭니다.*

```
[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. 설정한 권한을 유지하기 위해 **-p** 매개변수를 사용하여 클러스터의 다른 노드에 **crypt** 키 파일을 배포합니다.

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

4. 암호화된 **GFS2** 파일 시스템을 구성할 **LVM** 볼륨에 암호화된 장치를 생성합니다.

```
[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** 볼륨 그룹의 일부로 **crypt** 리소스를 생성합니다.

```
[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"
```

검증 단계

crypt 리소스가 **crypt** 장치를 생성했는지 확인합니다. 이 예에서 **/dev/mapper/luks_lv1** 입니다.

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

7.2.3. VMDK2 파일 시스템으로 암호화된 논리 볼륨을 포맷하고 클러스터에 대한 파일 시스템 리소스를 생성합니다.

사전 요구 사항

- 논리 볼륨을 암호화하여 **crypt** 리소스를 생성했습니다.

절차

1.

클러스터의 한 노드에서 **GFS2** 파일 시스템으로 볼륨을 포맷합니다. 파일 시스템을 마운트하는 각 노드에 저널 **1**개가 필요합니다. 클러스터의 각 노드에 충분한 저널을 생성해야 합니다. 잠금 테이블 이름의 형식은 **ClusterName:FSName** 입니다. 여기서 **ClusterName** 은 **GFS2** 파일 시스템이 생성되는 클러스터의 이름이고 **FSName** 은 클러스터 전체에서 모든 **lock_dlm** 파일 시스템에 대해 고유해야 하는 파일 시스템 이름입니다.

```
[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.

파일 시스템 리소스를 생성하여 모든 노드에 자동으로 **GFS2** 파일 시스템을 마운트합니다.

Pacemaker 클러스터 리소스로 관리되므로 파일 시스템을 **/etc/fstab** 파일에 추가하지 마십시오. 마운트 옵션은 **options=** 옵션을 사용하여 리소스 구성의 일부로 지정할 수 있습니다. 전체 구성 옵션에 대해 **pcs resource describe Filesystem** 명령을 실행합니다.

다음 명령은 파일 시스템 리소스를 생성합니다. 이 명령은 해당 파일 시스템의 논리 볼륨 리소스를 포함하는 리소스 그룹에 리소스를 추가합니다.

```
[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
```

검증 단계

1.

GFS2 파일 시스템이 클러스터의 두 노드에 마운트되었는지 확인합니다.

```
[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. *클러스터 상태를 확인합니다.*

```
[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 ]
...
```

추가 리소스

- ***GFS2** 파일 시스템 구성*

8장. RED HAT HIGH AVAILABILITY 클러스터에서 활성/활성 SAMBA 서버 구성

Red Hat High Availability Add-On은 활성/활성 클러스터 구성에서 Samba 구성을 지원합니다. 다음 예제에서는 2-노드 RHEL 클러스터에서 활성/활성 Samba 서버를 구성하고 있습니다.

Samba에 대한 지원 정책에 대한 자세한 내용은 RHEL High Availability - ctdb General Policies and Support Policies for RHEL Resilient Storage - Red Hat 고객 포털의 다른 프로토콜을 통해 gfs2 콘텐츠 내보내기 를 참조하십시오.

활성/활성 클러스터에서 Samba를 구성하려면 다음을 수행합니다.

1. ScanSetting2 파일 시스템 및 관련 클러스터 리소스를 구성합니다.
2. 클러스터 노드에서 Samba를 구성합니다.
3. Samba 클러스터 리소스를 구성합니다.
4. 구성한 Samba 서버를 테스트합니다.

8.1. 고가용성 클러스터에서 SAMBA 서비스에 대한 SCANSETTING2 파일 시스템 구성

Pacemaker 클러스터에서 활성/활성 Samba 서비스를 구성하기 전에 클러스터에 대한#1772 파일 시스템을 구성합니다.

사전 요구 사항

- 각 노드에 펜싱이 구성된 2-노드 Red Hat High Availability 클러스터
- 각 클러스터 노드에 사용 가능한 공유 스토리지
- AppStream 채널에 대한 서브스크립션 및 각 클러스터 노드의 복구 스토리지 채널

Pacemaker 클러스터를 생성하고 클러스터의 펜싱 구성에 대한 자세한 내용은 Pacemaker를 사용하여 Red Hat High-Availability 클러스터 생성을 참조하십시오.

절차

1. *클러스터의 두 노드 모두에서 다음 초기 설정 단계를 수행합니다.*

   a. *시스템 아키텍처에 해당하는 탄력적 스토리지의 리포지토리를 활성화합니다. 예를 들어 x86_64 시스템의 복구 스토리지 리포지토리를 활성화하려면 다음 subscription-manager 명령을 입력합니다.*

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

   탄력적 스토리지 리포지토리는 고가용성 리포지토리의 상위 세트입니다. 탄력적 스토리지 리포지토리를 활성화하면 고가용성 리포지토리도 활성화할 필요가 없습니다.

   b. *lvm2-lockd,gfs2-utils, dlm 패키지를 설치합니다.*

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

   c. */etc/lvm/lvm.conf 파일에서 use_lvmlockd 구성 옵션을 use_lvmlockd=1 로 설정합니다.*

   ```
   ...

   use_lvmlockd = 1

   ...
   ```

2. *클러스터의 한 노드에서 글로벌 Pacemaker 매개변수 no-quorum-policy 를 동결 하도록 설정합니다.*

참고

*기본적으로 **no-quorum-policy** 의 값은 **stop** 으로 설정됩니다. 쿼럼이 유실되면 나머지 파티션의 모든 리소스가 즉시 중지됨을 나타냅니다. 일반적으로 이 기본값은 가장 안전하고 최적의 옵션이지만 대부분의 리소스와 달리, **GFS2**를 사용하려면 쿼럼이 필요합니다. 쿼럼이 **GFS2** 마운트 및 **GFS2** 마운트 자체를 사용하는 애플리케이션 모두 손실되면 제대로 중지할 수 없습니다. 쿼럼 없이 이러한 리소스를 중지하려고 하면 결국 쿼럼이 손실될 때마다 전체 클러스터가 펜싱됩니다.*

*이 문제를 해결하려면 **GFS2**가 사용 중인 경우 **no-quorum-policy** 를 **freeze** 로 설정합니다. 즉, 쿼럼이 손실되면 쿼럼이 복구될 때까지 나머지 파티션은 아무 작업도 수행하지 않습니다.*

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

3. *dlm 리소스를 설정합니다. 이는 클러스터에서 **GFS2** 파일 시스템을 구성하는 데 필요한 종속 항목입니다. 이 예제에서는 **locking** 라는 리소스 그룹의 일부로 **dlm** 리소스를 생성합니다. 이전에 클러스터의 펜싱을 구성하지 않은 경우 이 단계가 실패하고 **pcs status** 명령으로 리소스 실패 메시지가 표시됩니다.*

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

4. *클러스터의 두 노드에서 리소스 그룹을 활성화할 수 있도록 잠금 리소스 그룹을 복제합니다.*

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

5. *lvmlockd 리소스를 잠금 리소스 그룹의 일부로 설정합니다.*

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

6. *공유 장치 **/dev/vdb** 에 물리 볼륨 및 공유 볼륨 그룹을 생성합니다. 이 예제에서는 공유 볼륨 그룹 **cECDHE_vg** 를 생성합니다.*

```
[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.

클러스터의 두 번째 노드에서 다음을 수행합니다.

8.

***lvm.conf** 파일에서 **use_devicesfile = 1** 매개변수를 사용하여 장치 파일을 사용하는 경우 클러스터의 두 번째 노드의 장치 파일에 공유 장치를 추가합니다. 이 기능은 기본적으로 활성화되어 있습니다.*

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

a.

공유 볼륨 그룹의 잠금 관리자를 시작합니다.

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

9.

*클러스터의 한 노드에서 **ovnDB**에서 내부 잠금에 독점적으로 사용할 **Octavia2** 파일 시스템으로 논리 볼륨을 생성하고 볼륨을 포맷합니다. 배포에서 여러 공유를 내보내는 경우에도 클러스터에는 하나의 파일 시스템만 필요합니다.*

***mkfs.gfs2** 명령의 **-t** 옵션으로 잠금 테이블 이름을 지정하는 경우 지정하는 **clustername:filesystemname** 의 첫 번째 구성 요소가 클러스터의 이름과 일치하는지 확인합니다. 이 예에서 클러스터 이름은 **my_cluster** 입니다.*

```
[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**를 통해 공유할 각 **VMDK2** 파일 시스템에 대한 논리 볼륨을 생성하고, **CloudEvent2** 파일 시스템으로 볼륨을 포맷합니다. 이 예제에서는 단일 **VMDK2** 파일 시스템 및 **Samba** 공유를 생성하지만 여러 파일 시스템 및 공유를 생성할 수 있습니다.*

```
[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.

*필요한 공유 볼륨이 활성화되도록 **LVM_ LoadBalancerivate** 리소스를 설정합니다. 이 예제에서는 리소스 그룹 **shared _vg** 의 일부로 **LVM _ECDHEivate** 리소스를 생성한 다음 해당 리소스 그룹을 복제하여 클러스터의 모든 노드에서 실행됩니다.*

필요한 순서 제약 조건을 구성하기 전에 자동으로 시작되지 않도록 리소스를 비활성화한 대로 생성합니다.

```
[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 리소스 그룹의 멤버보다 먼저 잠금 리소스 그룹의 모든 멤버를 시작하도록 순서 제한 조건을 구성합니다.*

```
[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 활성화 리소스를 활성화합니다.*

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

14. *클러스터의 한 노드에서 다음 단계를 수행하여 필요한 Filesystem 리소스를 생성합니다.*

a. *이전에 LVM 볼륨에 구성한 VMDK2 파일 시스템을 사용하여 복제된 리소스로 Filesystem 리소스를 생성합니다. 이렇게 하면 Pacemaker가 파일 시스템을 마운트 및 관리하도록 구성됩니다.*

참고

Pacemaker 클러스터 리소스로 관리되므로 파일 시스템을 /etc/fstab 파일에 추가하지 않아야 합니다. options = 옵션을 사용하여 리소스 구성의 일부로 마운트 옵션을 지정할 수있습니다. pcs resource describe Filesystem 명령을 실행하여 전체 구성 옵션을 표시합니다.

```
[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. *공유 볼륨 그룹이 shared_vg 를 시작한 후 Pacemaker에서 파일 시스템을 마운트하도록 순서 제약 조건을 구성합니다.*

```
[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. 고가용성 클러스터에서 SAMBA 구성

Pacemaker 클러스터에서 **Samba** 서비스를 구성하려면 클러스터의 모든 노드에서 서비스를 구성합니다.

사전 요구 사항

- 고가용성 클러스터에서 **Samba** 서비스에 대한 **devfile2** 파일 시스템 구성에 설명된 대로 **SMT2** 파일 시스템으로 구성된 **2-**노드 **Red Hat High Availability** 클러스터 입니다.
- **Samba** 공유에 사용할 **status2** 파일 시스템에 생성된 공용 디렉터리입니다. 이 예에서 디렉터리는 **/srv/ECDHE/share1** 입니다.
- 이 클러스터에서 내보낸 **Samba** 공유에 액세스하는 데 사용할 수 있는 공용 가상 **IP** 주소입니다.

절차

1. 클러스터의 두 노드 모두에서 **Samba** 서비스를 구성하고 공유 정의를 설정합니다.
   a. **Samba** 및 **IRQDB** 패키지를 설치합니다.

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

   b. **ctdb,>-< ,nmb, winbind** 서비스가 실행되지 않고 부팅 시 시작되지 않는지 확인합니다.

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

   c. **/etc/ECDHE/>-<.conf** 파일에서 **Samba** 서비스를 구성하고 하나의 공유를 사용하는 독립 실행형 서버의 다음 예제와 같이 공유 정의를 설정합니다.

   ```
   [global]
   ```

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

d.

/etc/ECDHE/>-<.conf 파일을 확인합니다.

```
# testparm
```

2.

*클러스터의 두 노드에서 **CloudEventDB**를 구성합니다.*

a.

/etc/ctdb/nodes 파일을 만들고 이 예제 노드 파일과 같이 클러스터 노드의 IP 주소를 추가합니다.

```
192.0.2.11
192.0.2.12
```

b.

/etc/ctdb/public_addresses 파일을 만들고 클러스터 공용 인터페이스의 IP 주소와 네트워크 장치 이름을 파일에 추가합니다. public_addresses 파일에 IP 주소를 할당할 때 이러한 주소가 사용되지 않고 해당 주소가 의도한 클라이언트에서 라우팅 가능한지 확인합니다. /etc/ctdb/public_addresses 파일의 각 항목에 있는 두 번째 필드는 해당 공용 주소에 대한 클러스터 시스템에서 사용할 인터페이스입니다. 이 예제 public_addresses 파일에서 enp1s0 인터페이스는 모든 공용 주소에 사용됩니다.

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

*클러스터의 공용 인터페이스는 클라이언트가 네트워크에서 **Samba**에 액세스하는 데 사용하는 인터페이스입니다. 로드 밸런싱을 위해 클러스터의 각 공용 **IP** 주소에 대한 **A** 레코드를 **DNS** 영역에 추가합니다. 이러한 각 레코드는 동일한 호스트 이름으로 확인되어야 합니다. 클라이언트는 호스트 이름을 사용하여 **Samba**에 액세스하고 **DNS**는 클러스터의 다른 노드에 클라이언트를 배포합니다.*

c.

firewalld 서비스를 실행하는 경우 ctdb 및 samba 서비스에 필요한 포트를 활성화합니다.

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

3. 클러스터의 노드 1에서 SELinux 컨텍스트를 업데이트합니다.

   a. Chrony2 공유에서 SELinux 컨텍스트를 업데이트합니다.

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

   b. Samba에서 공유된 디렉터리에서 SELinux 컨텍스트를 업데이트합니다.

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

추가 리소스

- 이 예와 같이 Samba를 독립 실행형 서버로 구성하는 방법에 대한 자세한 내용은 네트워크 파일 서비스 구성 및 사용의 서버로 Samba 사용 장을 참조하십시오.
- BIND 기본 서버에서 전달 영역 설정.

8.3. SAMBA 클러스터 리소스 구성

2-노드 고가용성 클러스터의 두 노드 모두에서 Samba 서비스를 구성한 후 클러스터에 대한 Samba 클러스터 리소스를 구성합니다.

사전 요구 사항

- 고가용성 클러스터에서 Samba 서비스에 대한 devfile2 파일 시스템 구성에 설명된 대로 SMT2 파일 시스템으로 구성된 2-노드 Red Hat High Availability 클러스터 입니다.
- 고가용성 클러스터에서 Samba 구성에 설명된 대로 두 클러스터 노드에 Samba가 구성된 Samba 서비스입니다.

절차

1. 클러스터의 한 노드에서 Samba 클러스터 리소스를 구성합니다.

a.

samba-group 그룹에서 ScanSettingDB 리소스를 만듭니다. CloudEventDB 리소스 에이전트는 pcs 명령으로 지정된 ctdb_* 옵션을 사용하여 IKEvDB 구성 파일을 생성합니다. 필요한 순서 제약 조건을 구성하기 전에 자동으로 시작되지 않도록 리소스를 비활성화한 대로 생성합니다.

```
[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 리소스 그룹을 복제합니다.

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

c.

samba-group 의 리소스보다 먼저 모든 Filesystem 리소스가 실행 중인지 확인하기 위해 순서 제약 조건을 만듭니다.

```
[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.

리소스 그룹 samba -group 에서 samba 리소스를 만듭니다. 이를 통해 추가된 순서에 따라 CloudEventDB와 Samba 간의 암시적 순서 제약 조건이 생성됩니다.

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

e.

ctdb 및 samba 리소스를 활성화합니다.

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

f.

모든 서비스가 성공적으로 시작되었는지 확인합니다.

참고

CloudEventDB가 Samba를 시작하고, 공유를 내보내고, 안정화하는 데 몇 분이 걸릴 수 있습니다. 이 프로세스가 완료되기 전에 클러스터 상태를 확인하는 경우 samba 서비스가 아직 실행되지 않은 것으로 표시될 수 있습니다.

```
[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. *클러스터의 두 노드 모두에서 테스트 공유 디렉터리의 로컬 사용자를 추가합니다.*

a. *사용자를 추가합니다.*

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

b. *사용자의 암호를 설정합니다.*

```
# passwd example_user
```

c. *사용자의 **BaseOS** 암호를 설정합니다.*

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

d. ***Samba** 데이터베이스에서 사용자를 활성화합니다.*

```
# smbpasswd -e example_user
```

e. ***Samba** 사용자의 **pacemaker2** 공유에 대한 파일 소유권 및 권한을 업데이트합니다.*

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

8.4. 클러스터형 SAMBA 구성 확인

클러스터형 **Samba** 구성이 성공하면 **Samba** 공유를 마운트할 수 있습니다. 공유를 마운트한 후 **Samba** 공유를 내보내는 클러스터 노드를 사용할 수 없게 되는 경우 **Samba** 복구를 테스트할 수 있습니다.

절차

1. 클러스터 노드의 **/etc/ctdb/public_addresses** 파일에 구성된 하나 이상의 공용 **IP** 주소에 액세스할 수 있는 시스템에서 이러한 공용 **IP** 주소 중 하나를 사용하여 **Samba** 공유를 마운트합니다.

```
[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. 파일 시스템이 마운트되었는지 확인합니다.

```
[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,ui
d=0,noforceuid,gid=0,noforcegid,addr=192.0.2.201,unix,posixpaths,serverino,mapposi
x,acl,rsize=1048576,wsize=65536,echo_interval=60,actimeo=1,user=example_user)
```

3. 마운트된 파일 시스템에 파일을 만들 수 있는지 확인합니다.

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

4. **Samba** 공유를 내보내는 클러스터 노드를 결정합니다.

   a. 각 클러스터 노드에서 **public_addresses** 파일에 지정된 인터페이스에 할당된 **IP** 주소를 표시합니다. 다음 명령은 각 노드의 **enp1s0** 인터페이스에 할당된 **IPv4** 주소를 표시합니다.

```
[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 명령 출력에서 공유를 마운트할 때 mount 명령으로 지정한 IP 주소로 노드를 찾습니다.*

이 예에서 마운트 명령에 지정된 IP 주소는 192.0.2.201입니다. ip 명령의 출력은 IP 주소 192.0.2.201이 z1.example.com 에 할당되었음을 보여줍니다.

5. *Samba 공유를 승격 모드로 내보내는 노드를 배치하여 노드가 클러스터 리소스를 호스팅할 수 없게 됩니다.*

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

6. *파일 시스템을 마운트한 시스템에서 파일 시스템에 파일을 계속 생성할 수 있는지 확인합니다.*

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

7. *생성한 파일을 삭제하여 파일 시스템이 성공적으로 마운트되었는지 확인합니다. 더 이상 파일 시스템을 마운트할 필요가 없는 경우 이 시점에서 마운트 해제합니다.*

```
[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. *클러스터 노드 중 하나에서 클러스터 서비스를 이전에 준비한 노드로 복원합니다. 이 경우 서비스를 해당 노드로 다시 이동할 필요는 없습니다.*

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

9장. PCSD WEB UI 시작하기

pcsd Web UI는 Pacemaker/Corosync 클러스터를 생성하고 구성하는 그래픽 사용자 인터페이스입니다.

9.1. PCSD WEB UI 설정

pcsd Web UI를 사용하여 다음 절차에 따라 클러스터를 구성하도록 시스템을 설정합니다.

사전 요구 사항

- Pacemaker 구성 도구가 설치되어 있습니다.
- 시스템이 클러스터 구성에 맞게 설정되어 있습니다.

클러스터 소프트웨어 설치 및 클러스터 구성을 위한 시스템 설정에 대한 지침은 클러스터 소프트웨어 설치를 참조하십시오.

절차

1. 모든 시스템에서 브라우저를 열고 클러스터 노드 중 하나를 지정하여 다음 URL로 엽니다(이는 https 프로토콜을 사용하는 점에 유의하십시오). 그러면 pcsd Web UI 로그인 화면이 나타납니다.

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

2. hacluster 사용자로 로그인합니다. 그러면 클러스터 페이지가 표시됩니다.

9.2. 고가용성 PCSD WEB UI 구성

pcsd Web UI를 사용하면 클러스터의 노드 중 하나에 연결하여 클러스터 관리 페이지를 표시합니다. 연결 중인 노드가 다운되거나 사용 불가능하게 되면 브라우저를 열고 클러스터의 다른 노드를 지정하는 URL로 브라우저를 열어 클러스터에 다시 연결할 수 있습니다. 그러나 고가용성을 위해 pcsd Web UI 자체를 구성할 수 있습니다. 이 경우 새 URL을 입력하지 않고도 클러스터를 계속 관리할 수 있습니다.

절차

*고가용성을 위해 **pcsd Web UI**를 구성하려면 다음 단계를 수행합니다.*

1. */etc/sysconfig/pcsd 구성 파일에서 **PCSD_SSL_CERT_SYNC_ENABLED** 를 **true** 로 설정하여 클러스터 노드에서 **pcsd** 인증서가 동기화되었는지 확인합니다. 인증서 동기화를 활성화하면 **pcsd** 가 클러스터 설정 및 **node add** 명령의 인증서를 동기화합니다. **PCSD_SSL_CERT_SYNC_ENABLED** 는 기본적으로 **false** 로 설정됩니다.*

2. ***pcsd Web UI**에 연결하는 데 사용할 유동 **IP** 주소인 **IPaddr2** 클러스터 리소스를 생성합니다. **IP** 주소는 물리적 노드와 이미 연결되어 있지 않아야 합니다. **IPaddr2** 리소스의 **NIC** 장치가 지정되지 않은 경우 유동 **IP**는 노드의 정적으로 할당된 **IP** 주소 중 하나와 동일한 네트워크에 있어야 합니다. 그렇지 않으면 유동 **IP** 주소를 할당하기 위해 **NIC** 장치를 올바르게 탐지할 수 없습니다.*

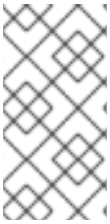
3. ***pcsd** 와 함께 사용할 사용자 정의 **SSL** 인증서를 생성하고 **pcsd Web UI**에 연결하는 데 사용되는 노드 주소에 유효한지 확인합니다.*

   a. *사용자 정의 **SSL** 인증서를 생성하려면 와일드카드 인증서를 사용하거나 주체 대체 이름 인증서 확장을 사용할 수 있습니다. **Red Hat Certificate System**에 대한 자세한 내용은 **Red Hat Certificate System Administration Guide** 를 참조하십시오.*

   b. ***pcs pcsd certkey** 명령을 사용하여 **pcsd** 의 사용자 정의 인증서를 설치합니다.*

   c. ***pcsd** 인증서를 **pcs pcsd sync-certificates** 명령과 함께 클러스터의 모든 노드에 동기화합니다.*

4. *클러스터 리소스로 구성한 유동 **IP** 주소를 사용하여 **pcsd Web UI**에 연결합니다.*



참고

*고가용성을 위해 **pcsd Web UI**를 구성하는 경우에도 연결하는 노드가 중단될 때 다시 로그인하라는 메시지가 표시됩니다.*

10장. RED HAT HIGH AVAILABILITY 클러스터에서 펜싱 구성

응답하지 않는 노드는 여전히 데이터에 액세스 중일 수 있습니다. 데이터가 안전하다는 것을 확인하는 유일한 방법은 **STONITH**를 사용하여 노드를 차단하는 것입니다. **STONITH**는 **"Shoot The Other Node In The head"**의 약어이며 악성 노드 또는 동시 액세스로 데이터가 손상되지 않도록 보호합니다. **STONITH**를 사용하면 다른 노드에서 데이터에 액세스할 수 있도록 허용하기 전에 노드가 실제로 오프라인 상태인지 확인할 수 있습니다.

또한 **STONITH**에는 클러스터형 서비스를 중지할 수 없는 경우에도 플레이 할 역할이 있습니다. 이 경우 클러스터는 **STONITH**를 사용하여 전체 노드를 오프라인으로 강제하여 서비스를 다른 위치에서 안전하게 시작할 수 있습니다.

Red Hat High Availability 클러스터의 펜싱 및 중요성에 대한 자세한 내용은 **Red Hat High Availability Cluster**의 **Fencing**을 참조하십시오.

클러스터 노드에 대한 펜스 장치를 구성하여 **Pacemaker** 클러스터에서 **STONITH**를 구현합니다.

10.1. 사용 가능한 펜스 에이전트 및 옵션 표시

다음 명령을 사용하여 사용 가능한 펜싱 에이전트와 특정 펜싱 에이전트에서 사용 가능한 옵션을 볼 수 있습니다.

참고

시스템의 하드웨어에 따라 클러스터에 사용할 펜싱 장치 유형이 결정됩니다. 지원되는 플랫폼 및 아키텍처 및 다양한 펜싱 장치에 대한 자세한 내용은 **RHEL** 고가용성 클러스터에 대한 지원 정책의 클러스터 플랫폼 및 아키텍처 섹션을 참조하십시오.

다음 명령을 실행하여 사용 가능한 모든 펜싱 에이전트를 나열합니다. 필터를 지정하면 이 명령은 필터와 일치하는 펜싱 에이전트만 표시합니다.

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

다음 명령을 실행하여 지정된 펜싱 에이전트의 옵션을 표시합니다.

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

*예를 들어 다음 명령은 **telnet/SSH**를 통해 **APC**의 펜스 에이전트 옵션을 표시합니다.*

```
# 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
```

주의

*메서드 옵션을 제공하는 차단 에이전트의 경우 **fence_sbd** 에이전트를 제외하고 주기 값이 지원되지 않으며 데이터 손상이 발생할 수 있으므로 지정하지 않아야 합니다. 그러나 **fence_sbd** 의 경우에도 메서드를 지정하지 않고 기본값을 사용해야 합니다.*

10.2. 펜스 장치 생성

*펜스 장치를 생성하는 명령의 형식은 다음과 같습니다. 사용 가능한 펜스 장치 생성 옵션 목록은 **pcs stonith -h** 디스플레이를 참조하십시오.*

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

다음 명령은 단일 노드에 대해 단일 펜싱 장치를 생성합니다.

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

일부 차단 장치는 단일 노드만 펜싱할 수 있지만 다른 장치는 여러 노드를 펜싱할 수 있습니다. 차단 장치를 생성할 때 지정하는 매개변수는 차단 장치가 지원하고 요구하는 사항에 따라 다릅니다.

- 일부 펜스 장치는 펜싱할 수 있는 노드를 자동으로 결정할 수 있습니다.
- 차단 장치를 생성할 때 **pcmk_host_list** 매개 변수를 사용하여 해당 펜싱 장치에서 제어하는 모든 시스템을 지정할 수 있습니다.
- 일부 펜스 장치에는 펜스 장치가 이해할 수 있는 사양과 호스트 이름을 매핑해야 합니다. 펜싱 장치를 생성할 때 **pcs mk_host_map** 매개변수를 사용하여 호스트 이름을 매핑할 수 있습니다.

pcmk_host_list 및 **pcmk_host_map** 매개변수에 대한 자세한 내용은 펜싱 장치의 일반 속성을 참조하십시오.

펜스 장치를 구성한 후 장치를 테스트하여 올바르게 작동하는지 확인해야 합니다. 펜스 장치를 테스트하는 방법에 대한 자세한 내용은 차단 장치 테스트를 참조하십시오.

10.3. 펜싱 장치의 일반 속성

펜싱 장치에는 설정할 수 있는 일반 속성은 물론 펜싱 동작을 결정하는 다양한 클러스터 속성이 있습니다.

모든 클러스터 노드는 펜스 리소스가 시작되거나 중지되었는지 여부에 관계없이 펜싱 장치를 사용하여 다른 클러스터 노드를 펜싱할 수 있습니다. 리소스가 시작되는지 여부에 관계없이 장치에 대한 반복 모니터만 제어되는지의 여부에는 다음과 같은 예외가 있습니다.

- **pcs stonith disable stonith_id**명령을 실행하여 펜싱 장치를 비활성화할 수 있습니다. 이렇게 하면 모든 노드가 해당 장치를 사용하지 못하게 됩니다.
- 특정 노드가 펜싱 장치를 사용하지 못하도록 하려면 **pcs** 제약 조건 위치 ...을 사용하여 펜싱 리소스에 대한 위치 제한 조건을 구성할 수 있습니다. 명령을 사용하지 마십시오.

- ***stonith-enabled=false*** *를 구성하면 펜싱이 모두 비활성화됩니다. 그러나* ***Red Hat****은 프로덕션 환경에 적합하지 않으므로 펜싱이 비활성화된 경우 클러스터를 지원하지 않습니다.*

다음 표에서는 펜싱 장치에 설정할 수 있는 일반 속성을 설명합니다.

표 10.1. 펜싱 장치의 일반 속성

필드	유형	기본값	설명
pcmk_host_map	string		호스트 이름을 지원하지 않는 장치의 호스트 이름과 포트 번호 매핑 예: **node1:1;node2:2,3** 은 클러스터에 node1에 포트 1과 node2에 대해 3을 사용하도록 지시합니다. **pcmk_host_map** 속성은 값 앞에 있는 백슬래시를 사용하여 **pcmk_host_map** 값 내부의 특수 문자를 지원합니다. 예를 들어 호스트 별칭에 공백을 포함하도록 **pcmk_host_map="node3:plug\ 1"** 을 지정할 수 있습니다.
pcmk_host_list	string		이 장치에서 제어되는 머신 목록 (선택사항(**pcmk_host_check=static-list**)이 아닌 경우 선택).
pcmk_host_check	string	* **static-list** if either **pcmk_host_list** or **pcmk_host_map** is set * 그렇지 않으면 fence 장치가 **목록** 작업을 지원하는 경우 **dynamic-list** * 그렇지 않으면 펜스 장치가 **상태** 작업을 지원하는 경우 **상태** * 다른 than one, **none**.	장치에서 제어하는 시스템을 결정하는 방법. 허용되는 값: **dynamic-list** (device 쿼리), **static-list** (**pcmk_host_list** 특성 확인), none(모든 장치를 펜싱할 수 있음)

다음 표에는 펜싱 장치에 대해 설정할 수 있는 추가 속성이 요약되어 있습니다. 이러한 속성은 고급 용도로만 사용됩니다.

표 10.2. 펜싱 장치의 고급 속성

필드	유형	기본값	설명
pcmk_host_argument	string	port	포트 대신 제공할 대체 매개 변수입니다. 일부 장치는 표준 포트 매개 변수를 지원하지 않거나 추가 포트를 제공할 수 있습니다. 이를 사용하여 펜싱할 시스템을 지정해야 하는 대체 장치별 매개 변수를 지정합니다. **none** 값은 클러스터에 추가 매개 변수를 제공하지 않도록 지시하는 데 사용할 수 없습니다.
pcmk_reboot_action	string	reboot	**재부팅** 하지 않고 실행할 대체 명령. 일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이를 사용하여 재부팅 작업을 구현하는 장치별 대체 명령을 지정합니다.
pcmk_reboot_timeout	time	60s	**stonith-timeout** 대신 재부팅 작업에 사용할 대체 시간 초과를 지정합니다. 일부 장치는 정상보다 완성하기 위해 훨씬 더 많은 시간이 필요합니다. 이를 사용하여 재부팅 작업에 대한 대체 장치별 시간 초과를 지정합니다.
pcmk_reboot_retries	integer	2	제한 시간 내에 **재부팅** 명령을 다시 시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 실패할 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. 중단하기 전에 Pacemaker에서 재부팅 작업을 재시도하는 횟수를 변경하려면 이 옵션을 사용합니다.
pcmk_off_action	string	Off	**꺼지** 기 대신 실행할 대체 명령. 일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이를 사용하여 off 작업을 구현하는 장치별 대체 명령을 지정합니다.
pcmk_off_timeout	time	60s	**stonith-timeout** 대신 끄기 작업에 사용할 대체 시간 초과를 지정합니다. 일부 장치는 정상보다 완료하기 위해 훨씬 더 많은 시간이 필요합니다. 이 옵션을 사용하여 off 작업에 대한 대체 장치별 시간 초과를 지정합니다.
pcmk_off_retries	integer	2	제한 시간 내에 off 명령을 다시 시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 실패할 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. Pacemaker에서 작업을 중지하기 전에 재시도 횟수를 변경하려면 이 옵션을 사용합니다.

필드	유형	기본값	설명
pcmk_list_action	string	list	**목록** 대신 실행할 대체 명령. 일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이를 사용하여 목록 작업을 구현하는 장치별 대체 명령을 지정합니다.
pcmk_list_timeout	time	60s	목록 작업에 사용할 대체 시간 제한을 지정합니다. 일부 장치는 정상보다 완료하기 위해 훨씬 더 많은 시간이 필요합니다. 이를 사용하여 목록 작업에 대한 대체 장치별 시간 초과를 지정합니다.
pcmk_list_retries	integer	2	제한 시간 내에 **list** 명령을 다시 시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 실패할 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. 중단하기 전에 Pacemaker 재시도 목록 작업의 횟수를 변경하려면 이 옵션을 사용합니다.
pcmk_monitor_action	string	모니터	**모니터** 대신 실행할 대체 명령. 일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이를 사용하여 모니터 작업을 구현하는 장치별 대체 명령을 지정합니다.
pcmk_monitor_timeout	time	60s	**stonith-timeout** 대신 모니터 작업에 사용할 대체 시간 초과를 지정합니다. 일부 장치는 정상보다 완료하기 위해 훨씬 더 많은 시간이 필요합니다. 이를 사용하여 모니터링 작업에 대한 대체 장치별 시간 초과를 지정합니다.
pcmk_monitor_retries	integer	2	제한 시간 내에 **monitor** 명령을 다시 시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 실패할 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. 중단하기 전에 Pacemaker에서 작업을 다시 시도하는 횟수를 변경하려면 이 옵션을 사용합니다.
pcmk_status_action	string	status	**상태** 대신 실행할 대체 명령. 일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이를 사용하여 상태 작업을 구현하는 장치별 대체 명령을 지정합니다.

필드	유형	기본값	설명
pcmk_status_timeout	time	60s	**stonith-timeout** 대신 상태 작업에 사용할 대체 시간 초과를 지정합니다. 일부 장치는 정상보다 완료하기 위해 훨씬 더 많은 시간이 필요합니다. 이를 사용하여 상태 작업에 대한 대체 장치별 시간 초과를 지정합니다.
pcmk_status_retries	integer	2	제한 시간 내에 status 명령을 재시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 실패할 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. 중단하기 전에 Pacemaker에서 상태 작업을 재시도하는 횟수를 변경하려면 이 옵션을 사용합니다.
pcmk_delay_base	string	0s	펜싱 작업에 대한 기본 지연을 활성화하고 기본 지연 값을 지정합니다. **pcmk_delay_base** 매개변수를 사용하여 다른 노드에 대해 다른 값을 지정할 수 있습니다. 펜싱 지연 매개 변수 및 상호 작용에 대한 일반적인 정보는 지연 을 참조하십시오.
pcmk_delay_max	time	0s	펜싱 작업에 대한 임의의 지연을 활성화하고 결합된 기본 지연 및 임의 지연의 최대 값인 최대 지연을 지정합니다. 예를 들어 기본 지연이 3이고 **pcmk_delay_max** 가 10이면 임의 지연은 3에서 10 사이입니다. 펜싱 지연 매개 변수 및 상호 작용에 대한 일반적인 정보는 지연 을 참조하십시오.
pcmk_action_limit	integer	1	이 장치에서 병렬로 수행할 수 있는 최대 작업 수입니다. 클러스터 속성 **concurrent-fencing=true** 를 먼저 구성해야 합니다(기본값). 값 -1은 무제한입니다.
pcmk_on_action	string	On	고급 용도의 경우 다음을 수행합니다. 의 대신 실행할 대체 명령. **일부 장치는 표준 명령을 지원하지 않거나 추가 명령을 제공할 수 있습니다. 이 명령을 사용하여 작업에 대해 구현하는 장치별 대체 명령을 지정합니다.**

필드	유형	기본값	설명
pcmk_on_timeout	time	60s	고급 용도의 경우 다음을 수행합니다. **stonith-timeout** 대신 **작업에** 사용할 대체 시간 초과를 지정합니다. 일부 장치는 정상보다 완료하기 위해 훨씬 더 많은 시간이 필요합니다. 이를 사용하여 작업에 **대한** 대체 장치별 시간 초과를 지정합니다.
pcmk_on_retries	integer	2	고급 용도의 경우 다음을 수행합니다. 제한 시간 내에 **on** 명령을 다시 시도하는 최대 횟수입니다. 일부 장치는 여러 연결을 지원하지 않습니다. 장치가 다른 작업으로 인해 사용 중인 경우 작업이 **실패할** 수 있으므로 Pacemaker에서 작업이 남아 있으면 작업을 자동으로 다시 시도합니다. 중단하기 전에 Pacemaker **에서** 작업을 다시 시도하는 횟수를 변경하려면 이 옵션을 사용합니다.

개별 펜스 장치에 대해 설정할 수 있는 속성 외에도 다음 표에 설명된 대로 펜싱 동작을 결정하는 클러스터 속성도 설정할 수 있습니다.

*표 **10.3. Fencing Behavior**를 결정하는 클러스터 속성*

옵션	기본값	설명
stonith-enabled	true	오류가 발생한 노드와 중지할 수 없는 리소스를 펜싱해야 함을 나타냅니다. 데이터를 보호하려면 이 **true** 를 설정해야 합니다. **true** 또는 설정되지 않은 경우 하나 이상의 STONITH 리소스도 구성되지 않는 한 클러스터에서 리소스 시작을 거부합니다. Red Hat은 이 값이 **true** 로 설정된 클러스터만 지원합니다.
stonith-action	reboot	펜싱 장치에 전달할 작업입니다. 허용되는 값: **재부팅**,**off**. 값 **poweroff** 도 허용되지만 레거시 장치에만 사용됩니다.
stonith-timeout	60s	STONITH 작업이 완료될 때까지 대기하는 시간입니다.
stonith-max-attempts	10	클러스터가 더 이상 즉시 다시 수행되지 않기 전에 대상에 대해 펜싱에 실패할 수 있는 횟수는 몇 번입니까.

옵션	기본값	설명
stonith-watchdog-timeout		노드가 하드웨어 워치독에 의해 종료된 것으로 간주될 때까지 대기하는 최대 시간입니다. 이 값을 하드웨어 워치독 시간 초과 값의 두 배로 설정하는 것이 좋습니다. 이 옵션은 펜싱에 watchdog 전용 SBD 구성을 사용하는 경우에만 필요합니다.
concurrent-fencing	true	펜싱 작업을 병렬로 수행할 수 있습니다.
fence-reaction	중지	자체 펜싱을 알리는 경우 클러스터 노드에서 대응하는 방법을 결정합니다. 펜싱이 잘못 구성된 경우 클러스터 노드는 자체 펜싱 알림을 수신하거나 패브릭 펜싱을 사용하여 클러스터 통신을 손상시키지 않는 경우 해당 펜싱 알림을 수신할 수 있습니다. 허용되는 값은 Pacemaker를 즉시 중지하고 중지되거나 **패닉** 이 로컬 노드를 즉시 재부팅하여 실패 시 중지되도록 시도합니다. **이 속성의 기본값은 stop 이지만 이 값에 가장 안전한 옵션은 패닉 이며 로컬 노드를 즉시 재부팅하려고 시도합니다. 패브릭 펜싱과 함께 가장 적합한 경우와 마찬가지로 중지 동작을 선호하는 경우 이를 명시적으로 설정하는 것이 좋습니다.**
priority-fencing-delay	0 (비활성화)	분할에서 가장 적은 리소스가 실행 중인 노드가 펜싱되는 노드가 펜싱되도록 2 노드 클러스터를 구성할 수 있는 펜싱 지연을 설정합니다. 펜싱 지연 매개 변수 및 상호 작용에 대한 일반적인 정보는 지연 을 참조하십시오.

클러스터 속성 설정에 대한 자세한 내용은 클러스터 속성 설정 및 제거를 참조하십시오.

10.4. 펜싱 지연

2-노드 클러스터에서 클러스터 통신이 손실되면 한 노드에서 먼저 이를 감지하고 다른 노드를 펜싱할 수 있습니다. 그러나 두 노드 모두 동시에 이를 감지하면 각 노드가 다른 노드의 펜싱을 시작하여 두 노드의 전원이 꺼지거나 재설정될 수 있습니다. 펜싱 지연을 설정하면 두 클러스터 노드가 서로 펜싱될 가능성을 줄일 수 있습니다. 두 개 이상의 노드가 있는 클러스터에서 지연을 설정할 수 있지만 쿼럼이 있는 파티션만 펜싱을 시작하기 때문에 일반적으로는 이점이 없습니다.

시스템 요구 사항에 따라 다양한 유형의 펜싱 지연을 설정할 수 있습니다.

- *정적 펜싱 지연*

  *정적 펜싱 지연은 미리 정의된 고정된 지연입니다. 한 노드에 정적 지연을 설정하면 노드가 펜싱될 가능성이 높아집니다. 이로 인해 다른 노드가 통신 손실된 통신을 탐지한 후 먼저 펜싱을 시작할 가능성이 높아집니다. 활성/수동 클러스터에서 패시브 노드에서 지연을 설정하면 통신이 중단될 때 패시브 노드가 펜싱될 가능성이 높아집니다. **pcs_delay_base** 클러스터 속성을 사용하여 정적 지연을 구성합니다. 각 노드에 별도의 차단 장치를 사용하거나 모든 노드에 단일 차단 장치를 사용하는 경우 이 속성을 설정할 수 있습니다.*

- *동적 펜싱 지연*

  *동적 펜싱 지연은 임의적입니다. 이는 다를 수 있으며 펜싱 시에 결정됩니다. 임의의 지연을 구성하고 **pcs_delay_max** 클러스터 속성을 사용하여 결합된 기본 지연 및 임의의 지연을 지정합니다. 각 노드의 펜싱 지연이 임의적인 경우 펜싱되는 노드도 임의적입니다. 이 기능은 클러스터가 활성/활성 설계의 모든 노드에 대해 단일 차단 장치로 구성된 경우 유용할 수 있습니다.*

- *우선 순위 펜싱 지연*

  *우선 순위 펜싱 지연은 활성 리소스 우선 순위를 기반으로 합니다. 모든 리소스의 우선 순위가 동일한 경우 가장 적은 리소스가 실행되는 노드는 펜싱되는 노드입니다. 대부분의 경우 하나의 지연 관련 매개변수만 사용하지만 결합할 수 있습니다. 지연 관련 매개 변수를 결합하면 리소스의 우선 순위 값이 함께 추가되어 총 지연이 생성됩니다. **priority-fencing-delay** 클러스터 속성을 사용하여 우선순위 펜싱 지연을 구성합니다. 이 기능은 노드 간 통신이 손실될 때 가장 적은 리소스를 실행하는 노드가 차단될 수 있으므로 활성/활성 클러스터 설계에서 유용할 수 있습니다.*

***pcmk_delay_base** 클러스터 속성*

***pcmk_delay_base** 클러스터 속성을 설정하면 펜싱의 기본 지연이 활성화되고 기본 지연 값을 지정합니다.*

***pcmk_delay_max** 클러스터 속성을 **pcmk_delay_base** 속성 외에도 설정할 때 전체 지연은 이 정적 지연에 추가된 임의의 지연 값에서 파생되어 합계가 최대 지연 아래로 유지됩니다. **pcmk_delay_base** 를*

설정했지만 **pcmk_delay_max** 를 설정하지 않으면 지연에 임의의 구성 요소가 없으며 **pcmk_delay_base** 의 값이 됩니다.

pcmk_delay_base 매개변수를 사용하여 다른 노드에 대해 다른 값을 지정할 수 있습니다. 이를 통해 노드마다 다른 지연과 함께 **2-**노드 클러스터에서 단일 펜스 장치를 사용할 수 있습니다. 별도의 지연을 사용하도록 두 개의 개별 장치를 구성할 필요가 없습니다. 다른 노드의 다른 값을 지정하려면 **pcmk_host_map** 과 유사한 구문을 사용하여 호스트 이름을 해당 노드의 지연 값에 매핑합니다. 예를 들어 **node1:0;node2:10s** 는 **node1** 을 펜싱할 때 **no delay**를 사용하고, **node2** 를 펜싱할 때 **10**초 지연을 사용합니다.

pcmk_delay_max 클러스터 속성

pcmk_delay_max 클러스터 속성을 설정하면 펜싱 작업에 대한 임의의 지연을 활성화하고 결합된 기본 지연 및 임의 지연의 최대 값인 최대 지연 시간을 지정합니다. 예를 들어 기본 지연이 **3**이고 **pcmk_delay_max** 가 **10**이면 임의 지연은 **3**에서 **10** 사이입니다.

pcmk_delay_base 클러스터 속성을 **pcmk_delay_max** 속성 외에도 설정할 때 전체 지연은 이 정적 지연에 추가된 임의의 지연 값에서 파생되어 합계가 최대 지연 아래로 유지됩니다. **pcmk_delay_max** 를 설정했지만 **pcmk_delay_base** 를 설정하지 않으면 지연에 대한 정적 구성 요소가 없습니다.

priority-fencing-delay 클러스터 속성

priority-fencing-delay 클러스터 속성을 설정하면 분할에서 가장 적은 리소스가 실행되는 노드가 펜싱되는 노드가 되도록 **2-**노드 클러스터를 구성할 수 있습니다.

priority-fencing-delay 속성은 시간 기간으로 설정할 수 있습니다. 이 속성의 기본값은 **0**(비활성화)입니다. 이 속성이 **0**이 아닌 값으로 설정되고 우선 순위 **meta-attribute**가 하나 이상의 리소스에 대해 구성된 경우 분할에서 실행 중인 모든 리소스의 우선 순위가 가장 높은 노드가 작동 상태를 유지할 가능성이 더 높습니다. 예를 들어 **pcs resource defaults update priority=1** 및 **pcs** 속성이 **priority-fencing-delay=15s** 를 설정하고 다른 우선순위가 설정되지 않은 경우 다른 노드가 펜싱을 시작하기 전에 **15**초를 기다리므로 대부분의 리소스를 실행하는 노드가 계속 작동할 가능성이 더 높습니다. 특정 리소스가 나머지 리소스보다 더 중요한 경우 우선순위를 높일 수 있습니다.

해당 복제본에 대해 우선 순위가 구성된 경우 승격 가능한 복제본의 승격된 역할을 실행 중인 노드에 추가 **1** 지점이 부여됩니다.

펜싱 지연의 상호 작용

펜싱 지연 유형을 두 개 이상 설정하면 다음과 같은 결과가 발생합니다.

- **priority-fencing-delay** 속성으로 설정된 지연은 **pcmk_delay_base** 및 **pcmk_delay_max** 차단 장치 속성의 지연에 추가됩니다. 이 동작은 **on-fail=fencing** 이 리소스 모니터 작업에 설정된

*경우와 같이 노드 손실 이외의 이유로 두 노드를 모두 펜싱해야 하는 경우 두 노드를 모두 지연할 수 있습니다. 이러한 지연을 조합하여 설정할 때 **priority-fencing-delay** 속성을 **pcmk_delay_base** 및 **pcmk_delay_max** 의 최대 지연보다 훨씬 큰 값으로 설정합니다. 이 속성을 두 번으로 설정하면 이 값은 항상 안전합니다.*

- *__Pacemaker__ 자체에서 예약한 펜싱만 펜싱 지연을 관찰합니다. **dlm_controld** 및 **pcs stonith fence** 명령으로 구현된 펜싱과 같은 외부 코드에서 예약한 펜싱은 펜싱 장치에 필요한 정보를 제공하지 않습니다.*

- *일부 개별 차단 에이전트는 에이전트에 의해 결정되며 **pcmk_delay_*** 속성으로 구성된 지연과 관계없이 지연 매개 변수를 구현합니다. 이러한 두 지연이 모두 구성된 경우 함께 추가되며 일반적으로 함께 사용되지 않습니다.*

10.5. 펜스 장치 테스트

*펜싱은 **Red Hat Cluster** 인프라의 기본 구성 요소이며 펜싱이 제대로 작동하는지 검증하거나 테스트하는 것이 중요합니다.*

절차

다음 절차에 따라 펜스 장치를 테스트합니다.

1. *__ssh, telnet, HTTP__ 또는 장치에 연결하여 수동으로 로그인하여 펜스 장치를 테스트하거나 제공된 출력을 확인하는 데 사용되는 원격 프로토콜을 사용합니다. 예를 들어 **IPMI** 사용 장치에 대한 펜싱을 구성하는 경우 **ipmitool** 을 사용하여 원격으로 로그인합니다. 펜싱 에이전트를 사용할 때 해당 옵션이 필요할 수 있으므로 수동으로 로그인할 때 사용되는 옵션을 기록해 두십시오.*

   *펜스 장치에 로그인할 수 없는 경우 장치를 **ping**할 수 있는지, 펜스 장치에 대한 액세스를 방지하는 방화벽 구성, 펜싱 장치에서 원격 액세스가 활성화되고 인증 정보가 올바른지 확인합니다.*

2. *펜스 에이전트 스크립트를 사용하여 수동으로 펜스 에이전트를 실행합니다. 이를 위해 클러스터에 장치를 구성하기 전에 클러스터 서비스가 실행 중일 필요는 없으므로 이 단계를 수행할 수 있습니다. 이렇게 하면 계속하기 전에 펜스 장치가 올바르게 응답하도록 할 수 있습니다.*

참고

*이 예에서는 **iLO** 장치에 **fence_ip extensionan** 펜스 에이전트 스크립트를 사용합니다. 사용할 실제 펜스 에이전트와 에이전트를 호출하는 명령은 서버 하드웨어에 따라 다릅니다. 지정한 옵션을 결정하기 위해 사용 중인 펜스 에이전트의 도움말 페이지를 참조해야 합니다. 일반적으로 펜스 장치의 로그인 및 암호 및 펜스 장치와 관련된 기타 정보를 알아야 합니다.*

*다음 예제에서는 **-o status** 매개 변수를 사용하여 **fence_ipmilan fence** 에이전트 스크립트를 실행하여 실제로 펜싱하지 않고 다른 노드의 펜스 장치 인터페이스의 상태를 확인하는 데 사용하는 형식을 보여줍니다. 이를 통해 노드를 재부팅하기 전에 장치를 테스트하고 작동하게 할 수 있습니다. 이 명령을 실행하는 경우 **iLO** 장치의 전원을 켜고 끄는 **iLO** 사용자의 이름과 암호를 지정합니다.*

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

*다음 예제에서는 **-o reboot** 매개변수로 **fence_ipmilan fence** 에이전트 스크립트를 실행하는 데 사용하는 형식을 보여줍니다. 한 노드에서 이 명령을 실행하면 이 **iLO** 장치에서 관리하는 노드가 재부팅됩니다.*

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

*펜스 에이전트에서 상태, 오프, 켜기 또는 재부팅 작업을 제대로 수행하지 못하는 경우 하드웨어, 펜스 장치의 구성, 명령의 구문을 확인해야 합니다. 또한 디버그 출력이 활성화된 **fence** 에이전트 스크립트를 실행할 수 있습니다. 디버그 출력은 일부 펜싱 에이전트에서 펜스 장치에 로그인할 때 펜싱 에이전트 스크립트가 실패하는 이벤트의 위치를 확인하는 데 유용합니다.*

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

발생한 오류를 진단할 때는 펜스 장치에 수동으로 로그인할 때 지정한 옵션이 펜스 에이전트 스크립트를 사용하여 펜스 에이전트에 전달한 옵션과 동일한지 확인해야 합니다.

*암호화된 연결을 지원하는 펜스 에이전트의 경우 인증서 검증 실패로 인해 오류가 표시될 수 있습니다. 호스트를 신뢰하거나 **fence** 에이전트의 **ssl-insecure** 매개변수를 사용해야 합니다. 마찬가지로 대상 장치에서 **SSL/TLS**가 비활성화된 경우, 펜스 에이전트에 대한 **SSL** 매개 변수를 설정할 때 이 문제를 고려해야 할 수 있습니다.*

참고

*테스트 중인 펜스 에이전트가 **fence_drac,fence_ilo** 또는 계속 실패하는 시스템 관리 장치에 대한 기타 펜싱 에이전트인 경우 **fence_ipmilan** 으로 돌아갑니다. 대부분의 시스템 관리 카드에서는 **IPMI** 원격 로그인을 지원하고 지원되는 유일한 펜싱 에이전트는 **fence_ipmilan** 입니다.*

3.

*다음 예와 같이 수동으로 작동하고 클러스터가 작동하는 동일한 옵션을 사용하여 클러스터에 펜스 장치를 구성한 후 모든 노드의 **pcs stonith fence** 명령(또는 여러 번)을 사용하여 다음 예와 같이 테스트합니다. **pcs stonith fence** 명령은 **CIB**에서 클러스터 구성을 읽고 펜스 에이전트를 구성된 대로 호출하여 펜스 작업을 실행합니다. 이렇게 하면 클러스터 구성이 올바른지 확인합니다.*

```
# pcs stonith fence node_name
```

pcs stonith fence 명령이 제대로 작동하는 경우 펜스 이벤트가 발생할 때 클러스터의 펜싱 구성이 작동해야 합니다. 명령이 실패하면 클러스터 관리에서 검색된 구성을 통해 펜스 장치를 호출할 수 없습니다. 다음 문제를 확인하고 필요에 따라 클러스터 구성을 업데이트합니다.

- *펜스 구성을 확인합니다. 예를 들어 호스트 맵을 사용하는 경우 시스템에서 제공한 호스트 이름을 사용하여 노드를 찾을 수 있는지 확인해야 합니다.*

- *장치의 암호 및 사용자 이름에 **bash** 쉘에서 잘못 해석될 수 있는 특수 문자가 포함되어 있는지 확인합니다. 따옴표로 묶은 암호와 사용자 이름을 입력하면 이 문제를 해결할 수 있습니다.*

- ***pcs stonith** 명령에 지정한 정확한 **IP** 주소 또는 호스트 이름을 사용하여 장치에 연결할 수 있는지 확인합니다. 예를 들어 **stonith** 명령에서 호스트 이름을 지정하고 **IP** 주소를 사용하여 테스트한 경우 이는 유효한 테스트가 아닙니다.*

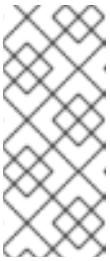
- *펜스 장치가 사용하는 프로토콜에 액세스할 수 있는 경우 해당 프로토콜을 사용하여 장치에 연결을 시도합니다. 예를 들어, 많은 에이전트가 **ssh** 또는 **telnet**을 사용합니다. 장치를 구성할 때 제공한 인증 정보를 사용하여 장치에 연결하여 유효한 프롬프트가 있는지 확인하고 장치에 로그인할 수 있는지 확인해야 합니다.*

*모든 매개변수가 적절하지만 여전히 펜스 장치에 연결하는 데 어려움이 있는 경우, 펜스 장치 자체에서 로깅을 확인할 수 있습니다. 장치가 이를 제공하면 사용자가 연결했는지와 사용자가 발급한 명령이 있는지 확인할 수 있습니다. **/var/log/ question** 파일에서 **stonith** 및 **error**의 인스턴스를 검색할 수도 있습니다. 이는 전송에 대한 아이디어를 줄 수 있지만 일부 에이전트는 추가 정보를 제공할 수 있습니다.*

4.

펜스 장치 테스트가 작동하고 클러스터가 가동되어 실행되면 실제 오류를 테스트합니다. 이렇게 하려면 토큰 손실을 시작해야 하는 클러스터에서 작업을 수행합니다.

- 네트워크를 중단하십시오. 네트워크를 사용하는 방법은 특정 구성에 따라 다릅니다. 대부분의 경우 물리적으로 네트워크를 끌어오거나 호스트에서 전원을 옮길 수 있습니다. 네트워크 오류 시뮬레이션에 대한 자세한 내용은 **RHEL** 클러스터에서 네트워크 오류를 시뮬레이션하는 적절한 방법은 무엇입니까?를 참조하십시오.



참고

네트워크 또는 전원 케이블을 물리적으로 분리하지 않고 로컬 호스트에서 네트워크 인터페이스를 비활성화하는 것은 일반적인 실제 오류를 정확하게 시뮬레이션하지 않기 때문에 펜싱 테스트로 권장되지 않습니다.

- 로컬 방화벽을 사용하여 인바운드 및 아웃바운드로 **corosync** 트래픽을 차단합니다.

다음 예제에서는 **corosync**가 기본 **corosync** 포트를 사용한다고 가정하고, **firewalld**가 로컬 방화벽으로 사용되고 **corosync**에 사용되는 네트워크 인터페이스는 기본 방화벽 영역에 있습니다.

```
# 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**를 사용하여 크래시 및 패닉을 시뮬레이션합니다. 그러나 커널 패닉을 트리거하는 경우 데이터 손실이 발생할 수 있습니다. 클러스터 리소스를 먼저 비활성화하는 것이 좋습니다.

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

10.6. 펜싱 수준 구성

Pacemaker는 펜싱 토폴로지라는 기능을 통해 여러 장치로 노드를 펜싱할 수 있습니다. 토폴로지를 구현하려면 일반적으로 개별 장치를 생성한 다음 구성의 펜싱 토폴로지 섹션에서 하나 이상의 펜싱 수준을 정의합니다.

Pacemaker는 펜싱 수준을 다음과 같이 처리합니다.

- 각 수준은 1부터 시작하여 오름차순으로 시도됩니다.
- 장치가 실패하면 처리가 현재 수준에 대해 종료됩니다. 해당 수준의 장치가 더 이상 실행되지 않으며 대신 다음 수준을 시도합니다.
- 모든 장치가 성공적으로 펜싱되면 해당 수준은 성공했으며 다른 수준은 시도하지 않습니다.
- 수준이 전달되거나 모든 수준을 시도(실패)하면 작업이 완료됩니다.

다음 명령을 사용하여 노드에 펜싱 수준을 추가합니다. 장치는 해당 수준에서 노드에 대해 시도되는 쉼표로 구분된 **stonith ids** 목록으로 제공됩니다.

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

다음 명령은 현재 구성된 모든 펜싱 수준을 나열합니다.

```
pcs stonith level
```

다음 예제에서는 노드 **rh7-2** 에 대해 구성된 두 개의 펜스 장치: **my_ilo** 라는 **ilo fence** 장치 및 **my_apc** 라는 **apc fence** 장치입니다. 이러한 명령은 펜스 수준을 설정하므로 **my_ilo** 장치가 실패하고 노드를 펜싱할 수 없는 경우 **Pacemaker**에서 **my_apc** 장치를 사용하려고 시도합니다. 이 예에서는 수준을 구성한 후 **pcs stonith level** 명령의 출력도 보여줍니다.

```
# 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
```

다음 명령은 지정된 노드 및 장치의 펜스 수준을 제거합니다. 노드 또는 장치가 지정되지 않은 경우 지정한 펜스 수준이 모든 노드에서 제거됩니다.

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

다음 명령은 지정된 노드 또는 **stonith ID**의 펜스 수준을 지웁니다. 노드 또는 **stonith ID**를 지정하지 않으면 모든 펜스 수준이 삭제됩니다.

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

두 개 이상의 **stonith ID**를 지정하는 경우 다음 예제와 같이 공백을 쉼표로 구분해야 합니다.

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

다음 명령은 펜스 수준에 지정된 모든 펜스 장치와 노드가 있는지 확인합니다.

```
pcs stonith level verify
```

노드 이름 및 노드 특성 및 해당 값에 의해 적용되는 정규식으로 펜싱 토폴로지에서 노드를 지정할 수 있습니다. 예를 들어 다음 명령은 **node1,node2** 및 **node3** 을 구성하여 펜싱 장치 **apc1** 및 **apc2,** 노드 **4 , node4, node 6** 을 사용하여 펜싱 장치 **apc3** 및 **apc4** 를 사용합니다.

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

다음 명령은 노드 속성 일치를 사용하여 동일한 결과를 제공합니다.

```
# 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. 중복 전원 공급 장치에 대한 펜싱 구성

중복 전원 공급 장치에 대한 펜싱을 구성할 때 클러스터는 전원 공급 장치를 다시 켜기 전에 호스트를 재부팅하려고 할 때 두 전원 공급 장치를 모두 꺼야 합니다.

노드의 전원이 완전히 손실되지 않으면 노드가 리소스를 해제하지 않을 수 있습니다. 이렇게 하면 이러한 리소스에 동시에 액세스할 수 있고 손상될 수 있습니다.

다음 예와 같이 각 장치를 한 번만 정의하고 두 장치를 모두 펜싱하는 데 모두 지정해야 합니다.

```
# 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. 구성된 펜스 장치 표시

다음 명령은 현재 구성된 모든 펜스 장치를 보여줍니다. **stonith_id** 가 지정된 경우 명령은 구성된 펜싱 장치에 대한 옵션만 표시합니다. **--full** 옵션을 지정하면 구성된 모든 펜싱 옵션이 표시됩니다.

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

10.9. PCS 명령으로 차단 장치 내보내기

Red Hat Enterprise Linux 9.1에서는 **pcs stonith config** 명령의 **--output-format=cmd** 옵션을 사용하여 다른 시스템에서 구성된 펜싱 장치를 다시 만드는 데 사용할 수 있는 **pcs** 명령을 표시할 수 있습니다.

다음 명령은 **fence_apc_snmp** 차단 장치를 생성하고 장치를 다시 만드는 데 사용할 수 있는 **pcs** 명령을 표시합니다.

```
# 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. 펜스 장치 수정 및 삭제

다음 명령을 사용하여 현재 구성된 펜싱 장치에 옵션을 수정하거나 추가합니다.

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

pcs stonith update 명령을 사용하여 **SCSI** 펜싱 장치를 업데이트하면 펜싱 리소스가 실행 중인 동일한 노드에서 실행되는 모든 리소스가 다시 시작됩니다. 다음 명령의 버전을 사용하여 다른 클러스터 리소

스를 재시작하지 않고 **SCSI** 장치를 업데이트할 수 있습니다. **RHEL 9.1**에서는 **SCSI** 펜싱 장치를 다중 경로 장치로 구성할 수 있습니다.

```
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
```

다음 명령을 사용하여 현재 구성에서 펜싱 장치를 제거합니다.

```
pcs stonith delete stonith_id
```

10.11. 클러스터 노드 수동 펜싱

다음 명령을 사용하여 노드를 수동으로 펜싱할 수 있습니다. **--off** 를 지정하면 **stonith**에 대한 **off API** 호출을 사용하므로 재부팅하지 않고 노드가 꺼집니다.

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

펜스 장치가 더 이상 활성 상태가 아닌 경우에도 노드를 펜싱할 수 없는 경우 클러스터에서 노드의 리소스를 복구하지 못할 수 있습니다. 이 경우 노드의 전원이 꺼졌는지 수동으로 확인한 후 다음 명령을 입력하여 노드의 전원이 꺼지고 복구할 수 있는지 클러스터를 확인할 수 있습니다.

주의

지정하는 노드가 실제로 꺼져 있지 않지만 일반적으로 클러스터에서 제어하는 클러스터 소프트웨어 또는 서비스를 실행하면 데이터 손상/클러스터 오류가 발생합니다.

```
pcs stonith confirm node
```

10.12. 펜스 장치 비활성화

펜싱 장치/리소스를 비활성화하려면 **pcs stonith disable** 명령을 실행합니다.

다음 명령은 펜스 장치 **myapc** 를 비활성화합니다.

```
# pcs stonith disable myapc
```

10.13. 펜싱 장치를 사용하여 노드에서 노드 방지

특정 노드에서 펜싱 장치를 사용하지 못하도록 하려면 펜싱 리소스에 대한 위치 제한 조건을 구성할 수 있습니다.

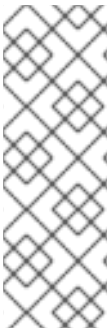
다음 예제에서는 **fence** 장치 **node1-ipmi** 가 **node1** 에서 실행되지 않도록 합니다.

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

10.14. 통합 펜스 장치와 함께 사용하도록 ACPI 구성

클러스터에서 통합 펜스 장치를 사용하는 경우 즉시 펜싱을 완료하도록 **ACPI**(고급 구성 및 전원 인터페이스)를 구성해야 합니다.

통합 펜스 장치로 클러스터 노드를 펜싱하도록 구성된 경우 해당 노드에 대해 **ACPI soft-Off**를 비활성화합니다. **ACPI soft-Off**를 비활성화하면 통합 펜스 장치가 클린 종료(예: **shutdown -h**)가 아닌 즉시 노드를 완전히 중단할 수 있습니다. 그렇지 않으면 **ACPI soft-Off**가 활성화된 경우 통합된 펜스 장치는 노드를 끄는 데 **4**초 이상 걸릴 수 있습니다(다음 참고 사항 참조). 또한 **ACPI soft-Off**가 활성화되어 종료되는 동안 노드가 패닉 또는 정지되면 통합된 펜스 장치가 노드를 해제하지 못할 수 있습니다. 이러한 상황에서는 펜싱이 지연되거나 실패했습니다. 결과적으로 통합 펜스 장치로 노드를 펜싱하고 **ACPI soft-Off**가 활성화되면 클러스터가 느리게 복구되거나 복구에 대한 관리 개입이 필요합니다.



참고

노드를 펜싱하는 데 필요한 시간은 사용된 통합 펜스 장치에 따라 다릅니다. 일부 통합된 펜스 장치는 전원 버튼을 눌러 유지하는 것과 동일한 작업을 수행합니다. 따라서 펜스 장치는 **4~5**초 내에 노드를 끕니다. 다른 통합된 펜스 장치는 일시적으로 전원 버튼을 눌러서 노드를 끄도록 운영 체제에 의존합니다. 따라서 펜스 장치는 **4~5**초 이상 기간 내에 노드를 끕니다.

- **ACPI soft-Off**를 비활성화하는 기본 방법은 아래의 **"instant-off"** 또는 **"instant-off"**로 노드를 끄는 것과 동등한 설정을 변경하는 것입니다.

BIOS를 사용하여 **ACPI soft-Off**를 비활성화하는 것은 일부 시스템에서는 가능하지 않을 수 있습니다. **BIOS**를 사용하여 **ACPI soft-Off**를 비활성화하는 것이 클러스터에 적합하지 않은 경우 다음 대체 방법 중 하나를 사용하여 **ACPI soft-Off**를 비활성화할 수 있습니다.

- */etc/systemd/logind.conf* 파일에서 ***HandlePowerKey=ignore*** 를 설정하고 ***logind.conf*** 파일의 ***"Disabling ACPI-Off in the logind.conf*** 파일에 설명된 대로 펜싱 시 노드 노드가 즉시 꺼져 있는지 확인합니다. ***ACPI soft-Off***를 비활성화하는 첫 번째 대체 방법입니다.

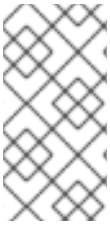
- 아래에 ***"GRUB 2*** 파일 비활성화"에 설명된 대로 커널 부팅 명령줄에 ***acpi=off*** 를 추가합니다. 이는 ***preferred*** 또는 첫 번째 대체 방법을 사용할 수 없는 경우 ***ACPI soft-Off***를 비활성화하는 두 번째 대체 방법입니다.

중요

이 방법은 ***ACPI***를 완전히 비활성화합니다. ***ACPI***가 완전히 비활성화되면 일부 컴퓨터가 올바르게 부팅되지 않습니다. 다른 방법이 클러스터에 적합하지 않은 경우에만 이 방법을 사용하십시오.

*10.14.1. BIOS*를 사용하여 *ACPI soft-Off* 비활성화

다음 절차에 따라 각 클러스터 노드의 ***BIOS***를 구성하여 ***ACPI soft-Off***를 비활성화할 수 있습니다.



참고

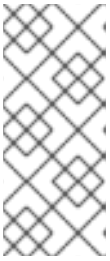
BIOS를 사용하여 ***ACPI soft-Off***를 비활성화하는 절차는 서버 시스템마다 다를 수 있습니다. 하드웨어 문서에서 이 절차를 확인해야 합니다.

절차

1. 노드를 재부팅하고 ***BIOS CMOS*** 설치 유틸리티 프로그램을 시작합니다.

2. ***Power*** 메뉴(또는 동등한 전원 관리 메뉴)로 이동합니다.

3. ***Power*** 메뉴에서 ***PWR-BTTN*** 함수(또는 이에 상응하는)를 ***Instant-Off*** 로 설정합니다(또는 지연 없이 전원 버튼을 통해 노드를 끄는 것과 동등한 설정). 아래의 ***BIOS CMOS*** 설정 ***octetsiy*** 예제에서는 ***PWR-BTTN***이 ***Instant-Off*** 로 설정된 ***ACPI*** 함수 를 활성화 및 ***soft-Off*** 로 설정한 전원 메뉴를 보여줍니다.



참고

ACPI 함수와 동등한PWR-BTTN 및 Instant- Off 는 컴퓨터마다 다를 수 있습니다. 그러나 이 절차의 목표는 컴퓨터가 지연 없이 전원 버튼을 통해 꺼지도록 BIOS를 구성하는 것입니다.

4. BIOS CMOS 설정 유틸리티 프로그램을 종료하고 BIOS 설정을 저장합니다.

5. 펜싱할 때 노드가 즉시 꺼져 있는지 확인합니다. 펜스 장치를 테스트하는 방법에 대한 자세한 내용은 차단 장치 테스트를 참조하십시오.

BIOS CMOS 설정 유틸리티:

```
`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         |                  |
|                                             |                  |
|                                             |                  |
+---------------------------------------------|------------------+
```

이 예에서는 Enabled 로 설정된 ACPI Function 및 PWR-BTTN의 soft-Off 를 Instant-Off 로 설정합니다.

10.14.2. logind.conf 파일에서 ACPI soft-Off 비활성화

/etc/systemd/logind.conf 파일에서 power-key 핸드링을 비활성화하려면 다음 절차를 사용하십시오.

절차

1. /etc/systemd/logind.conf 파일에 다음 구성을 정의합니다.

```
HandlePowerKey=ignore
```

2. systemd-logind 서비스를 다시 시작하십시오.

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

3. 펜싱할 때 노드가 즉시 꺼져 있는지 확인합니다. 펜스 장치를 테스트하는 방법에 대한 자세한 내용은 차단 장치 테스트를 참조하십시오.

10.14.3. GRUB 2 파일에서 ACPI를 완전히 비활성화

커널의 GRUB 메뉴 항목에 acpi=off 를 추가하여 ACPI soft-Off를 비활성화할 수 있습니다.

중요

이 방법은 ACPI를 완전히 비활성화합니다. ACPI가 완전히 비활성화되면 일부 컴퓨터가 올바르게 부팅되지 않습니다. 다른 방법이 클러스터에 적합하지 않은 경우에만 이 방법을 사용하십시오.

절차

GRUB 2 파일에서 ACPI를 비활성화하려면 다음 절차를 사용하십시오.

1. grubby 툴의 --update-kernel 옵션과 함께 --args 옵션을 사용하여 다음과 같이 각 클러스터 노드의 grub.cfg 파일을 변경합니다.

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

2. *노드를 재부팅합니다.*

3. *펜싱할 때 노드가 즉시 꺼져 있는지 확인합니다. 펜스 장치를 테스트하는 방법에 대한 자세한 내용은 차단 장치 테스트를 참조하십시오.*

11장. 클러스터 리소스 구성

다음 명령을 사용하여 클러스터 리소스를 생성하고 삭제합니다.

클러스터 리소스를 생성하는 명령 형식은 다음과 같습니다.

```
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]]
```

주요 클러스터 리소스 생성 옵션에는 다음이 포함됩니다.

- **--before** 및 **--after** 옵션은 리소스 그룹에 이미 존재하는 리소스와 관련하여 추가된 리소스의 위치를 지정합니다.
- **--disabled** 옵션을 지정하면 리소스가 자동으로 시작되지 않음을 나타냅니다.

클러스터에서 생성할 수 있는 리소스 수에는 제한이 없습니다.

해당 리소스에 대한 제약 조건을 구성하여 클러스터의 리소스 동작을 확인할 수 있습니다.

리소스 생성 예

다음 명령은 표준 **ocf**, 공급자 하트비트, **IPaddr2** 유형의 이름 **VirtualIP** 를 사용하여 리소스를 생성합니다. 이 리소스의 유동 주소는 **192.168.0.120**이며 시스템은 **30**초마다 리소스가 실행되고 있는지 확인합니다.

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

또는 **standard** 및 **provider** 필드를 생략하고 다음 명령을 사용할 수 있습니다. 이는 기본적으로 **ocf** 표준과 하트비트 공급자입니다.

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

구성된 리소스 삭제

다음 명령을 사용하여 구성된 리소스를 삭제합니다.

```
pcs resource delete resource_id
```

예를 들어 다음 명령은 리소스 ID가 VirtualIP 인 기존 리소스를 삭제합니다.

```
# pcs resource delete VirtualIP
```

11.1. 리소스 에이전트 식별자

리소스에 대해 정의한 식별자는 클러스터에 리소스에 사용할 에이전트, 해당 에이전트를 찾을 위치 및 해당 표준을 찾습니다.

다음 표에서는 리소스 에이전트의 이러한 속성을 설명합니다.

표 11.1. 리소스 에이전트 식별자

필드	설명
표준	에이전트가 준수하는 표준입니다. 허용되는 값 및 해당 의미는 다음과 같습니다. * **OCF** - 지정된 *유형은* Open Cluster Framework Resource Agent API를 준수하고 **/usr/lib/ocf/resource.d/*공급자*아래에 있는 실행 파일의 이름입니다.** ***** LSB - 지정된 *유형은* Linux 표준 기본 Init 스크립트 작업을 따르는 실행 파일의 이름입니다. 유형이 전체 경로를 지정하지 않으면 시스템은 **/etc/init.d** 디렉터리에서 시스템을 찾습니다. **systemd** - 지정된 *유형은* 설치된 **systemd** 장치의 이름입니다. * **service** - Pacemaker는 지정된 *유형을* 먼저 **lsb** 에이전트로 검색한 다음 **systemd** 에이전트로 검색합니다. * rsh - 지정된 *유형은* /usr/share/nagios/plugins 디렉터리에 있으며 **/usr/libexec/nagios/plugins** 디렉터리에 있는 실행 파일의 이름입니다. /OCF 스타일 메타데이터는 **/usr/share/nagios/plugins-metadata** 디렉터리에 별도로 저장됩니다(일반 플러그인의 경우 **nagios -agents-metadata** 패키지에 사용 가능).
type	사용할 리소스 에이전트의 이름(예: **IPaddr** 또는 **Filesystem**)

필드	설명
공급자	OCF 사양을 사용하면 여러 공급 업체가 동일한 리소스 에이전트를 제공할 수 있습니다. Red Hat에서 제공하는 대부분의 에이전트는 공급업체로 **하트비트** 를 사용합니다.

다음 표에는 사용 가능한 리소스 속성을 표시하는 명령이 요약되어 있습니다.

표 11.2. 리소스 속성을 표시하는 명령

pcs Display 명령	출력 결과
pcs resource list	사용 가능한 모든 리소스 목록을 표시합니다.
pcs resource standards	사용 가능한 리소스 에이전트 표준 목록을 표시합니다.
pcs resource providers	사용 가능한 리소스 에이전트 공급자 목록을 표시합니다.
pcs resource list *string*	지정된 문자열로 필터링된 사용 가능한 리소스 목록을 표시합니다. 이 명령을 사용하여 표준, 공급자 또는 유형의 이름으로 필터링된 리소스를 표시할 수 있습니다.

11.2. 리소스별 매개변수 표시

개별 리소스의 경우 다음 명령을 사용하여 리소스에 대한 설명, 해당 리소스에 대해 설정할 수 있는 매개변수 및 리소스에 설정된 기본값을 표시할 수 있습니다.

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

*예를 들어 다음 명령은 **apache** 유형의 리소스에 대한 정보를 표시합니다.*

```
# 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. 리소스 메타 옵션 구성

리소스별 매개 변수 외에도 리소스에 대한 추가 리소스 옵션을 구성할 수 있습니다. 이러한 옵션은 클러스터에서 리소스가 작동하는 방식을 결정하는 데 사용됩니다.

다음 표에서는 리소스 메타 옵션을 설명합니다.

표 11.3. 리소스 메타 옵션

필드	기본값	설명
priority	**0**	모든 리소스가 활성 상태가 아닌 경우 우선 순위가 더 높은 리소스를 활성 상태로 유지하기 위해 클러스터에서 우선순위가 낮은 리소스를 중지합니다.
target-role	**Started**	클러스터가 이 리소스를 유지하려고 시도하는 상태를 나타냅니다. 허용되는 값: * **중지됨** - 리소스를 중지하도록 강제 시행 * **시작** - 리소스가 시작되도록 허용(및 적절한 경우 승격된 복제의 경우) * 승격 - 리소스를 시작하고 해당하는 경우 승격 가능 * **promoted** - 리소스가 승격 가능한 경우에만 지원되지 않는 모드에서만 리소스를 시작할 수 있습니다.
is-managed	**true**	클러스터가 리소스를 시작하고 중지할 수 있는지 여부를 나타냅니다. 허용되는 값: **true**,**false**
resource-stickiness	1	값이 값을 사용하여 리소스가 현재 위치에 남아 있는 것을 선호하는지 지정합니다.

필드	기본값	설명
필수 항목	계산된	리소스를 시작할 수 있는 조건을 나타냅니다. 아래에 명시된 조건을 제외하고 **펜싱** 의 기본값은 기본값입니다. 가능한 값은 다음과 같습니다. * **아무것도 없음** - 클러스터는 항상 리소스를 시작할 수 있습니다. **쿼럼** - 대부분의 구성된 노드가 활성화된 경우에만 클러스터는 이 리소스를 시작할 수 있습니다. **stonith-enabled** 가 **false** 이거나 리소스의 **표준이 stonith** 인 경우 기본값입니다. * **펜싱** - 클러스터는 구성된 대부분의 노드가 활성 *상태이고* 실패하거나 알 수 없는 노드가 펜싱된 경우에만 이 리소스를 시작할 수 있습니다. * **Unfencing** - 클러스터는 대부분의 구성된 노드가 활성 *상태이고* 실패하거나 알 수 없는 노드가 펜싱된 경우에만 이 리소스를 시작할 수 *있습니다. 이 값은 펜싱 장치에 대해* ***provides=unfencing stonith*** *meta 옵션이 설정된 경우 기본값입니다.*
migration-threshold	**INFINITY**	이 노드가 이 리소스를 호스팅할 수 없는 것으로 표시되기 전에 노드에서 이 리소스에 대해 이러한 오류가 발생할 수 있습니다. 값 0은 이 기능이 비활성화되어 있음을 나타냅니다(노드는 볼 수 없음). 반면 클러스터는 **INFINITY** (기본값)를 매우 큰 숫자이지만 한정된 숫자로 처리합니다. 이 옵션은 실패한 작업에 **on-fail=restart** (기본값)가 있는 경우에만 적용되며 클러스터 속성 **start-failure-is-is-fatal** 이 **false** 인 경우 실패한 시작 작업에도 적용됩니다.
failure-timeout	**0** (비활성화)	**migration-threshold** 옵션과 함께 사용되며 오류가 발생하지 않은 것처럼 작동하기 전에 대기할 시간(초)을 나타내며 리소스가 실패한 노드로 다시 허용할 수 있습니다.

필드	기본값	설명
multiple-active	**stop_start**	둘 이상의 노드에서 액티브 리소스를 발견하면 클러스터에서 수행해야 하는 작업을 나타냅니다. 허용되는 값: **블록** - 관리되지 않는 리소스로 표시 * **stop_only** - 활성 인스턴스를 모두 중지하고 해당 인스턴스를 종료합니다. * **stop_start** - 활성 인스턴스를 모두 중지하고 한 위치에서만 리소스를 시작합니다. * **stop_unexpected** - (RHEL 9.1 이상)는 전체 재시작 없이 예기치 않은 리소스 인스턴스만 중지합니다. 서비스 및 해당 리소스 에이전트가 전체 재시작 없이 추가 활성 인스턴스와 함께 작동할 수 있는지 확인하는 것은 사용자의 책임입니다.
심각	**true**	리소스가 리소스 그룹의 일부일 때 생성된 암시적 공동 배치 제약 조건으로 리소스를 종속 리소스(*target_resource*)로 포함하는 모든 공동 배치 제한 조건에 대한 **influence** 옵션에 대한 기본값을 설정합니다. **영향** 공동 배치 제약 조건 옵션은 종속 리소스가 실패의 마이그레이션 임계값에 도달하면 클러스터가 기본 및 종속 리소스를 다른 노드로 이동할지 또는 서비스 전환을 유발하지 않고 클러스터가 종속 리소스를 오프라인으로 유지할지 여부를 결정합니다. **중요** 리소스 메타 옵션에는 기본값인 **true** 또는 **false** 가 있을 수 있습니다.
allow-unhealthy-nodes	**false**	(RHEL 9.1 이상) **true** 로 설정하면 노드 상태가 성능 저하로 인해 리소스가 노드를 강제 해제하지 않습니다. 상태 리소스에 이 속성이 설정된 경우 노드의 상태가 복구되는지 자동으로 탐지하여 리소스를 다시 이동할 수 있습니다. 노드의 상태는 로컬 조건에 따라 상태 리소스 에이전트가 설정한 상태 특성의 조합과 클러스터가 해당 조건에 대응하는 방법을 결정하는 전략 관련 옵션의 조합에 따라 결정됩니다.

11.3.1. 리소스 옵션의 기본값 변경

***pcs resource defaults update** 명령을 사용하여 모든 리소스에 대한 리소스 옵션의 기본값을 변경할 수 있습니다. 다음 명령은 **resource-stickiness** 기본값을 **100**으로 재설정합니다.*

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

이전 릴리스의 모든 리소스에 대한 기본값을 설정하는 원래 **pcs resource defaults name=value** 명령은 둘 이상의 기본값 집합이 구성된 경우가 아니면 계속 지원됩니다. 그러나 **pcs resource defaults** 업데이트 는 이제 기본 명령 버전입니다.

11.3.2. 리소스 세트의 리소스 옵션의 기본값 변경

pcs resource defaults set create 명령을 사용하여 여러 리소스 기본값 세트를 생성할 수 있으므로 리소스 식이 포함된 규칙을 지정할 수 있습니다. 이 명령으로 지정하는 규칙에서는 및 및 괄호를 포함한 리소스 및 날짜 식만 허용됩니다.

pcs resource defaults set create 명령을 사용하면 특정 유형의 모든 리소스에 대한 기본 리소스 값을 구성할 수 있습니다. 예를 들어 중지 시간이 오래 걸리는 데이터베이스를 실행하는 경우 해당 리소스가 원하는 것보다 더 자주 다른 노드로 이동하지 못하도록 데이터베이스 유형의 모든 리소스에 대한 **resource-stickiness** 기본값을 늘릴 수 있습니다.

다음 명령은 유형 **pqsql** 의 모든 리소스에 대해 **resource-stickiness** 의 기본값을 **100**으로 설정합니다.

- 리소스 기본값의 이름을 지정하는 **id** 옵션은 필수가 아닙니다. 이 옵션을 설정하지 않으면 **pcs** 가 **ID**를 자동으로 생성합니다. 이 값을 설정하면 보다 설명적인 이름을 제공할 수 있습니다.
- 이 예에서 **::pgsql** 은 모든 클래스의 리소스, 모든 공급자, **pgsql** 을 의미합니다.
  - **ocf:heartbeat:pgsql** 을 지정하면 클래스 **ocf**, 공급자 하트비트, **pgsql**,
  - **ocf:pacemaker:** 를 지정하면 모든 유형의 **ocf**, 공급자 **pacemaker** 의 모든 리소스가 표시됩니다.

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

기존 세트의 기본값을 변경하려면 **pcs resource defaults set update** 명령을 사용합니다.

11.3.3. 현재 구성된 리소스 기본값 표시

pcs resource defaults 명령은 사용자가 지정한 규칙을 포함하여 현재 구성된 리소스 옵션에 대한 기

본값 목록을 표시합니다.

다음 예제에서는 **resource-stickiness** 기본값을 **100**으로 재설정한 후 이 명령의 출력을 보여줍니다.

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

다음 예제에서는 **pqsql** 유형의 모든 리소스에 대해 **resource-stickiness** 의 기본값을 **100**으로 재설정한 후 이 명령의 출력을 표시하고 **id** 옵션을 **id=pgsql-stickiness** 로 설정합니다.

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

11.3.4. 리소스 생성 시 메타 옵션 설정

리소스 메타 옵션의 기본값을 재설정했는지 여부에 관계없이 리소스를 생성할 때 특정 리소스에 대한 리소스 옵션을 기본값 이외의 값으로 설정할 수 있습니다. 다음은 리소스 메타 옵션의 값을 지정할 때 사용하는 **pcs resource create** 명령의 형식을 보여줍니다.

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

예를 들어 다음 명령은 **resource-stickiness** 값이 **50**인 리소스를 생성합니다.

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

다음 명령을 사용하여 기존 리소스, 그룹 또는 복제 리소스에 대한 리소스 메타 옵션의 값을 설정할 수도 있습니다.

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

다음 예제에서는 **dummy_resource** 라는 기존 리소스가 있습니다. 이 명령은 리소스가 **20**초 내에 동일한 노드에서 재시작할 수 있도록 **failure-timeout** 메타 옵션을 **20**초로 설정합니다.

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

이 명령을 실행한 후 리소스 값을 표시하여 **failure-timeout=20s** 가 설정되었는지 확인할 수 있습니다.

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

11.4. 리소스 그룹 구성

클러스터의 가장 일반적인 요소 중 하나는 함께 배치하고 순차적으로 시작하고 역방향 순서로 중지해야 하는 리소스 집합입니다. 이 구성을 단순화하기 위해 **Pacemaker**는 리소스 그룹의 개념을 지원합니다.

11.4.1. 리소스 그룹 생성

다음 명령을 사용하여 리소스 그룹을 생성하여 그룹에 포함할 리소스를 지정합니다. 그룹이 없으면 이 명령은 그룹을 생성합니다. 그룹이 있는 경우 이 명령은 그룹에 리소스를 추가합니다. 리소스는 이 명령으로 지정한 순서대로 시작되고 시작 순서의 역방향 순서로 중지합니다.

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

이 명령의 **--before** 및 **--after** 옵션을 사용하여 그룹에 이미 존재하는 리소스와 관련하여 추가된 리소스의 위치를 지정할 수 있습니다.

다음 명령을 사용하여 리소스를 생성할 때 기존 그룹에 새 리소스를 추가할 수도 있습니다. 생성한 리소스가 **group_name** 이라는 그룹에 추가됩니다. **group_name** 그룹이 없으면 생성됩니다.

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

그룹에서 포함할 수 있는 리소스 수에는 제한이 없습니다. 그룹의 기본 속성은 다음과 같습니다.

- 리소스는 그룹 내에 배치됩니다.
- 리소스는 지정한 순서대로 시작됩니다. 그룹의 리소스를 어디에서나 실행할 수 없는 경우 해당 리소스 다음에 지정된 리소스를 실행할 수 없습니다.

- 리소스를 지정하는 역방향 순서로 리소스가 중지됩니다.

다음 예제에서는 기존 리소스 **IPaddr** 및 **Email** 이 포함된 바로 가기 라는 리소스 그룹을 생성합니다.

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

이 예제에서는 다음을 수행합니다.

- **IPaddr** 이 먼저 시작된 다음 **Email** 이 시작됩니다.
- **Email** 리소스가 먼저 중지된 다음 **IPAddr.**
- **IPaddr** 은 어디에서나 실행할 수 없는 경우 이메일 도 실행할 수 없습니다.
- 그러나 이메일 은 어디에서나 실행할 수 없는 경우 **IPaddr** 은 어떤 방식으로든 영향을 미치지 않습니다.

11.4.2. 리소스 그룹 제거

다음 명령을 사용하여 그룹에서 리소스를 제거합니다. 그룹에 남아 있는 리소스가 없는 경우 이 명령은 그룹 자체를 제거합니다.

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

11.4.3. 리소스 그룹 표시

다음 명령은 현재 구성된 모든 리소스 그룹을 나열합니다.

```
pcs resource group list
```

11.4.4. 그룹 옵션

리소스 그룹에 대해 다음 옵션을 설정할 수 있으며 단일 리소스에 대해 설정된 경우와 동일한 의미를 유지할 수 있습니다. **priority,target-role,is-managed.** 리소스 메타 옵션에 대한 자세한 내용은 리소스 메

타 옵션 구성을 참조하십시오.

11.4.5. 그룹 고정

자원의 위치를 얼마나 유지하기를 원하는지, 얼마나 많은 자원을 가지고 있는지에 대한 측정은 그룹에 추가됩니다. 그룹의 모든 활성 리소스는 그룹의 합계에 고착 값을 제공합니다. 따라서 기본 **resource-stickiness** 가 **100**이고 그룹에 **7**개의 멤버가 있는 경우 그 중 **5**개가 활성 상태인 경우 전체 그룹이 현재 위치를 **500**으로 우선합니다.

11.5. 리소스 동작 확인

해당 리소스에 대한 제약 조건을 구성하여 클러스터의 리소스 동작을 확인할 수 있습니다. 다음 유형의 제약 조건을 구성할 수 있습니다.

- 위치 제한 조건 - 리소스가 실행할 수 있는 노드를 결정합니다. 위치 제약 조건을 구성하는 방법에 대한 자세한 내용은 리소스가 실행할 수 있는 노드 결정을 참조하십시오.
- 순서 제약 조건 - 순서 제한 조건에 따라 리소스가 실행되는 순서를 결정합니다. 순서 제한 조건 구성에 대한 자세한 내용은 클러스터 리소스가 실행되는 순서 결정을 참조하십시오.
- 공동 배치 제약 조건 - 공동 배치 제한 조건은 다른 리소스에 대해 리소스를 배치할 위치를 결정합니다. 공동 배치 제약 조건에 대한 자세한 내용은 클러스터 리소스 배치를 참조하십시오.

리소스 세트를 함께 찾고 리소스를 순차적으로 시작하고 역순으로 중지되는지 확인하는 제약 조건 집합을 구성하는 한 **Pacemaker**에서 리소스 그룹의 개념을 지원합니다. 리소스 그룹을 생성한 후에는 개별 리소스에 대한 제약 조건을 구성하는 것처럼 그룹 자체에 대한 제약 조건을 구성할 수 있습니다.

12장. 리소스에서 실행할 수 있는 노드 확인

위치 제한 조건은 리소스에서 실행할 수 있는 노드를 결정합니다. 리소스 우선 여부를 결정하거나 지정된 노드를 방지할 수 있도록 위치 제약 조건을 구성할 수 있습니다.

위치 제약 조건 외에도 리소스를 실행하는 노드는 해당 리소스의 **resource-stickiness** 값의 영향을 받으며, 리소스가 현재 실행 중인 노드에 남아 있는 정도를 결정합니다. **resource-stickiness** 값을 설정하는 방법에 대한 자세한 내용은 현재 노드를 선호하도록 리소스 구성을 참조하십시오.

12.1. 위치 제한 조건 구성

리소스가 노드를 선호하는지 여부를 지정하도록 기본 위치 제약 조건을 구성할 수 있으며 선택적 점수 값은 제약 조건에 대한 상대적 기본 설정 수준을 나타낼 수 있습니다.

다음 명령은 지정된 노드 또는 노드를 선호하는 리소스에 대한 위치 제한 조건을 생성합니다. 단일 명령으로 둘 이상의 노드에 대해 특정 리소스에 대한 제약 조건을 생성할 수 있습니다.

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

다음 명령은 지정된 노드 또는 노드를 방지하기 위해 리소스에 대한 위치 제한 조건을 생성합니다.

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

다음 표에는 위치 제약 조건을 구성하는 기본 옵션의 의미가 요약되어 있습니다.

표 12.1. 위치 제한 조건 옵션

필드	설명
rsc	리소스 이름
node	노드의 이름

필드	설명
점수	지정된 리소스가 지정된 노드를 선호하는지 또는 사용하지 않아야 하는지에 대한 기본 설정 수준을 나타내는 양의 정수 값입니다. **INFINITY** 는 리소스 위치 제약 조건의 기본 **점수** 값입니다. **pcs constraint location *rsc* prefers** 명령의 **점수** 값은 **리소스가 노드를 사용할 수 있는 경우 해당 노드를 선호하는 것을 선호하지만 지정된 노드를 사용할 수 없는 경우 리소스가 다른 노드에서 실행되지 않는 것을 나타냅니다.** **pcs constraint 위치 *rsc* avoids 명령의 점수 값은 다른 노드를 사용할 수 없는 경우에도 해당 노드에서 리소스가 실행되지 않음을 나타냅니다. 이는 점수가 -INFINITY 인 pcs constraint location add 명령을 설정하는 것과 동일합니다.** **숫자 점수(즉, INFINITY가 아님)는 제약 조건이 선택 사항임을 나타내며 다른 요인은 이를 제외하지 않습니다. 예를 들어, 리소스가 이미 다른 노드에 배치되고 resource-stickiness 점수가 prefers 위치 제약 조건의 점수보다 높으면 리소스는 그대로 유지됩니다.**

*다음 명령은 리소스 **Webserver** 에서 노드 **node1** 을 우선하도록 지정하는 위치 제한 조건을 생성합니다.*

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

***pcs** 는 명령줄의 위치 제약 조건에서 정규식을 지원합니다. 이러한 제약 조건은 일치하는 리소스 이름에 따라 여러 리소스에 적용됩니다. 이를 통해 단일 명령줄을 사용하여 여러 위치 제약 조건을 구성할 수 있습니다.*

*다음 명령은 위치 제한 조건을 생성하여 리소스가 **dummy0** 을 **dummy9 prefer node1** 로 지정합니다.*

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

***Pacemaker**는*

http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html#tag_09_04 에 설명된 POSIX 확장 정규식을 사용하므로 다음 명령을 사용하여 동일한 제약 조건을 지정할 수 있습니다.

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

12.2. 노드의 하위 집합으로 리소스 검색 제한

Pacemaker가 리소스를 어디에서나 시작하기 전에 먼저 모든 노드에서 일회성 모니터 작업(예: "probe")을 실행하여 리소스가 이미 실행 중인지 확인합니다. 이 리소스 검색 프로세스에서는 모니터를 실행할 수 없는 노드에 오류가 발생할 수 있습니다.

노드에 위치 제한 조건을 구성할 때 pcs constraint location 명령의 resource-discovery 옵션을 사용하여 Pacemaker에서 지정된 리소스의 리소스 검색을 수행해야 하는지의 기본 설정을 표시할 수 있습니다. 리소스 검색을 물리적으로 실행할 수 있는 노드의 하위 집합으로 제한하면 대규모 노드 세트가 있을 때 성능이 크게 향상될 수 있습니다. pacemaker_remote 를 사용하여 노드 수를 수백 개의 노드 범위로 확장하는 경우 이 옵션을 고려해야 합니다.

다음 명령은 pcs constraint location 명령의 resource-discovery 옵션을 지정하는 형식을 보여줍니다. 이 명령에서 양수 값은 노드를 선호하도록 리소스를 구성하는 기본 위치 제약 조건에 해당하지만 점수의 음수 값은 노드를 방지하기 위해 리소스를 구성하는 기본 위치 지정에 해당합니다. 기본 위치 제약 조건과 마찬가지로 이러한 제약 조건이 있는 리소스에도 정규식을 사용할 수 있습니다.

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

다음 표에는 리소스 검색에 대한 제약 조건을 구성하기 위한 기본 매개변수의 의미가 요약되어 있습니다.

표 12.2. 리소스 검색 제약 조건 매개변수

필드	설명
id	제약 조건 자체의 사용자 선택 이름입니다.
rsc	리소스 이름
node	노드의 이름

점수	지정된 리소스가 지정된 노드를 선호하는지 또는 사용하지 않아야 하는지에 대한 기본 설정 수준을 나타내는 정수 값입니다. 점수의 양수 값은 노드를 선호하도록 리소스를 구성하는 기본 위치 제약 조건에 해당하지만 점수의 음수 값은 노드를 방지하기 위해 리소스를 구성하는 기본 위치 제약 조건에 해당합니다. **INFINITY** for **score** 값은 노드를 사용할 수 있는 경우 리소스가 노드를 선호하지만 지정된 노드를 사용할 수 없는 경우 리소스가 다른 노드에서 실행되지 않도록 합니다. **점수** 의 **-INFINITY** 값은 다른 노드를 사용할 수 없는 경우에도 해당 노드에서 리소스가 실행되지 않음을 나타냅니다. **INFINITY** 또는 **-INFINITY**가 아닌 숫자 점수는 제약 조건이 선택 사항임을 의미하며 다른 요인이 제외되면 적용됩니다. 예를 들어, 리소스가 이미 다른 노드에 배치되고 **resource-stickiness** 점수가 **prefers** 위치 제약 조건의 점수보다 높으면 리소스는 그대로 유지됩니다.
resource-discovery 옵션	**Always** - 이 노드에서 지정된 리소스에 대해 항상 리소스 검색을 수행합니다. 이는 리소스 위치 제약 조건의 기본 **resource-discovery** 값입니다. **Never** - 이 노드에서 지정된 리소스에 대해 리소스 검색을 수행하지 마십시오. * **exclusive** - 이 노드에서만 지정된 리소스에 대해서만 리소스 검색을 수행합니다(및 **배타적**으로 표시된 다른 노드). 서로 다른 노드에서 동일한 리소스에 대한 **전용** 검색을 사용하는 여러 위치 제한 조건은 **리소스 검색의 하위 집합을 생성하는** 방법은 다음과 같습니다. 하나 이상의 노드에서 리소스를 **배타적** 으로 검색하도록 표시된 경우 해당 리소스는 노드의 하위 집합 내에만 배치할 수 있습니다.

주의

***resource-discovery** 를 **never** 또는 **exclusive** 로 설정하면 **Pacemaker**에서 원하지 않는 서비스 인스턴스를 탐지하고 중지합니다. 시스템 관리자는 리소스 검색 없이 (예: 관련 소프트웨어를 제거한 상태로 두는 등) 노드에서 서비스를 활성화할 수 없도록 하는 것은 아닙니다.*

12.3. 위치 제한 전략 구성

위치 제약 조건을 사용하는 경우 리소스에서 실행할 수 있는 노드를 지정하기 위해 일반 전략을 구성할 수 있습니다.

- *옵트인 클러스터 - 기본적으로 리소스를 실행할 수 없는 클러스터를 구성한 다음 특정 리소스에 대해 허용되는 노드를 선택적으로 활성화합니다.*

- *옵트아웃 클러스터 - 기본적으로 모든 리소스를 실행하고 특정 노드에서 실행되지 않는 리소스에 대한 위치 제한 조건을 생성할 수 있는 클러스터를 구성합니다.*

클러스터를 옵트인 또는 옵트아웃 클러스터로 구성할지 여부는 클러스터 설정 및 구성 설정에 따라 달라집니다. 대부분의 노드에서 리소스를 실행할 수 있는 경우 옵트아웃 배열은 더 간단한 구성을 초래할 수 있습니다. 반면 대부분의 리소스가 노드의 작은 하위 집합에서만 실행할 수 있는 경우 옵트인 구성이 더 간단할 수 있습니다.

12.3.1. "Opt-In" 클러스터 구성

*옵트인 클러스터를 생성하려면 기본적으로 리소스가 어디에서나 실행되지 않도록 **symmetric-cluster** 클러스터 속성을 **false** 로 설정합니다.*

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

*개별 리소스의 노드를 활성화합니다. 다음 명령은 리소스 **Webserver** 리소스가 노드 **example-1** 을 선호하고, 리소스 **Database** 는 노드 **example-2** 를 선호하며, 기본 노드가 실패하는 경우 두 리소스가 노드 **example-3** 을 초과할 수 있도록 위치 제약 조건을 구성합니다. 옵트인 클러스터에 대한 위치 제한 조건을 구성할 때 **0**을 설정하면 노드 우선 순위 또는 방지를 표시하지 않고 노드에서 리소스를 실행할 수 있습니다.*

```
# 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. "Opt-Out" 클러스터 구성

*옵트아웃 클러스터를 생성하려면 기본적으로 리소스가 모든 위치에서 실행되도록 **symmetric-cluster** 클러스터 속성을 **true** 로 설정합니다. **symmetric-cluster** 가 명시적으로 설정되지 않은 경우 기본 구성입니다.*

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

*그러면 다음 명령에서 **"Opt-In"** 클러스터 구성"에서 예제와 동일한 구성을 생성합니다. 모든 노드에는 암시적 점수가 **0**이므로 기본 노드가 실패하면 두 리소스 모두 **node example-3** 로 장애 조치할 수 있습니*

다.

```
# 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
```

이 명령에서 점수는 점수의 기본값이므로 해당 명령에서 INFINITY 점수를 지정할 필요는 없습니다.

12.4. 현재 노드를 선호하도록 리소스 구성

리소스에는 리소스 메타 옵션 구성에 설명된 대로 리소스를 생성할 때 meta 속성으로 설정할 수 있는 resource-stickiness 값이 있습니다. resource-stickiness 값은 현재 실행 중인 노드에 남아 있는 리소스 양을 결정합니다. Pacemaker는 다른 설정 (예: 위치 제약 조건의 점수 값)과 함께 resource-stickiness 값을 사용하여 리소스를 다른 노드로 이동할지 아니면 그대로 유지할지 여부를 결정합니다.

resource-stickiness 값이 0이면 클러스터에서 리소스를 노드 간에 분산시키는 데 필요한 대로 이동할 수 있습니다. 그러면 관련 리소스가 시작되거나 중지되면 리소스가 이동될 수 있습니다. 긍정적인 고착성을 사용하면 리소스를 우선적으로 유지하고 다른 상황이 고착성을 능가하는 경우에만 이동해야 합니다. 이로 인해 새로 추가된 노드는 관리자의 개입 없이 해당 노드에 리소스를 할당하지 못할 수 있습니다.

RHEL 9에서 새로 만든 클러스터는 resource-stickiness 의 기본값을 1로 설정합니다. 이 작은 값은 생성한 다른 제약 조건으로 쉽게 재정의할 수 있지만 Pacemaker가 클러스터 전체에서 정상적인 리소스를 불필요하게 이동하는 것을 방지하는 데 충분합니다. resource-stickiness 값이 0인 클러스터 동작을 원하는 경우 다음 명령을 사용하여 resource-stickiness 기본값을 0으로 변경할 수 있습니다.

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

기존 클러스터를 RHEL 9로 업그레이드하고 resource-stickiness의 기본값을 명시적으로 설정하지 않은 경우 resource-stickiness 값은 0으로 유지되고 pcs resource defaults 명령은 고착도를 위한 어떤 것도 표시하지 않습니다.

양의 resource-stickiness 값을 사용하면 리소스가 새로 추가된 노드로 이동되지 않습니다. 리소스 분산이 필요한 경우 일시적으로 resource-stickiness 값을 0으로 설정할 수 있습니다.

위치 제한 조건 점수가 resource-stickiness 값보다 큰 경우에도 클러스터에서 위치 제한 조건이 가리키는 노드로 정상적인 리소스를 이동할 수 있습니다.

Pacemaker에서 리소스 배치 위치를 결정하는 방법에 대한 자세한 내용은 노드 배치 전략 구성을 참조

하십시오.

12.5. 리소스 제약 조건을 PCS 명령으로 내보내기

Red Hat Enterprise Linux 9.3부터 pcs constraint 명령의 --output-format=cmd 옵션을 사용하여 다른 시스템에서 구성된 리소스 제약 조건을 다시 생성하는 데 사용할 수 있는 pcs 명령을 표시할 수 있습니다.

다음 명령은 IPaddr2 리소스 및 apache 리소스를 생성합니다.

```
# 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')
```

다음 명령은 위치 제약 조건, 공동 배치 제약 조건 및 두 리소스에 대한 순서 제약 조건을 구성합니다.

```
# 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)
```

리소스 및 제약 조건을 생성한 후 다음 명령은 다른 시스템에서 제약 조건을 다시 생성하는 데 사용할 수 있는 pcs 명령을 표시합니다.

```
# 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
```

13장. 클러스터 리소스가 실행되는 순서 확인

리소스가 실행되는 순서를 결정하려면 순서 제한 조건을 구성합니다.

다음은 순서 제한 조건을 구성하는 명령의 형식을 보여줍니다.

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

다음 표에는 순서 제한 조건을 구성하기 위한 속성 및 옵션이 요약되어 있습니다.

표 13.1. 순서 제약 조건의 속성

필드	설명
resource_id	작업이 수행되는 리소스의 이름입니다.
작업	리소스에 대해 정렬할 작업입니다. *action* 속성의 가능한 값은 다음과 같습니다. * **start** - 리소스의 시작 작업 순서 지정. * **stop** (종료) - 리소스의 중지 작업 순서를 지정합니다. * **승격** - 프로모션되지 않은 리소스에서 승격된 리소스로 리소스를 승격합니다. **demote** - 승격된 리소스에서 보호되지 않은 리소스로 리소스를 시연합니다. 작업을 지정하지 않으면 기본 작업이 **시작됩니다**.

필드	설명
유형 옵션	제약 조건을 적용하는 방법 **kind** 옵션의 가능한 값은 다음과 같습니다. * **선택 사항** - 두 리소스 모두 지정된 작업을 실행하는 경우에만 적용됩니다. 선택적 순서 지정에 대한 자세한 내용은 권고 순서 구성을 참조하십시오. * **필수** - 항상 제약 조건을 적용합니다(기본값). 지정한 첫 번째 리소스가 중지되거나 시작할 수 없는 경우 지정한 두 번째 리소스를 중지해야 합니다. 필수 순서 지정에 대한 자세한 내용은 필수 순서 지정을 참조하십시오. * **serialize - 지정하는 리소스에 대해 두 개의 중지/시작 작업이 동시에 발생하지 않도록 합니다.* serialize** - Ensure that no two stop/start actions occur concurrently for the resources you specify. 지정한 첫 번째 리소스와 두 번째 리소스는 순서대로 시작할 수 있지만 다른 리소스를 시작하기 전에 완료해야 합니다. 일반적인 사용 사례는 리소스 시작 시 호스트에 부하가 높은 경우입니다.
대칭 옵션	true인 경우 제약 조건의 역방향이 반대의 작업에 적용됩니다(예: A가 시작된 후 B가 중지되기 전에 B 중지). **종류** 인 순서 제한 조건은 대칭일 수 없습니다. **Serialize의 경우 기본값은 Mandatory 및 선택적 종류의 경우 false 입니다.**

다음 명령을 사용하여 순서가 지정된 제약 조건에서 리소스를 제거합니다.

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

13.1. 필수 순서 구성

필수 순서 지정 제약 조건은 첫 번째 작업이 첫 번째 리소스가 성공적으로 완료될 때까지 두 번째 리소스에 대해 두 번째 작업을 시작하지 않아야 함을 나타냅니다. 주문될 수 있는 작업은 승격 테이블 복제, 데모 및 승격 에 대한 중지,시작 및 추가 기능입니다. 예를 들어, "A then B" (이는 "start A, start B")는 A가 성공적으로 시작될 때까지 B가 시작되지 않음을 의미합니다. 제약 조건의 kind 옵션이 Mandatory 로 설정되거나 기본값으로 남아 있는 경우 순서 제약 조건이 필요합니다.

대칭 옵션이 true 로 설정되었거나 기본값으로 유지되면 반대의 동작이 순서대로 정렬됩니다. 시작 및 중지 작업은 반대이며, 데모와 프로모션 은 반대입니다. 예를 들어 대칭 "프로모트 A"는 "B then demote

A"를 중지한다는 것을 의미합니다. 즉, B가 성공적으로 중지되지 않는 한 A를 예상 할 수 없음을 의미합니다. 대칭 순서 지정은 A의 상태에 있는 변경이 B에 대해 작업을 예약할 수 있음을 의미합니다. 예를 들어 "A then B"는 실패로 인해 재시작이 중지되면 A가 중지되면 A가 시작되고 B가 시작됩니다.

클러스터는 각 상태 변경에 반응합니다. 첫 번째 리소스가 다시 시작되고 두 번째 리소스가 중지 작업을 시작하기 전에 다시 시작되면 두 번째 리소스를 다시 시작할 필요가 없습니다.

13.2. 권고 순서 구성

정렬 제약 조건에 kind=Optional 옵션이 지정되면 제약 조건은 선택 사항으로 간주되며 두 리소스가 모두 지정된 작업을 실행하는 경우에만 적용됩니다. 지정한 첫 번째 리소스에서 상태 변경 사항은 지정한 두 번째 리소스에는 영향을 미치지 않습니다.

다음 명령은 VirtualIP 및 dummy_resource 라는 리소스에 대한 권고 순서 제약 조건을 구성합니다.

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

13.3. 순서가 지정된 리소스 세트 구성

관리자가 순서가 지정된 리소스 체인을 생성하는 경우가 있습니다. 예를 들어 리소스 A가 리소스 C보다 먼저 시작되기 전에 리소스 A가 시작됩니다. 구성에 배치 및 시작된 리소스 세트를 순서대로 생성해야 하는 경우 해당 리소스가 포함된 리소스 그룹을 구성할 수 있습니다.

그러나 리소스 그룹이 적합하지 않기 때문에 지정된 순서로 시작해야 하는 리소스를 구성하는 경우가 있습니다.

- *시작하기 위해 리소스를 구성해야 할 수 있으며 리소스가 반드시 결합되지는 않습니다.*
- *A 또는 B 리소스를 시작한 후 시작해야 하는 리소스 C가 있을 수 있지만 A와 B 간의 관계가 없습니다.*
- *A와 B 리소스가 모두 시작된 후 시작해야 하는 리소스 C와 D가 있을 수 있지만, A와 B 또는 C와 D 사이에 관계가 없습니다.*

이러한 경우 pcs constraint order set 명령을 사용하여 세트 또는 리소스 집합에 대한 순서 제한 조건을 생성할 수 있습니다.

pcs constraint order set 명령을 사용하여 리소스 집합에 대해 다음 옵션을 설정할 수 있습니다.

- *순차적인 경우 리소스 집합을 서로 기준으로 정렬해야 하는지 여부를 나타내기 위해 **true** 또는 **false** 로 설정할 수 있습니다. 기본값은 **true**입니다.*

  sequential 를 false 로 설정하면 정렬 제약 조건의 다른 세트에 대해 집합을 정렬할 수 있으며 멤버는 서로 상대적으로 정렬되지 않습니다. 따라서 제약 조건에 여러 세트가 나열된 경우에만 이 옵션이 적합합니다. 그러지 않으면 제약 조건이 적용되지 않습니다.

- *keep-all - 계속 진행하기 전에 세트의 모든 리소스가 활성화되어야 하는지 여부를 나타내기 위해 true 또는 false 로 설정할 수 있습니다. require-all 을 false 로 설정하면 다음 세트를 진행하기 전에 세트의 리소스 하나만 시작해야 합니다. order되지 않은 세트와 함께 사용되는 경우 require-all 을 false 로 설정하면 sequential 가 false 로 설정된 경우 적용됩니다. 기본값은 true입니다.*

- *클러스터 리소스가 실행되는 순서를 결정하여 "주문 제약 조건의 속성" 테이블에 설명된 대로 작업을 시작하도록 설정할 수 있습니다.*

- *role (역할) - Stopped,Started, Promoted또는 Unpromoted 로 설정할 수 있습니다.*

pcs constraint order set 명령의 setoptions 매개변수에 따라 리소스 집합에 대해 다음 제약 조건 옵션을 설정할 수 있습니다.

- *ID: 정의 중인 제약 조건의 이름을 제공합니다.*

- *kind. . . . . https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_high_availability_clusters/assembly_determining-resource-order.adoc-configuring-and-managing-high-availability-clusters*

- *대칭 형을 사용하여 제약 조건의 역방향이 클러스터 리소스가 실행되는 순서를 결정하는 "주문 제약 조건의 속성" 테이블에 설명된 대로 반대 작업에 적용할지 여부를 설정합니다.*

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

D1,D2 및 D3 라는 리소스가 세 개 있는 경우 다음 명령을 실행하면 순서가 지정된 리소스 세트로 구성됩니다.

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

A,B,C,D,E 및 F 라는 6개의 리소스가 있는 경우 이 예제에서는 다음과 같이 시작하는 리소스 집합에 대한 순서 제한 조건을 구성합니다.

- *A 와 B 는 서로 독립적으로 시작*
- *A 또는 B 가 시작되면 C 를 시작합니다.*
- *C 가 시작되면 D 를 시작합니다.*
- *D 가 시작되면 E 및 F 가 서로 독립적으로 시작합니다.*

symmetrical=false 가 설정되어 있기 때문에 리소스 중지는 이 제약 조건의 영향을 받지 않습니다.

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

13.4. PACEMAKER에서 관리하지 않는 리소스 종속 항목의 시작 순서 구성

클러스터에서 클러스터가 관리하지 않는 종속 항목이 있는 리소스를 포함할 수 있습니다. 이 경우 Pacemaker를 시작하기 전에 해당 종속 항목이 시작되고 Pacemaker가 중지된 후 중지되었는지 확인해야 합니다.

systemd resource-agents-deps 대상을 통해 이러한 상황을 설명하도록 시작 순서를 구성할 수 있습니다. 이 대상에 대한 systemd 드롭인 장치를 생성할 수 있으며 Pacemaker는 이 대상을 기준으로 적절하게 주문합니다.

예를 들어 클러스터에 클러스터에서 관리하지 않는 외부 서비스 foo 에 따라 달라지는 리소스가 포함된 경우 다음 절차를 수행합니다.

1.

다음 항목이 포함된 드롭인 **/etc/systemd/system/resource-agents-deps.target.d/foo.conf** 를 만듭니다.

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

2. **systemctl daemon-reload** 명령을 실행합니다.

이 방식으로 지정된 클러스터 종속성은 서비스 이외의 다른 것일 수 있습니다. 예를 들어 **/srv** 에 파일 시스템 마운트에 대한 종속성이 있을 수 있습니다. 이 경우 다음 절차를 수행할 수 있습니다.

1. **/srv** 가 **/etc/fstab** 파일에 나열되어 있는지 확인합니다. 이 값은 시스템 관리자의 구성이 다시 로드될 때 부팅 시 **systemd** 파일 **srv.mount** 로 자동으로 변환됩니다. 자세한 내용은 **systemd.mount(5)** 및 **systemd-fstab-generator(8)** 매뉴얼 페이지를 참조하십시오.

2. **Pacemaker**가 디스크가 마운트된 후 시작되도록 드롭인 **/etc/systemd/system/resource-agents-deps.target.d/srv.conf** 를 생성합니다.

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

3. **systemctl daemon-reload** 명령을 실행합니다.

Pacemaker 클러스터에서 사용하는 **LVM** 볼륨 그룹에 **iSCSI** 대상과 같은 원격 블록 스토리지에 있는 하나 이상의 물리 볼륨이 포함된 경우 **Pacemaker**가 시작되기 전에 서비스가 시작되도록 대상에 대해 **systemd resource-agents-deps** 대상 및 **systemd** 드롭인 장치를 구성할 수 있습니다.

다음 절차에서는 **blk-availability.service** 를 종속성으로 구성합니다. **blk-availability.service** 서비스는 다른 서비스 중에서 **iscsi.service** 를 포함하는 래퍼입니다. 배포에 필요한 경우 **iscsi.service (iSCSI**만 해당) 또는 **remote-fs.target** 을 **blk-availability** 대신 종속성으로 구성할 수 있습니다.

1. 다음 항목이 포함된 드롭인 **/etc/systemd/system/resource-agents-deps.target.d/blk-availability.conf** 를 만듭니다.

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

-

2. ***systemctl daemon-reload** 명령을 실행합니다.*

14장. 클러스터 리소스 찾기

한 리소스의 위치가 다른 리소스의 위치에 따라 달라지도록 지정하려면 공동 배치 제한 조건을 구성합니다.

두 리소스 간에 공동 배치 제한 조건을 생성하는 데 중요한 부작용이 있습니다. 이는 노드에 리소스를 할당하는 순서에 영향을 미칩니다. 리소스 **B**가 어디에 있는지 모르는 경우가 아니면 리소스 **B**에 상대적인 리소스를 배치할 수 없기 때문입니다. 따라서 공동 배치 제한 조건을 생성할 때 리소스 **A**와 리소스 **B**와 리소스 **A**를 배치해야 하는지를 고려하는 것이 중요합니다.

배치 제한 조건을 생성할 때 고려해야 할 또 다른 사항은 리소스 **A**가 리소스 **B**와 배치된다고 가정하면 클러스터가 리소스 **B**에 대해 선택할 노드를 결정할 때 리소스 **A**의 기본 설정을 고려한다는 것입니다.

다음 명령은 공동 배치 제한 조건을 생성합니다.

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

다음 표에는 공동 배치 제약 조건을 구성하기 위한 속성 및 옵션이 요약되어 있습니다.

표 14.1. 공동 배치 제약 조건의 매개변수

매개변수	설명
source_resource	공동 배치 소스입니다. 제약 조건을 충족할 수 없는 경우 클러스터에서 리소스가 전혀 실행되도록 허용하지 않도록 할 수 있습니다.
target_resource	공동 배치 대상입니다. 클러스터는 이 리소스를 먼저 배치할 위치를 결정한 다음 소스 리소스를 배치할 위치를 결정합니다.
점수	양수 값은 동일한 노드에서 리소스를 실행해야 함을 나타냅니다. 음수 값은 동일한 노드에서 리소스를 실행할 수 없음을 나타냅니다. 기본값은 *source_resource* 가 *target_resource* 와 동일한 노드에서 실행되어야 함을 나타냅니다. **값이 -INFINITY 는 *source_resource* 가 *target_resource* 와 동일한 노드에서 실행되지 않아야 함을 나타냅니다.**

매개변수	설명
영향 옵션	종속 리소스가 실패에 도달한 경우 클러스터가 기본 리소스 (source_*resource*) 및 종속 리소스(*target_resource)* 를 모두 다른 노드로 이동할지 또는 클러스터가 서비스 전환을 유발하지 않고 종속 리소스를 오프라인으로 나가는지 여부를 결정합니다. **영향** 공동 배치 제약 조건 옵션에는**true** 또는**false** 값이 있을 수 있습니다. 이 옵션의 기본값은 종속 리소스의 **중요한** 리소스 메타 옵션의 값으로 결정되며, 기본값은**true** 입니다. 이 옵션의 값이 **true** 인 경우 Pacemaker는 기본 및 종속 리소스를 모두 활성 상태로 유지합니다. 종속 리소스가 실패의 마이그레이션 임계값에 도달하면 두 리소스 모두 가능한 경우 다른 노드로 이동합니다. 이 옵션의 값이 **false** 인 경우 Pacemaker는 종속 리소스 상태의 결과로 기본 리소스를 이동하지 않습니다. 이 경우 종속 리소스가 실패의 마이그레이션 임계값에 도달하면 기본 리소스가 활성 상태이고 현재 노드에 남아 있을 수 있는 경우 중지됩니다.

14.1. 리소스의 필수 배치 지정

*필수 배치는 제약 조건의 점수가 **+INFINITY** 또는 **-INFINITY** 인 경우 언제든지 수행됩니다. 이러한 경우 제약 조건을 충족할 수 없는 경우 **source_resource** 를 실행할 수 없습니다. **score=INFINITY** 의 경우 **target_resource** 가 활성화되지 않은 경우가 포함됩니다.*

***myresource1** 이 항상 **myresource2** 와 동일한 머신에서 실행하려면 다음 제약 조건을 추가합니다.*

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

***INFINITY** 가 사용되었으므로 **myresource2** 를 클러스터 노드(있는 이유)에서 실행할 수 없는 경우 **myresource1** 을 실행할 수 없습니다.*

*또는 **myresource1** 이 **myresource2** 와 동일한 시스템에서 실행할 수 없는 클러스터를 반대로 구성할 수도 있습니다. 이 경우 **use score=-INFINITY***

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

*다시 **-INFINITY** 를 지정하면 제약 조건이 바인딩됩니다. 실행할 수 있는 유일한 위치가 **myresource2** 가 이미 있는 경우 **myresource1** 은 어디에서나 실행할 수 없습니다.*

14.2. 리소스의 권고 배치 지정

리소스의 권고 배치는 리소스 배치가 기본 설정이지만 필수는 아님을 나타냅니다. **-INFINITY** 보다 크고 **INFINITY** 이하의 점수가 있는 제약 조건의 경우 클러스터는 귀하의 요청을 수용하려고 하지만 일부 클러스터 리소스를 중지하려는 경우 이를 무시할 수 있습니다.

14.3. 리소스 세트 공동 배치

구성에 배치 및 시작된 리소스 세트를 순서대로 생성해야 하는 경우 해당 리소스가 포함된 리소스 그룹을 구성할 수 있습니다. 그러나 리소스 그룹으로 배치해야 하는 리소스를 구성하는 것은 적절하지 않은 경우가 있습니다.

- 리소스 세트를 배치해야 할 수도 있지만, 리소스가 반드시 순서대로 시작할 필요는 없습니다.
- 리소스 **A** 또는 **B**와 함께 배치되어야 하는 리소스 **C**가 있을 수 있지만 **A**와 **B** 사이에는 관계가 없습니다.
- **A** 및 **B** 리소스 모두에 배치되어야 하는 **C** 및 **D** 리소스가 있을 수 있지만 **A**와 **B** 또는 **D** 사이에는 관계가 없습니다.

이러한 경우 **pcs constraint colocation set** 명령을 사용하여 세트 또는 리소스 집합에 공동 배치 제약 조건을 생성할 수 있습니다.

pcs constraint colocation set 명령을 사용하여 리소스 집합에 대해 다음 옵션을 설정할 수 있습니다.

- 집합의 멤버를 서로 결합해야 하는지 여부를 나타내기 위해 순차적 또는 **false** 로 설정할 수 있습니다.**Sequent , which can be set to true or false to indicate whether the members of the set must be colocated with each other.**

  sequential 를 **false** 로 설정하면 이 세트의 멤버가 활성 상태의 멤버와 관계없이 나중에 제약 조건에 나열된 다른 집합과 결합할 수 있습니다. 따라서 이 옵션은 제약 조건의 이 집합 이후에 다른 집합이 나열되는 경우에만 의미가 있습니다. 그렇지 않으면 제약 조건이 적용되지 않습니다.

- **role** (역할) **- Stopped,Started, Promoted**또는 **Unpromoted** 로 설정할 수 있습니다.

pcs constraint colocation set 명령의 **setoptions** 매개변수에 따라 리소스 세트에 대해 다음 제약 조

건 옵션을 설정할 수 있습니다.

- **ID:** 정의 중인 제약 조건의 이름을 제공합니다.
- 이 제약 조건에 대한 기본 설정 수준을 나타내는 점수 입니다. 이 옵션에 대한 자세한 내용은 위치 제약 조건 구성의 **"Location Constraint Options"** 표를 참조하십시오.

집합의 멤버를 나열할 때 각 멤버는 앞에 배치됩니다. 예를 들어 **"set A B"**는 **"B**가 **A**와 함께 배치됨**"**을 의미합니다. 그러나 여러 세트를 나열할 때 각 세트는 그 뒤에 하나씩 배치됩니다. 예를 들어, **"set C D sequential=false set A"**는 **"set C D(C**와 **D**가 서로 관계가 없는 경우**) set A B(**여기서 **B**가 **A**와 결합됨**)** 를 의미합니다.

다음 명령은 리소스 집합 또는 세트에 공동 배치 제약 조건을 생성합니다.

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

다음 명령을 사용하여 **source_resource** 와의 공동 배치 제약 조건을 제거합니다.

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

15장. 리소스 제약 조건 및 리소스 종속성 표시

구성된 제약 조건을 표시하는 데 사용할 수 있는 몇 가지 명령이 있습니다. 구성된 모든 리소스 제약 조건을 표시하거나 **esource** 제약 조건 표시를 특정 유형의 리소스 제약 조건으로 제한할 수 있습니다. 구성된 리소스 종속성을 표시할 수도 있습니다.

구성된 모든 제약 조건 표시

다음 명령은 모든 현재 위치, 순서 및 공동 배치 제약 조건을 나열합니다. **full** 옵션이 지정된 경우 내부 제약 조건 **ID**를 표시합니다.

```
pcs constraint [list|show] [--full]
```

기본적으로 리소스 제약 조건을 나열해도 만료된 제약 조건이 표시되지 않습니다. 만료된 구성 요소를 목록에 포함하려면 **pcs constraint** 명령의 **--all** 옵션을 사용합니다. 이렇게 하면 제한 조건 및 관련 규칙이 표시되지 않는 만료된 제약 조건이 나열됩니다.

위치 제한 조건 표시

다음 명령은 모든 현재 위치 제약 조건을 나열합니다.

- 리소스를 지정하면 리소스당 위치 제약 조건이 표시됩니다. 이는 기본 동작입니다.
- 노드를 지정하면 노드당 위치 제약 조건이 표시됩니다.
- 특정 리소스 또는 노드를 지정하면 해당 리소스 또는 노드에 대한 정보만 표시됩니다.

```
pcs constraint location [show [resources [resource...]] | [nodes [node...]]] [--full]
```

순서 제한 조건 표시

다음 명령은 현재 순서 지정 제약 조건을 모두 나열합니다.

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

공동 배치 제약 조건 표시

다음 명령은 현재 공동 배치 제약 조건을 모두 나열합니다.

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

리소스별 제약 조건 표시

다음 명령은 특정 리소스를 참조하는 제약 조건을 나열합니다.

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

리소스 종속성 표시

다음 명령은 트리 구조에서 클러스터 리소스 간의 관계를 표시합니다.

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

--full 옵션을 사용하면 명령에서 제약 조건 ID 및 리소스 유형을 포함하여 추가 정보를 표시합니다.

다음 예제에서는 구성된 리소스 3개가 있습니다. **C, D** 및 **E.**

```
# 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
```

다음 예제에서는 구성된 리소스 2개가 있습니다. A 및 B. 리소스 A 및 B는 리소스 그룹 G의 일부입니다.

```
# 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
```

16장. 규칙을 사용하여 리소스 위치 확인

더 복잡한 위치 제약 조건은 Pacemaker 규칙을 사용하여 리소스의 위치를 확인할 수 있습니다.

16.1. PACEMAKER 규칙

Pacemaker 규칙을 사용하여 구성을 보다 동적으로 만들 수 있습니다. 규칙 사용 중 하나는 시간을 기반으로 다양한 처리 그룹(노드 특성 사용)에 머신을 할당한 다음 위치 제약 조건을 생성할 때 해당 특성을 사용하는 것입니다.

각 규칙에는 여러 표현식, date-expression 및 기타 규칙이 포함될 수 있습니다. 표현식의 결과는 규칙의 부울-op 필드를 기반으로 결합하여 규칙이 최종적으로 true 또는 false 로 평가되는지 확인합니다. 다음은 규칙이 사용되는 컨텍스트에 따라 달라집니다.

표 16.1. 규칙의 속성

필드	설명
role	리소스가 해당 역할에 있을 때만 적용할 규칙을 제한합니다. 허용되는 값: **시작**,**Unpromoted 및 Promoted**. 알림: **role="Promoted"** 인 규칙은 복제 인스턴스의 초기 위치를 확인할 수 없습니다. 승격할 활성 인스턴스에만 영향을 미칩니다.
점수	규칙이 **true** 로 평가되는 경우 적용할 점수입니다. 위치 제약 조건의 일부인 규칙에서 사용하도록 제한됩니다.
score-attribute	규칙이 **true** 로 평가되면 조회하고 점수로 사용할 노드 속성입니다. 위치 제약 조건의 일부인 규칙에서 사용하도록 제한됩니다.
boolean-op	여러 표현식 오브젝트의 결과를 결합하는 방법 허용되는 값: **및 또는**. 기본값은 **및** 입니다.

16.1.1. 노드 특성 표현식

노드 특성 표현식은 노드 또는 노드에서 정의한 특성을 기반으로 리소스를 제어하는 데 사용됩니다.

표 16.2. Expression의 속성

필드	설명
attribute	테스트할 노드 속성입니다.

필드	설명
type	값을 테스트해야 하는 방법을 결정합니다. 허용되는 값은 **string**,**integer**,**number**,**version** 입니다. 기본값은 **문자열** 입니다.

필드	설명
operation	수행할 비교입니다. 허용되는 값: * **lt - 노드 특성의 값이 값보다 작으면 True** * **gt - 노드 특성 값이값보다 큰 경우 True** * **Long - 노드 특성의 값이 value보다 작거나 같은 경우 True** * **GT E - 노드 특성의 값이 value보다 크거나 같은 경우 True** * **EQ - 노드 특성의 값이 값과 같은 경우 True** * **NE - 노드 속성의 값이 값과 같지 않으면 True** * **정의된 - 노드에 이름이 지정된 속성이 있는 경우 True** * **not_defined - 노드에 named 속성이 없는 경우 True**

필드	설명
value	비교를 위해 사용자가 제공한 값 (작업이 **정의** 되거나 **not _defined**이 아닌 경우 필수)

관리자가 추가한 속성 외에도 클러스터는 다음 표에 설명된 대로 사용할 수 있는 각 노드의 특수 기본 제공 노드 속성을 정의합니다.

표 16.3. 기본 제공 노드 속성

이름	설명
#uname	노드 이름
#id	노드 ID
#kind	노드 유형. 가능한 값은 **cluster**,**remote**, **container** 입니다. **kind** 값은 **ocf:pacemaker:remote** 리소스 및 Pacemaker **원격** 게스트 노드 및 번들 노드에 대한 **컨테이너로** 생성된 Pacemaker 원격 노드에 대해 원격 입니다.
#is_dc	**true** 이 노드가 Designated Controller (DC)인 경우 true이고, 그렇지 않으면 **false**
#cluster_name	설정된 경우 **cluster-name** 클러스터 속성의 값
#site_name	설정된 경우 **site-name** 노드 속성의 값(예:**#cluster-name**)
#role	관련 승격 가능한 복제본이 이 노드에 있는 역할입니다. 승격 가능한 복제에 대한 위치 제약 조건에 대한 규칙 내에서만 유효합니다.

16.1.2. 시간/시간 기반 표현식

날짜 표현식은 현재 날짜/시간에 따라 리소스 또는 클러스터 옵션을 제어하는 데 사용됩니다. 선택적 날짜 사양을 포함할 수 있습니다.

표 16.4. 날짜 표현식의 속성

필드	설명
start	ISO8601 사양을 준수하는 날짜/시간입니다.

필드	설명
end	ISO8601 사양을 준수하는 날짜/시간입니다.
operation	상황에 따라 시작 또는 종료 날짜 또는 시작 날짜 및 종료 날짜와 현재 날짜를 비교합니다.Compares the current date/time with the start or the end date or both the start and end date, depending on the context. 허용되는 값: * **gt** - 현재 날짜/시간이 **시작된**후인 경우 True * **lt - 현재 날짜/시간이 종료되기 전인 경우 True** *** in_range - 현재 날짜/시간이 시작 후와 종료전인 경우 True** *** date-spec - 현재 날짜/시간에 대한 cron 과 유사한 비교 수행**

16.1.3. 날짜 사양

*날짜 사양은 시간과 관련된 **cron** 유사 표현식을 생성하는 데 사용됩니다. 각 필드에는 단일 숫자 또는 단일 범위가 포함될 수 있습니다. 기본값을 **0**으로 설정하는 대신 입력되지 않은 필드는 무시됩니다.*

*예를 들어, **monthdays="1"** 은 매달 첫 날과 일치하며 **hours="09-17"** 은 오전 **9**시부터 오후 **5**시(포함) 사이의 시간과 일치합니다. 그러나 여러 범위가 포함되어 있으므로 **weekdays="1,2"** 또는 **weekdays="1-2,5-6"** 을 지정할 수 없습니다.*

*표 **16.5.** 날짜 사양의 속성*

필드	설명
id	날짜의 고유 이름입니다.
시간	허용되는 값: 0-23

필드	설명
월days	허용되는 값: 0-31 (월과 연도에 따라)
weekdays	허용되는 값: 1-7 (1=Monday, 7=Sunday)
yeardays	허용되는 값: 1-366 (올해에 따라 다름)
months	허용되는 값: 1-12
weeks	허용되는 값: 1-53 (**주간**에 따라 다름)
years	Gregorian calendar에 따른 년
weekyears	예를 들어, **2005-001Ornal**은 **2005 -01-01 Gregorian** 또한 **2004-W53-6 Weekly**와 다를 수 있습니다.
달	허용되는 값: 0-7 (0은 새로운, 4는 전체 달)입니다.

16.2. 규칙을 사용하여 PACEMAKER 위치 제약 조건 구성

*다음 명령을 사용하여 규칙을 사용하는 **Pacemaker** 제약 조건을 구성합니다. **score** 가 생략된 경우 기본값은 **INFINITY**입니다. **resource-discovery** 를 생략하면 기본값은 **always** 입니다.*

***resource-discovery** 옵션에 대한 자세한 내용은 노드의 하위 집합으로 리소스 검색 제한을 참조하십시오.*

기본 위치 제약 조건과 마찬가지로 이러한 제약 조건이 있는 리소스에도 정규식을 사용할 수 있습니다.

*규칙을 사용하여 위치 제약 조건을 구성할 때 점수 값은 양수 또는 음수일 수 있으며, **"prefers"**를 나타내는 양수 값과 **"avoids"**를 나타내는 음수 값을 사용할 수 있습니다.*

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

*표현식 옵션은 **duration_options** 및 **date_spec_options** 가 시간, 월**days, weekdays, yeardays, months, weeks, years, weeks, weekyears** 및 월 테이블에 있는 경우 다음 중 하나일 수 있습니다.*
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_and_managing_high_availability_clusters/assem

bly_determining-resource-location-with-rules-configuring-and-managing-high-availability-clusters#date_specifications

- *defined|not_defined attribute*
- *attribute lt|gt|lte|gte|eq|ne [string|integer|number|version] value*
- *날짜 gt|lt date*
- *date in_range date to date*
- *date in_range date to duration_options …*
- *date-spec date_spec_options*
- *expression 및|or 표현식*
- *(expression)*

기간은 계산을 통해 in_range 작업의 끝을 지정하는 대체 방법입니다. 예를 들어 기간을 19개월로 지정할 수 있습니다.

다음 위치 제한 조건은 지금이 2018 년 중 임의의 시간인 경우 true를 설정하는 식을 구성합니다.

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

다음 명령은 월요일부터 금요일까지 오전 9시부터 오후 5시까지 실제 표현식을 구성합니다. 시간 16의 시간 값은 최대 16:59:59와 일치합니다. 숫자 값(시간)은 계속 일치합니다.

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

*다음 명령은 금요일에 달이 있을 때 **true**인 표현식을 구성합니다.*

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

규칙을 제거하려면 다음 명령을 사용합니다. 제거하는 규칙이 제약 조건의 마지막 규칙이면 제약 조건이 제거됩니다.

```
pcs constraint rule remove rule_id
```

17장. 클러스터 리소스 관리

클러스터 리소스를 표시, 수정 및 관리하는 데 사용할 수 있는 다양한 명령이 있습니다.

17.1. 구성된 리소스 표시

구성된 모든 리소스 목록을 표시하려면 다음 명령을 사용합니다.

```
pcs resource status
```

예를 들어, 시스템이 VirtualIP 라는 리소스와 Web-172.25.250이라는 리소스로 구성된 경우 **pcs resource status** 명령은 다음 출력을 생성합니다.

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

리소스에 대해 구성된 매개 변수를 표시하려면 다음 명령을 사용합니다.

```
pcs resource config resource_id
```

예를 들어 다음 명령은 리소스 VirtualIP 에 대해 현재 구성된 매개변수를 표시합니다.

```
# 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
```

개별 리소스의 상태를 표시하려면 다음 명령을 사용합니다.

```
pcs resource status resource_id
```

예를 들어, 시스템이 VirtualIP 라는 리소스로 구성된 경우 **pcs** 리소스 상태 VirtualIP 명령으로 다음 출력이 생성됩니다.

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

특정 노드에서 실행 중인 리소스의 상태를 표시하려면 다음 명령을 사용합니다. 이 명령을 사용하여 클러스터 및 원격 노드 모두에 리소스 상태를 표시할 수 있습니다.

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

예를 들어 node-01 이 VirtualIP 라는 리소스를 실행하고 있는 경우 pcs resource status node=node-01 명령으로 다음 출력이 표시될 수 있습니다.

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

17.2. PCS 명령으로 클러스터 리소스 내보내기

Red Hat Enterprise Linux 9.1에서는 pcs 리소스 구성 명령의 --output-format=cmd 옵션을 사용하여 다른 시스템에 구성된 클러스터 리소스를 다시 만드는 데 사용할 수 있는 pcs 명령을 표시할 수 있습니다.

다음 명령은 Red Hat 고가용성 클러스터에서 활성/수동형 Apache HTTP 서버에 대해 생성된 4개의 리소스(LVM 활성화 리소스, 파일 시스템 리소스, IPaddr2 리소스, Apache 리소스)를 생성합니다.

```
# 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
```

리소스를 생성한 후 다음 명령은 다른 시스템에서 해당 리소스를 다시 만드는 데 사용할 수 있는 pcs 명령을 표시합니다.

```
# 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 명령 또는 명령을 표시하려면 구성된 리소스만 다시 만드는 데 사용할 수 있습니다. 해당 리소스의 리소스 ID를 지정합니다.

```
# 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
```

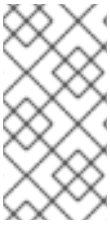
17.3. 리소스 매개변수 수정

구성된 리소스의 매개 변수를 수정하려면 다음 명령을 사용합니다.

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

다음 명령 시퀀스는 리소스 VirtualIP 에 대해 구성된 매개변수의 초기 값을 보여주고, 명령은 ip 매개 변수의 값을 변경하는 명령, update 명령 이후의 값을 표시합니다.

```
# 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
```



참고

pcs resource update 명령을 사용하여 리소스 작업을 업데이트할 때 특별히 호출하지 않는 옵션은 기본값으로 재설정됩니다.

17.4. 클러스터 리소스의 장애 상태 삭제

리소스가 실패하면 **pcs status** 명령을 사용하여 클러스터 상태를 표시할 때 실패 메시지가 표시됩니다. 실패의 원인을 해결한 후 **pcs status** 명령을 다시 실행하여 리소스의 업데이트된 상태를 확인할 수 있으며 **pcs resource failcount show --full** 명령을 사용하여 클러스터 리소스의 실패 수를 확인할 수 있습니다.

pcs resource cleanup 명령을 사용하여 리소스의 실패 상태를 지울 수 있습니다. **pcs resource cleanup** 명령은 리소스의 리소스 상태 및 **failcount** 값을 재설정합니다. 이 명령은 또한 리소스의 작업 기록을 제거하고 현재 상태를 다시 감지합니다.

다음 명령은 **resource_id** 에서 지정한 리소스의 리소스 상태 및 **failcount** 값을 재설정합니다.

```
pcs resource cleanup resource_id
```

resource_id 를 지정하지 않으면 **pcs resource cleanup** 명령에서 실패 횟수가 있는 모든 리소스의 리소스 상태 및 **failcount** 값을 재설정합니다.

pcs resource cleanup resource_id 명령 외에도 리소스 상태를 재설정하고 **pcs resource refresh** **resource refresh resource_id** 명령을 사용하여 리소스의 작업 기록을 지울 수도 있습니다. **pcs resource cleanup** 명령과 마찬가지로 지정된 옵션 없이 **pcs resource refresh** 명령을 실행하여 모든 리소스의 리소스 상태 및 **failcount** 값을 재설정할 수 있습니다.

pcs resource cleanup 및 **pcs resource refresh** 명령은 모두 리소스의 작업 기록을 지우고 리소스의 현재 상태를 다시 감지합니다. **pcs resource cleanup** 명령은 클러스터 상태에 표시된 것처럼 실패한 작업으로만 작동하는 반면 **pcs resource refresh** 명령은 현재 상태와 관계없이 리소스에서 작동합니다.

17.5. 클러스터에서 리소스 이동

Pacemaker는 필요한 경우 한 노드에서 다른 노드로 이동하고 리소스를 수동으로 이동할 수 있도록 리소스를 구성하는 다양한 메커니즘을 제공합니다.

수동으로 클러스터 리소스 이동에 설명된 대로 **pcs resource move** 및 **pcs resource relocate** 명령을

사용하여 클러스터에서 리소스를 수동으로 이동할 수 있습니다. 이러한 명령 외에도 Disabling, 활성화 및 금지에 설명된 대로 리소스를 활성화, 비활성화, 금지하여 클러스터 리소스의 동작을 제어할 수도 있습니다.

정의된 수의 실패 후 새 노드로 이동할 수 있도록 리소스를 구성하고 외부 연결이 손실될 때 리소스를 구성하여 리소스를 구성할 수 있습니다.

17.5.1. 실패로 인한 리소스 이동

리소스를 생성할 때 해당 리소스에 대한 **migration-threshold** 옵션을 설정하여 정의된 수의 실패 후 새 노드로 이동하도록 리소스를 구성할 수 있습니다. 임계값에 도달한 후 이 노드는 다음까지 실패한 리소스를 더 이상 실행할 수 없습니다.

- 리소스의 **failure-timeout** 값에 도달합니다.
- 관리자는 **pcs resource cleanup** 명령을 사용하여 리소스 실패 횟수를 수동으로 재설정합니다.

migration-threshold 값은 기본적으로 **INFINITY** 로 설정됩니다. **INFINITY** 는 내부적으로 매우 크고 한정된 숫자로 정의됩니다. 값이 **0**이면 **migration-threshold** 기능을 비활성화합니다.

참고

리소스에 대한 **migration-threshold** 는 마이그레이션용 리소스를 구성하는 것과 같지 않으며, 리소스가 상태 손실 없이 다른 위치로 이동합니다.

다음 예제는 **dummy_resource** 라는 리소스에 **10**의 마이그레이션 임계값을 추가합니다. 이는 리소스가 **10**개의 실패 후 리소스가 새 노드로 이동됨을 나타냅니다.

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

다음 명령을 사용하여 전체 클러스터의 기본값에 마이그레이션 임계값을 추가할 수 있습니다.

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

*리소스의 현재 실패 상태 및 제한을 확인하려면 **pcs resource failcount show** 명령을 사용합니다.*

*마이그레이션 임계값 개념에는 두 가지 예외가 있습니다. 리소스가 시작되지 않거나 중지에 실패할 때 발생합니다. 클러스터 속성 **start-failure-is-fatal** 이 **true** (기본값)로 설정된 경우 시작 실패로 인해 **failcount** 가 **INFINITY** 로 설정되고 항상 리소스가 즉시 이동합니다.*

*중단 실패는 약간 다르며 중요합니다. 리소스가 중지되지 않고 **STONITH**가 활성화된 경우 클러스터는 노드를 펜싱하여 다른 위치에서 리소스를 시작할 수 있습니다. **STONITH**가 활성화되지 않은 경우 클러스터는 계속 사용할 수 없으며 리소스를 다른 위치에서 시작하려고하지 않지만 오류 시간 초과 후 다시 중지하려고 시도합니다.*

17.5.2. 연결 변경으로 인한 리소스 이동

외부 연결이 손실되면 리소스를 이동하도록 클러스터를 설정하는 작업은 두 단계로 진행됩니다.

1. *클러스터에 **ping** 리소스를 추가합니다. **ping** 리소스는 동일한 이름의 시스템 유틸리티를 사용하여 시스템 목록(**DNS** 호스트 이름 또는 **IPv4**/**IPv6** 주소로 지정)에 연결할 수 있는지 테스트하고 결과를 사용하여 **pingd** 라는 노드 특성을 유지 관리합니다.*
2. *연결이 손실될 때 리소스를 다른 노드로 이동할 리소스에 대한 위치 제한 조건을 구성합니다.*

*다음 표에서는 **ping** 리소스에 대해 설정할 수 있는 속성을 설명합니다.*

*표 **17.1. ping** 리소스의 속성*

필드	설명
dampen	추가 변경 사항이 발생할 때까지 대기하는 시간(ampening)입니다. 이렇게 하면 클러스터 노드가 약간 다른 시간에 연결 손실을 알 때 클러스터 관련 리소스가 표시되지 않습니다.
multiplier	연결된 ping 노드 수에 이 값이 곱하여 점수를 얻습니다. 여러 ping 노드가 구성된 경우 유용합니다.
host_list	현재 연결 상태를 확인하기 위해 연결할 머신입니다. 허용되는 값에는 확인 가능한 DNS 호스트 이름, IPv4 및 IPv6 주소가 포함됩니다. 호스트 목록의 항목은 공백으로 구분됩니다.

다음 예제 명령은 **gateway.example.com** 에 대한 연결을 확인하는 **ping** 리소스를 생성합니다. 실제로는 네트워크 게이트웨이/라우터에 대한 연결을 확인합니다. 모든 클러스터 노드에서 리소스가 실행되도록 **ping** 리소스를 복제본으로 구성합니다.

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

다음 예제에서는 **Webserver** 라는 기존 리소스에 대한 위치 제한 조건 규칙을 구성합니다. 이렇게 하면 현재 실행 중인 호스트가 **gateway.example.com** 을 **ping**할 수 없는 경우 **Webserver** 리소스가 **ping.example .com**에서 **ping**할 수 있는 호스트로 이동합니다.

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

17.6. 모니터 작업 비활성화

반복 모니터를 중지하는 가장 쉬운 방법은 이를 삭제하는 것입니다. 그러나 일시적으로 비활성화하려는 경우가 있을 수 있습니다. 이 경우 작업 정의에 **enabled="false"** 를 추가합니다. 모니터링 작업을 다시 얻으려면 **enabled="true"** 를 작업 정의로 설정합니다.

pcs resource update 명령을 사용하여 리소스 작업을 업데이트할 때 특별히 호출하지 않는 옵션은 기본값으로 재설정됩니다. 예를 들어 사용자 정의 시간 제한 값 **600**으로 모니터링 작업을 구성한 경우 다음 명령을 실행하면 시간제한 값을 기본값인 **20**(또는 기본값을 **pcs resource op defaults** 명령을 사용하여 설정함)으로 재설정합니다.

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

이 옵션에 대해 원래 값을 **600**으로 유지하려면 모니터링 작업을 다시 시작할 때 다음 예제와 같이 해당 값을 지정해야 합니다.

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

17.7. 클러스터 리소스 태그 구성 및 관리

pcs 명령을 사용하여 클러스터 리소스에 태그를 지정할 수 있습니다. 이를 통해 단일 명령으로 지정된 리소스 세트를 활성화, 비활성화, 관리 또는 관리할 수 있습니다.

17.7.1. 카테고리별 관리를 위해 클러스터 리소스 태그

다음 절차에서는 리소스 태그로 두 개의 리소스에 태그를 지정하고 태그된 리소스를 비활성화합니다. 이 예에서 태그를 지정할 기존 리소스의 이름은 **d-01** 및 **d-02** 입니다.

절차

1. 리소스 **d-01** 및 **d-02** 에 대한 **special-resources** 라는 태그를 생성합니다.

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

2. 리소스 태그 구성을 표시합니다.

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

3. **special-resources** 태그가 지정된 모든 리소스를 비활성화합니다.

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

4. 리소스의 상태를 표시하여 **d-01** 및 **d-02** 리소스가 비활성화되어 있는지 확인합니다.

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

pcs resource disable 명령 외에도 **pcs** 리소스를 사용하면,**pcs resource manage, pcs resource unmanage** 명령이 태그된 리소스의 관리를 지원합니다.

리소스 태그를 생성한 후 다음을 수행합니다.

- **pcs tag delete** 명령을 사용하여 리소스 태그를 삭제할 수 있습니다.
- **pcs tag update** 명령을 사용하여 기존 리소스 태그의 리소스 태그 구성을 수정할 수 있습니다.

17.7.2. 태그가 지정된 클러스터 리소스 삭제

pcs 명령으로 태그가 지정된 클러스터 리소스를 삭제할 수 없습니다. 태그된 리소스를 삭제하려면 다음 절차를 사용하십시오.

절차

1. 리소스 태그를 제거합니다.

   a. 다음 명령은 해당 태그가 있는 모든 리소스에서 **resource tag special-resources** 를 제거합니다.

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

   b. 다음 명령은 리소스 **d-01** 에서만 리소스 태그 **special-resources** 를 제거합니다.

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

2. 리소스를 삭제합니다.

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

18장. 여러 노드에서 활성 상태인 클러스터 리소스 생성 (복제된 리소스)

여러 노드에서 리소스를 활성화할 수 있도록 클러스터 리소스를 복제할 수 있습니다. 예를 들어 복제된 리소스를 사용하여 노드 밸런싱을 위해 클러스터 전체에서 배포할 **IP** 리소스의 여러 인스턴스를 구성할 수 있습니다. 리소스 에이전트가 지원하는 모든 리소스를 복제할 수 있습니다. 복제본은 하나의 리소스 또는 리소스 그룹으로 구성됩니다.

참고

여러 노드에서 동시에 활성화할 수 있는 리소스만 복제에 적합합니다. 예를 들어 공유 메모리 장치에서 **ext4** 와 같은 클러스터되지 않은 파일 시스템을 마운트하는 **Filesystem** 리소스는 복제해서는 안 됩니다. **ext4** 파티션은 클러스터를 인식하지 않으므로 이 파일 시스템은 여러 노드에서 동시에 발생하는 읽기/쓰기 작업에 적합하지 않습니다.

18.1. 복제된 리소스 생성 및 제거

리소스 및 해당 리소스의 복제본을 동시에 생성할 수 있습니다.

다음 단일 명령으로 리소스 및 리소스 복제를 생성하려면 다음을 수행합니다.

```
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]
```

기본적으로 복제 이름은 **resource_id-clone** 입니다. **clone_id** 옵션의 값을 지정하여 복제의 사용자 지정 이름을 설정할 수 있습니다.

단일 명령에서 리소스 그룹과 해당 리소스 그룹의 복제본을 생성할 수 없습니다.

또는 다음 명령을 사용하여 이전에 생성한 리소스 또는 리소스 그룹의 복제본을 만들 수 있습니다.

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

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

기본적으로 복제 이름은 **resource_id-clone** 또는 **group_name-clone** 입니다. **clone_id** 옵션의 값을

지정하여 복제의 사용자 지정 이름을 설정할 수 있습니다.

참고

하나의 노드에서만 리소스 구성 변경 사항을 구성해야 합니다.

참고

제약 조건을 구성할 때 항상 그룹 또는 복제의 이름을 사용합니다.

*리소스 복제본을 생성할 때 기본적으로 복제본은 이름에 **-clone** 이 추가된 리소스 이름에 사용됩니다. 다음 명령은 **webfarm**이라는 이름의 **apache** 유형 리소스와 **webfarm -clone** 이라는 해당 리소스의 복제본을 생성합니다.*

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

참고

*다른 복제본 이후에 주문되는 리소스 또는 리소스 그룹 복제를 생성할 때 거의 항상 **interleave=true** 옵션을 설정해야 합니다. 이렇게 하면 종속 복제본의 복사본이 동일한 노드에서 종속된 복제가 중지되거나 시작될 때 중지되거나 시작할 수 있습니다. 이 옵션을 설정하지 않으면 복제된 리소스 **B**에 따라 복제된 리소스 **A**와 노드가 클러스터로 돌아가고 리소스 **A**가 해당 노드에서 시작되면 모든 노드에서 리소스 **B** 복사본이 다시 시작됩니다. 종속 복제된 리소스에 **interleave** 옵션이 설정되지 않은 경우 해당 리소스의 모든 인스턴스에서 종속된 리소스의 실행 중인 인스턴스에 종속되기 때문입니다.*

다음 명령을 사용하여 리소스 또는 리소스 그룹의 복제본을 제거합니다. 리소스 또는 리소스 그룹 자체를 제거하지는 않습니다.

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

다음 표에서는 복제된 리소스에 지정할 수 있는 옵션을 설명합니다.

*표 **18.1.** 리소스 복제 옵션*

필드	설명

필드	설명
priority, target-role, is-managed	리소스 메타 옵션 구성의 "리소스 메타 옵션" 테이블에 설명된 대로 복제 중인 리소스에서 상속되는 옵션입니다.
clone-max	시작할 리소스의 사본 수입니다. 기본값은 클러스터의 노드 수입니다.
clone-node-max	단일 노드에서 시작할 수 있는 리소스 사본 수입니다. 기본값은 **1** 입니다.
알림	복제 복사본을 중지하거나 시작할 때 사전에 모든 다른 복사본과 작업이 성공적으로 수행되었는지 알려줍니다. 허용되는 값: **false**,**true**. 기본값은 **false**입니다.
globally-unique	복제본의 각 사본이 다른 기능을 수행합니까? 허용되는 값: **false**,**true** 이 옵션의 값이 **false** 이면 이러한 리소스는 실행 중인 모든 위치에서 동일하게 동작하므로 머신당 활성 복제의 복사본은 하나만 있을 수 있습니다. 이 옵션의 값이 **true** 인 경우 한 시스템에서 실행 중인 복제본의 복사본이 다른 인스턴스에서 실행되지 않는 경우, 해당 인스턴스가 다른 노드에서 실행 중인지 또는 동일한 노드에서 실행 중인지와 동일합니다. **clone-node-max** 값이 둘보다 크면 기본값은 **true** 이고, 그렇지 않으면 기본값은 **false** 입니다.
ordered	복사본이 시리즈로 시작되어야 하는 경우(parallel으로 대신함). 허용되는 값: **false**,**true**. 기본값은 **false**입니다.
interleave	첫 번째 복제본의 복사본이 두 번째 복제본의 동일한 노드에서 복사가 시작되거나 중지되는 즉시 정렬 제약 조건(복제본의 모든 인스턴스가 시작 또는 중지될 때까지 대기함)이 시작되거나 중지될 수 있도록 합니다. 허용되는 값: **false**,**true**. 기본값은 **false**입니다.
clone-min	값을 지정하면 이 복제 후 정렬된 모든 복제본은 **interleave** 옵션이 **true** 로 설정된 경우에도 원래 복제본의 지정된 수의 인스턴스가 실행될 때까지 시작할 수 없습니다.

안정적인 할당 패턴을 달성하기 위해 복제본은 기본적으로 약간 고정되므로 실행 중인 노드에 그대로 유지됩니다. resource-stickiness 값이 제공되지 않으면 복제본에서 1 값을 사용합니다. 값이 작기 때문에 다른 리소스의 점수 계산이 최소화되지만 Pacemaker가 클러스터 주변의 복사본을 쉽게 이동할 수 있는 것으로 충분합니다. resource-stickiness 리소스 meta-option 설정에 대한 자세한 내용은 리소스 메타 옵션 구성을 참조하십시오.

18.2. 복제 리소스 제약 조건 구성

대부분의 경우 복제본은 각 활성 클러스터 노드에 하나의 사본을 갖습니다. 그러나 리소스 복제본의 **clone-max** 를 클러스터의 총 노드 수보다 작은 값으로 설정할 수 있습니다. 이 경우 클러스터가 리소스 위치 제약 조건을 사용하여 복사본을 우선적으로 할당해야 하는 노드를 나타낼 수 있습니다. 이러한 제약 조건은 복제의 **ID**를 사용해야 한다는 점을 제외하고는 일반 리소스에 대해 다르게 기록되지 않습니다.

다음 명령은 클러스터의 위치 제한 조건을 생성하여 리소스 복제 **webfarm-clone** 을 **node1** 에 우선적으로 할당합니다.

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

복제의 순서 지정 제한 사항은 약간 다르게 작동합니다. 아래 예제에서 **interleave clone** 옵션은 기본값으로 **false** 로 남아 있으므로 시작해야 하는 **webfarm-clone** 의 모든 인스턴스가 완료될 때까지 **webfarm-stats** 인스턴스가 시작되지 않습니다. **webfarm-clone** 을 시작할 수 없는 경우에만 **webfarm-stats** 가 활성화되지 않습니다. 또한 **webfarm-clone** 은 자체적으로 중지하기 전에 **webfarm-stats** 가 중지될 때까지 기다립니다.

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

복제본과 함께 일반(또는 그룹) 리소스를 공동 배치하면 복제본이 활성 상태인 모든 시스템에서 리소스를 실행할 수 있습니다. 클러스터는 복제본이 실행 중인 위치와 리소스의 자체 위치 기본 설정에 따라 사본을 선택합니다.

또한 클론 간의 공동 배치가 가능합니다. 이러한 경우 복제본에 허용되는 위치 세트는 복제가 활성 상태인 노드로 제한됩니다. 그런 다음 할당은 정상적으로 수행됩니다.

다음 명령은 리소스 **webfarm-stats** 가 **webfarm-clone** 의 활성 사본과 동일한 노드에서 실행되도록 공동 배치 제한 조건을 생성합니다.

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

18.3. 승격 가능한 복제 리소스

승격 가능한 복제 리소스는 **promotable** 메타 특성이 **true** 로 설정된 복제 리소스입니다. 두 가지 운영 모드 중 하나에 인스턴스를 사용할 수 있습니다. 이러한 모드를 승격 하고 프로모션되지 않습니다. 모드의 이름에는 인스턴스가 시작될 때 **Unpromoted** 상태가 되어야 한다는 제한 사항을 제외하고 특정 의미가 없습니다. 참고: **Promoted** 및 **Unpromoted** 역할 이름은 이전 **RHEL** 릴리스에서 **Master(**마스터**)** 및 **Slave Pacemaker** 역할과 동일한 기능을 수행합니다.

18.3.1. 승격 가능한 복제 리소스 생성

다음 단일 명령을 사용하여 승격 가능한 복제로 리소스를 생성할 수 있습니다.

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

기본적으로 **promotable** 복제의 이름은 **resource_id-clone** 입니다.

clone_id 옵션의 값을 지정하여 복제의 사용자 지정 이름을 설정할 수 있습니다.

또는 다음 명령을 사용하여 이전에 생성한 리소스 또는 리소스 그룹에서 승격 테이블 리소스를 생성할 수 있습니다.

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

기본적으로 **promotable** 복제의 이름은 **resource_id-clone** 또는 **group_name-clone** 입니다.

clone_id 옵션의 값을 지정하여 복제의 사용자 지정 이름을 설정할 수 있습니다.

다음 표에서는 승격 가능한 리소스에 지정할 수 있는 추가 복제 옵션을 설명합니다.

표 **18.2. Promotable Clones**에 사용 가능한 추가 복제 옵션

필드	설명
promoted-max	승격할 수 있는 리소스 사본 수입니다. 기본 1.
promoted-node-max	단일 노드에서 승격할 수 있는 리소스 사본 수입니다. 기본 1.

18.3.2. 승격 테이블 리소스 제약 조건 구성

대부분의 경우 승격 가능한 리소스에는 각 활성 클러스터 노드에 하나의 사본이 있습니다. 그렇지 않은 경우 클러스터가 리소스 위치 제약 조건을 사용하여 복사본을 우선적으로 할당해야 하는 노드를 나타낼 수 있습니다. 이러한 제약 조건은 일반 리소스의 경우와 달리 기록됩니다.

리소스가 승격되거나 승격되지 않은 역할에서 작동하는지를 지정하는 공동 배치 제한 조건을 생성할 수 있습니다. 다음 명령은 리소스 공동 배치 제약 조건을 생성합니다.

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

공동 배치 제약 조건에 대한 자세한 내용은 클러스터 리소스 배치를 참조하십시오.

승격 가능한 리소스가 포함된 순서 지정 제한 조건을 구성할 때 리소스에 지정할 수 있는 작업 중 하나가 승격 되므로 리소스가 승격되지 않은 역할에서 승격된 역할로 승격됩니다. 또한 **demote** 의 작업을 지정할 수 있으며, 리소스는 승격된 역할에서 프로모션되지 않은 역할로 강등되는 것으로 표시되었습니다.

순서 제한 조건을 구성하는 명령은 다음과 같습니다.

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

리소스 순서 제약 조건에 대한 자세한 내용은 클러스터 리소스가 실행되는 순서 결정을 참조하십시오.

18.4. 실패 시 승격된 리소스 데모

승격 테이블 리소스를 구성하여 해당 리소스에 대한 승격 또는 모니터 작업이 실패하거나 리소스가 쿼럼을 실행하는 파티션이 손실되면 리소스가 시연되지만 완전히 중지되지는 않습니다. 이렇게 하면 리소스를 완전히 중지해야 하는 상황에서 수동 개입이 필요하지 않을 수 있습니다.

- 승격 작업이 실패할 때 승격 가능한 리소스를 시연하려면 다음 예제와 같이 **on-fail** 작업 **meta** 옵션을 **demote** 로 설정합니다.

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

- 모니터 작업이 실패할 때 강등하도록 승격 가능한 리소스를 구성하려면 간격 을 **0**이 아닌 값으로 설정하고, **on-fail** 작업 메타 옵션을 **demote** 로 설정하고 역할을 **Promoted** 로 설정합니다.

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

- 클러스터 파티션이 쿼럼을 손실하지만 계속 실행 중이 고 다른 모든 리소스가 중지되도록 클러스터를 구성하려면 **no-quorum-policy** 클러스터 속성을 **demote**로 설정합니다.

***on-fail meta-attribute*를 작업에 대한 *demote* 로 설정하면 리소스 승격이 결정되는 방식에 영향을 미치지 않습니다. 영향을 받는 노드에 계속 가장 높은 승격 점수가 있는 경우 다시 승격하도록 선택됩니다.**

19장. 클러스터 노드 관리

클러스터 서비스를 시작 및 중지하고 클러스터 노드를 추가 및 제거하는 명령을 포함하여 클러스터 노드를 관리하는 데 사용할 수 있는 다양한 **pcs** 명령이 있습니다.

19.1. 클러스터 서비스 중지

다음 명령은 지정된 노드 또는 노드에서 클러스터 서비스를 중지합니다. **pcs cluster start** 와 마찬가지로 **--all** 옵션은 모든 노드에서 클러스터 서비스를 중지하고 노드를 지정하지 않으면 로컬 노드에서만 클러스터 서비스가 중지됩니다.

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

다음 명령을 사용하여 로컬 노드에서 클러스터 서비스 중지를 강제 실행하여 **kill -9** 명령을 수행할 수 있습니다.

```
pcs cluster kill
```

19.2. 클러스터 서비스 활성화 및 비활성화

다음 명령을 사용하여 클러스터 서비스를 활성화합니다. 이렇게 하면 지정된 노드 또는 노드에서 시작 시 클러스터 서비스가 실행되도록 구성됩니다.

활성화하면 노드가 펜싱된 후 클러스터에 자동으로 다시 참여할 수 있으므로 클러스터가 전체 강점보다 적은 시간을 최소화할 수 있습니다. 클러스터 서비스가 활성화되어 있지 않은 경우 관리자는 클러스터 서비스를 수동으로 시작하기 전에 문제가 발생한 내용을 수동으로 조사할 수 있으므로, 예를 들어 에서 하드웨어 문제가 있는 노드는 다시 실패할 가능성이 있는 경우 클러스터로 다시 허용되지 않습니다.

- **--all** 옵션을 지정하면 명령은 모든 노드에서 클러스터 서비스를 활성화합니다.
- 노드를 지정하지 않으면 로컬 노드에서만 클러스터 서비스가 활성화됩니다.

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

다음 명령을 사용하여 지정된 노드 또는 노드에서 시작 시 실행되지 않도록 클러스터 서비스를 구성합니다.

- *--all 옵션을 지정하면 명령은 모든 노드에서 클러스터 서비스를 비활성화합니다.*

- *노드를 지정하지 않으면 로컬 노드에서만 클러스터 서비스가 비활성화됩니다.*

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

19.3. 클러스터 노드 추가

다음 절차에 따라 기존 클러스터에 새 노드를 추가합니다.

이 절차에서는 corosync 를 실행하는 표준 클러스터 노드가 추가되었습니다. 비corosync 노드를 클러스터에 통합하는 방법에 대한 자세한 내용은 non-corosync 노드를 클러스터로 통합(pacemaker_remote 서비스)을 참조하십시오.

참고

프로덕션 유지 관리 기간 동안에만 기존 클러스터에 노드를 추가하는 것이 좋습니다. 이를 통해 새 노드와 해당 펜싱 구성에 대해 적절한 리소스 및 배포 테스트를 수행할 수 있습니다.

이 예에서 기존 클러스터 노드는 clusternode-01.example.com,clusternode-02.example.com, clusternode-03.example.com 입니다. 새 노드는 newnode.example.com 입니다.

절차

클러스터에 추가할 새 노드에서 다음 작업을 수행합니다.

1. *클러스터 패키지를 설치합니다. 클러스터에서 SBD, Booth 티켓 관리자 또는 쿼럼 장치를 사용하는 경우 해당 패키지(sbd,booth-site,corosync-qdevice)를 새 노드에 수동으로 설치해야 합니다.*

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

클러스터 패키지 외에도 기존 클러스터 노드에 설치된 클러스터에서 실행 중인 모든 서비스를 설치하고 구성해야 합니다. 예를 들어 Red Hat 고가용성 클러스터에서 Apache HTTP 서버를 실

*행하는 경우, 추가하는 노드에 서버를 설치해야 하며 서버 상태를 확인하는 **wget** 툴을 설치해야 합니다.*

2. *firewalld 데몬을 실행하는 경우 다음 명령을 실행하여 **Red Hat High Availability Add-On**에 필요한 포트를 활성화합니다.*

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

3. *사용자 **ID hacluster** 의 암호를 설정합니다. 클러스터의 각 노드에 동일한 암호를 사용하는 것이 좋습니다.*

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

4. *다음 명령을 실행하여 **pcsd** 서비스를 시작하고 시스템 시작 시 **pcsd** 를 활성화합니다.*

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

기존 클러스터의 노드에서 다음 작업을 수행합니다.

1. *새 클러스터 노드에서 **hacluster** 사용자를 인증합니다.*

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

2. *기존 클러스터에 새 노드를 추가합니다. 이 명령은 추가하는 새 노드를 포함하여 클러스터 구성 파일 **corosync.conf** 를 클러스터의 모든 노드와 동기화합니다.*

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

클러스터에 추가할 새 노드에서 다음 작업을 수행합니다.

1. 새 노드에서 클러스터 서비스를 시작하고 활성화합니다.

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

2. 새 클러스터 노드의 펜싱 장치를 구성하고 테스트하십시오.

19.4. 클러스터 노드 제거

다음 명령은 지정된 노드를 종료하고 클러스터 구성 파일 **corosync.conf** 에서 클러스터의 다른 모든 노드에서 제거합니다.

```
pcs cluster node remove node
```

19.5. 여러 링크가 있는 클러스터에 노드 추가

여러 링크가 있는 클러스터에 노드를 추가할 때 모든 링크에 대한 주소를 지정해야 합니다.

다음 예제에서는 첫 번째 링크에 **IP** 주소 **192.168.122.203**을 지정하고 두 번째 링크로 **192.168.123.203**을 지정하여 **rh80-node3** 노드를 클러스터에 추가합니다.

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

19.6. 기존 클러스터에서 링크 추가 및 수정

대부분의 경우 클러스터를 다시 시작하지 않고 기존 클러스터에서 링크를 추가하거나 수정할 수 있습니다.

19.6.1. 기존 클러스터에서 링크 추가 및 제거

실행 중인 클러스터에 새 링크를 추가하려면 **pcs cluster link add** 명령을 사용합니다.

- 링크를 추가할 때 각 노드의 주소를 지정해야 합니다.

- 링크 추가 및 제거는 **knet** 전송 프로토콜을 사용하는 경우에만 가능합니다.
- 클러스터에 있는 하나 이상의 링크는 언제든지 정의해야 합니다.
- 클러스터의 최대 링크 수는 **8**, 번호 **0-7**입니다. 정의된 링크는 중요하지 않으므로 예를 들어 링크 **3**, **6** 및 **7**만 정의할 수 있습니다.
- 링크 번호를 지정하지 않고 링크를 추가하면 **pcs** 에서 사용 가능한 가장 낮은 링크를 사용합니다.
- 현재 구성된 링크의 링크 번호는 **corosync.conf** 파일에 포함되어 있습니다. **corosync.conf** 파일을 표시하려면 **pcs cluster corosync** 명령 또는 **pcs cluster config show** 명령을 실행합니다.

다음 명령은 링크 번호 **5**를 **3**개의 노드 클러스터에 추가합니다.

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

기존 링크를 제거하려면 **pcs cluster link delete** 또는 **pcs cluster link remove** 명령을 사용합니다. 다음 명령 중 하나가 클러스터에서 링크 번호 **5**를 제거합니다.

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

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

19.6.2. 여러 링크가 있는 클러스터에서 링크 수정

클러스터에 링크가 여러 개 있고 해당 링크 중 하나를 변경하려면 다음 절차를 수행하십시오.

절차

1. 변경할 링크를 제거합니다.

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

2. 업데이트된 주소 및 옵션을 사용하여 클러스터에 다시 링크를 추가합니다.

```
[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. 단일 링크를 사용하여 클러스터에서 링크 주소 수정

클러스터에서 하나의 링크만 사용하고 다른 주소를 사용하도록 해당 링크를 수정하려면 다음 절차를 수행합니다. 이 예에서 원래 링크는 링크 1입니다.

1. 새 주소 및 옵션을 사용하여 새 링크를 추가합니다.

```
[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. 원본 링크를 제거합니다.

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

클러스터에 링크를 추가할 때 현재 사용 중인 주소를 지정할 수 없습니다. 예를 들어 하나의 링크가 있는 2-노드 클러스터가 있고 한 노드의 주소만 변경하려면 위의 절차를 사용하여 하나의 새 주소와 하나의 기존 주소를 지정하는 새 링크를 추가할 수 없습니다. 대신 다음 예제와 같이 기존 링크를 제거하고 업데이트된 주소로 다시 추가할 수 있습니다.

이 예제에서는 다음을 수행합니다.

- 기존 클러스터의 링크는 노드 1에 대해 10.0.5.11 주소 및 노드 2의 경우 10.0.5.12 주소를 사용하는 링크 1입니다.
- 노드 2의 주소를 10.0.5.31로 변경하고 싶습니다.

절차

단일 링크를 사용하여 2-노드 클러스터의 주소 중 하나만 업데이트하려면 다음 절차를 사용하십시오.

1. 현재 사용하지 않는 주소를 사용하여 기존 클러스터에 새 임시 링크를 추가합니다.

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

2. 원본 링크를 제거합니다.

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

3. 새 링크, 수정된 링크를 추가합니다.

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

4. 생성한 임시 링크 삭제

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

19.6.4. 단일 링크를 사용하여 클러스터의 링크 옵션 수정

클러스터에서 링크를 하나만 사용하고 해당 링크의 옵션을 수정하려는 경우, 사용할 주소를 변경하지 않으려면 수정 링크를 제거하고 업데이트하기 전에 임시 링크를 추가할 수 있습니다.

이 예제에서는 다음을 수행합니다.

- 기존 클러스터의 링크는 노드 **1**에 대해 **10.0.5.11** 주소 및 노드 **2**의 경우 **10.0.5.12** 주소를 사용하는 링크 **1**입니다.
- **link** 옵션 **link_priority** 를 **11**로 변경하고 싶습니다.

절차

다음 절차에 따라 단일 링크를 사용하여 클러스터에서 **link** 옵션을 수정합니다.

1. 현재 사용하지 않는 주소를 사용하여 기존 클러스터에 새 임시 링크를 추가합니다.

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

2. 원본 링크를 제거합니다.

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

3. 업데이트된 옵션을 사용하여 원래 링크를 다시 추가합니다.

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

4. 임시 링크를 제거합니다.

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

19.6.5. 새 링크를 추가할 때 링크를 수정할 수 없습니다.

어떤 이유로 구성에서 새 링크를 추가할 수 없으며 유일한 옵션은 기존 단일 링크를 수정하는 것입니다.

절차

다음 예제 절차에서는 클러스터에서 링크 번호 1을 업데이트하고 링크의 **link_priority** 옵션을 **11**로 설정합니다.

1. 클러스터의 클러스터 서비스를 중지합니다.

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

2. 링크 주소 및 옵션을 업데이트합니다.

pcs cluster link update 명령은 모든 노드 주소와 옵션을 지정할 필요가 없습니다. 대신 변경할 주소만 지정할 수 있습니다. 이 예제에서는 **node1** 및 **node3** 의 주소를 수정하고 **link_priority** 옵션만 수정합니다.

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

옵션을 제거하려면 옵션을 **option=** 형식으로 **null** 값으로 설정할 수 있습니다.

3. 클러스터를 다시 시작

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

19.7. 노드 상태 전략 구성

노드는 클러스터 멤버십을 유지하기에 충분히 잘 작동할 수 있지만 일부 측면에서는 리소스의 바람직하지 않은 위치가 될 수 있습니다. 예를 들어 디스크 드라이브가 **SMART** 오류를 보고하거나 **CPU**가 고도로 로드될 수 있습니다. **RHEL 9.1**에서는 **Pacemaker**에서 노드 상태 전략을 사용하여 비정상 노드의 리소스를 자동으로 이동할 수 있습니다.

CPU 및 디스크 상태에 따라 노드 특성을 설정하는 다음 상태 노드 리소스 에이전트를 사용하여 노드의 상태를 모니터링할 수 있습니다.

- **OCF:pacemaker:HealthCPU, CPU** 유휴 상태 모니터링
- **oCF:pacemaker:HealthIOWait: CPU I/O** 대기를 모니터링합니다.
- **OCF:pacemaker:HealthSMART,** 디스크 드라이브의 **SMART** 상태를 모니터링
- **OCF:pacemaker:SysInfo:** 로컬 시스템 정보를 사용하여 다양한 노드 특성을 설정하고 디스크 공간 사용량을 모니터링하는 상태 에이전트로도 기능하는 **OCF:pacemaker:SysInfo**

또한 모든 리소스 에이전트에서 상태 노드 전략을 정의하는 데 사용할 수 있는 노드 특성을 제공할 수 있습니다.

절차

다음 절차에서는 **CPU I/O** 대기 시간이**ECDSA**를 초과하는 노드에서 리소스를 이동하는 클러스터에 대한 상태 노드 전략을 구성합니다.

1. **Pacemaker**가 노드 상태 변경에 응답하는 방법을 정의하려면 **health-node-strategy** 클러스터 속성을 설정합니다.

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

2. 상태 노드 리소스 에이전트를 사용하여 복제된 클러스터 리소스를 생성하고 **allow-unhealthy-nodes** 리소스 메타 옵션을 설정하여 노드의 상태가 복구되는지 여부를 정의하고 리소스를 노드로 다시 이동합니다. 모든 노드의 상태를 지속적으로 점검하도록 반복 모니터 작업으로 이 리소스를 구성합니다.

   이 예제에서는 **HealthIOWait** 리소스 에이전트를 생성하여 **CPU I/O** 대기를 모니터링하여 노드에서 리소스를 **25%**로 이동하기 위한 빨간색 제한을 설정합니다. 이 명령은 **allow-unhealthy-nodes** 리소스 메타 옵션을 **true** 로 설정하고 반복되는 모니터 간격을 **10**초로 구성합니다.

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

19.8. 많은 리소스를 사용하여 대규모 클러스터 구성

배포하는 클러스터가 많은 수의 노드와 많은 리소스로 구성된 경우 클러스터의 다음 매개변수 기본값을 수정해야 할 수 있습니다.

cluster-ipc-limit 클러스터 속성

cluster-ipc-limit cluster 속성은 하나의 클러스터 데몬이 다른 클러스터 데몬의 연결을 끊기 전에 최대 **IPC** 메시지 백로그입니다. 대규모 클러스터에서 다수의 리소스를 정리하거나 수정하지 않으면 한 번에 많은 **CIB** 업데이트가 제공됩니다. 이로 인해 **CIB** 이벤트 큐 임계값에 도달하기 전에 **Pacemaker** 서비스에 모든 구성 업데이트를 처리할 시간이 없는 경우 클라이언트 속도가 느려질 수 있습니다.

대규모 클러스터에서 사용할 **cluster-ipc-limit** 의 권장 값은 클러스터의 리소스 수에 노드 수를 곱한 값입니다. 로그에 클러스터 데몬 **PID**에 대한 **"Evicting client"** 메시지가 표시되면 이 값이 발생할 수 있습니다.

pcs 속성 **set** 명령을 사용하여 **cluster-ipc-limit** 의 값을 기본값인 **500**에서 늘릴 수 있습니다. 예를 들어 리소스가 **200**개인 노드 클러스터의 경우 다음 명령을 사용하여 **cluster-ipc-limit** 값을 **2000**으로 설정할 수 있습니다.

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

PCMK_ipc_buffer Pacemaker 매개변수

대규모 배포에서 내부 **Pacemaker** 메시지가 메시지 버퍼 크기를 초과할 수 있습니다. 이 경우 다음 형식의 시스템 로그에 메시지가 표시됩니다.

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

*이 메시지가 표시되면 각 노드의 **/etc/sysconfig/pacemaker** 구성 파일에서 **PCMK_ipc_buffer** 값을 늘릴 수 있습니다. 예를 들어, **PCMK_ipc_buffer** 의 값을 기본값에서 **13396332** 바이트로 늘리려면 다음과 같이 클러스터의 각 노드의 **/etc/sysconfig/pacemaker** 파일에서 주석 처리된 **PCMK_ipc_buffer** 필드를 변경합니다.*

```
PCMK_ipc_buffer=13396332
```

이 변경 사항을 적용하려면 다음 명령을 실행합니다.

```
# systemctl restart pacemaker
```

20장. PACEMAKER 클러스터에 대한 사용자 권한 설정

hacluster 이외의 특정 사용자에 대한 권한을 부여하여 Pacemaker 클러스터를 관리할 수 있습니다. 개별 사용자에게 부여할 수 있는 두 가지 권한 세트가 있습니다.

- 개별 사용자가 웹 UI를 통해 클러스터를 관리하고 네트워크를 통해 노드에 연결하는 pcs 명령을 실행할 수 있는 권한을 부여합니다. 네트워크를 통해 노드에 연결하는 명령에는 클러스터를 설정하거나 클러스터에서 노드를 추가하거나 제거하는 명령이 포함됩니다.

- 로컬 사용자가 클러스터 구성에 대한 읽기 전용 또는 읽기-쓰기 액세스를 허용할 수 있는 권한입니다. 네트워크를 통해 연결할 필요가 없는 명령에는 리소스를 생성하고 제약 조건을 구성하는 명령과 같이 클러스터 구성을 편집하는 명령이 포함됩니다.

두 권한 세트가 모두 할당된 경우 네트워크를 통해 연결하는 명령에 대한 권한이 먼저 적용된 다음 로컬 노드에서 클러스터 구성을 편집하는 권한이 적용됩니다. 대부분의 pcs 명령에는 네트워크 액세스가 필요하지 않으며, 이 경우 네트워크 권한이 적용되지 않습니다.

20.1. 네트워크를 통해 노드 액세스에 대한 권한 설정

특정 사용자가 웹 UI를 통해 클러스터를 관리하고 네트워크를 통해 노드에 연결하는 pcs 명령을 실행하려면 haclient 그룹에 해당 사용자를 추가합니다. 클러스터의 모든 노드에서 이 작업을 수행해야 합니다.

20.2. ACL을 사용하여 로컬 권한 설정

pcs acl 명령을 사용하여 로컬 사용자가 ACL(액세스 제어 목록)을 사용하여 클러스터 구성에 대한 읽기 전용 또는 읽기-쓰기 액세스를 허용할 수 있습니다.

기본적으로 ACL은 활성화되어 있지 않습니다. ACL이 활성화되지 않으면 모든 노드에서 haclient 그룹의 멤버인 모든 사용자에게 클러스터 구성에 대한 전체 로컬 읽기/쓰기 액세스 권한이 있지만 haclient의 멤버가 아닌 사용자는 액세스 권한이 없습니다. 그러나 ACL이 활성화되면 haclient 그룹의 멤버인 사용자도 ACL을 통해 해당 사용자에게 부여된 항목에만 액세스할 수 있습니다. root 및 hacluster 사용자 계정은 ACL이 활성화된 경우에도 항상 클러스터 구성에 대한 전체 액세스 권한을 갖습니다.

로컬 사용자에 대한 권한 설정은 다음 두 단계로 수행됩니다.

1. pcs acl 역할 create...을 실행합니다. 해당 역할에 대한 권한을 정의하는 역할을 생성하는 명령입니다.

2. *pcs acl user create* 명령을 사용하여 사용자에게 생성한 역할을 할당합니다. 동일한 사용자에게 여러 역할을 할당하면 거부 권한이 우선하며 쓰기 가 우선합니다.

절차

다음 예제 절차에서는 클러스터 구성에 대해 **rouser** 라는 로컬 사용자에게 읽기 전용 액세스를 제공합니다. 구성의 특정 부분에만 액세스를 제한할 수도 있습니다.

주의

이 절차를 **root**로 수행하거나 모든 구성 업데이트를 작업 파일에 저장한 다음 완료 시 활성 **CIB**로 푸시할 수 있습니다. 그렇지 않으면 자신을 더 이상 변경하지 못하도록 잠글 수 있습니다. 작업 중인 파일에 대한 구성 업데이트를 저장하는 방법에 대한 자세한 내용은 작업 중인 파일에 대한 구성 변경 사항을 참조하십시오.

1. 이 절차를 수행하려면 사용자 **rouser** 가 로컬 시스템에 있고 사용자 **rouser** 가 **haclient** 그룹의 멤버여야 합니다.

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

2. **pcs acl enable** 명령을 사용하여 **Pacemaker ACL**을 활성화합니다.

```
# pcs acl enable
```

3. **cib**에 대한 읽기 전용 권한을 사용하여 **read-only** 라는 역할을 만듭니다.

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

4. **pcs ACL** 시스템에 사용자 **rouser** 를 생성하고 해당 사용자에게 읽기 전용 역할을 할당합니다.

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

5. *현재 **ACL**을 확인합니다.*

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

6. ***rouser** 가 **pcs** 명령을 실행하는 각 노드에서 **rouser** 로 로그인하고 로컬 **pcsd** 서비스에 인증합니다. **ACL** 사용자와 같은 특정 **pcs** 명령을 실행하려면 이 작업이 필요합니다.*

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

21장. 리소스 모니터링 작업

*리소스가 정상 상태로 유지되도록 모니터링 작업을 리소스 정의에 추가할 수 있습니다. 리소스에 대한 모니터링 작업을 지정하지 않으면 기본적으로 **pcs** 명령은 리소스 에이전트에 의해 결정되는 간격과 함께 모니터링 작업을 생성합니다. 리소스 에이전트가 기본 모니터링 간격을 제공하지 않으면 **pcs** 명령은 **60** 초 간격으로 모니터링 작업을 생성합니다.*

다음 표에는 리소스 모니터링 작업의 속성이 요약되어 있습니다.

*표 **21.1.** 작업 속성*

필드	설명
id	작업의 고유 이름입니다. 시스템이 작업을 구성할 때 이 값을 할당합니다.
name	수행할 작업입니다. 공통 값: **monitor**,**start**,**stop**
간격	0이 아닌 값으로 설정하면 이 빈도에서 반복되는 반복 작업이 생성됩니다. 0이 아닌 값은 작업 **이름이 monitor** 로 설정된 경우에만 적합합니다. 반복 모니터 작업은 리소스 시작이 완료되면 즉시 실행되고 이후 모니터 작업은 이전 모니터 작업이 완료된 후 시작됩니다. 예를 들어, **interval=20s** 를 갖는 모니터 작업이 01:00:00에서 실행되는 경우, 다음 모니터 작업은 01:00:20에서 발생하지 않지만 첫 번째 모니터 작업이 완료된 후 20초 후에 수행됩니다. 기본값인 0으로 설정하면 이 매개변수를 사용하여 클러스터에서 생성한 작업에 사용할 값을 제공할 수 있습니다. 예를 들어 **간격이** 0으로 설정되면 작업 **이름이 start** 로 설정되고 **시간제한** 값이 40으로 설정된 경우 Pacemaker에서 이 리소스를 시작할 때 40초의 시간 초과를 사용합니다. 0 간격을 사용하는 **모니터** 작업을 사용하면 기본값이 필요하지 않은 경우 모든 리소스의 현재 상태를 가져오기 위해 Pacemaker에서 시작 시 Pacemaker에서 수행하는 프로브의 **시간 초과**/**on-fail**/**enabled** 값을 설정할 수 있습니다.
timeout	이 매개변수로 설정된 시간 내에 작업이 완료되지 않으면 작업을 중단하여 실패한 것으로 간주합니다. **pcs resource op defaults** 명령으로 설정된 경우 기본값은 **timeout** 값이며 설정되지 않은 경우 20초입니다. 시스템에 작업을 수행할 수 있는 시간보다 더 많은 시간이 필요한 리소스가 포함되어 있는 경우(예: **start**,**stop** 또는 **monitor**), 원인을 조사하고 장기간 실행 시간이 예상되면 이 값을 늘릴 수 있습니다. **시간 초과** 값은 모든 종류의 지연이 아니며 시간 초과 기간이 완료되기 전에 작업이 반환되는 경우 전체 시간 초과 기간을 대기하지 않습니다.

필드	설명
On-fail	이 작업이 실패하면 수행할 작업입니다. 허용되는 값: * **Ignore** - 리소스가 실패하지 않았습니다. * **block** - 리소스에서 추가 작업을 수행하지 마십시오. * **stop** - 리소스를 중지하고 다른 곳에서 시작하지 마십시오. * **재시작** - 리소스를 중지하고 다시 시작합니다(다른 노드에서). * **fence** - 리소스가 실패한 노드를 대기합니다. * **standby** - 리소스가 실패한 노드에서 모든 리소스를 이동합니다. **데모 -** 리소스에 대한 **승격** 작업이 실패하면 리소스가 데모되지만 완전히 중지되지는 않습니다. 리소스에 대한 **모니터** 작업이 실패하면 간격이 0이 아닌 값으로 **설정되고 role** 이 Promoted(승격)로 설정된 경우 **리소스** 가 강등되지만 완전히 중지되지는 않습니다. STONITH가 활성화되어 있으면 **중지** 작업의 기본값은 **fence** 입니다. 그렇지 않으면 **차단** 됩니다. 다른 모든 작업은 기본적으로 다시 **시작됩니다**.
enabled	**false** 인 경우 작업이 존재하지 않는 것처럼 처리됩니다. 허용되는 값: **true**,**false**

21.1. 리소스 모니터링 작업 구성

다음 명령을 사용하여 리소스를 생성할 때 모니터링 작업을 구성할 수 있습니다.

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

*예를 들어 다음 명령은 모니터링 작업을 사용하여 **IPaddr2** 리소스를 생성합니다. 새 리소스는 **IP** 주소가 **192.168.0.99**이고 넷마스크가 **eth2** 에서 **24**인 **VirtualIP** 라고 합니다. 모니터링 작업은 **30**초마다 수행됩니다.*

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

또는 다음 명령을 사용하여 기존 리소스에 모니터링 작업을 추가할 수 있습니다.

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

다음 명령을 사용하여 구성된 리소스 작업을 삭제합니다.

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

참고

기존 작업을 제대로 제거하려면 정확한 작업 속성을 지정해야 합니다.

모니터링 옵션 값을 변경하려면 리소스를 업데이트할 수 있습니다. 예를 들어 다음 명령을 사용하여 **VirtualIP** 를 생성할 수 있습니다.

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

기본적으로 이 명령은 이러한 작업을 생성합니다.

```
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)
```

중지 시간 제한 작업을 변경하려면 다음 명령을 실행합니다.

```
# 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. 글로벌 리소스 작업 기본값 구성

pcs resource op defaults update 명령을 사용하여 모든 리소스에 대한 리소스 작업의 기본값을 변경할 수 있습니다.

다음 명령은 모든 모니터링 작업에 대해 시간 초과 값**240**초의 글로벌 기본값을 설정합니다.

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

이전 릴리스의 모든 리소스에 대한 리소스 작업 기본값을 설정하는 원래 **pcs resource op defaults**

name=value 명령은 둘 이상의 기본값 세트가 구성되어 있지 않은 한 계속 지원됩니다. 그러나 pcs resource op defaults 업데이트가 명령의 기본 버전입니다.

21.2.1. 리소스별 작업 값 덮어쓰기

클러스터 리소스는 옵션이 클러스터 리소스 정의에 지정되지 않은 경우에만 전역 기본값을 사용합니다. 기본적으로 리소스 에이전트는 모든 작업에 대한 시간 제한 옵션을 정의합니다. 글로벌 작업 시간 초과 값을 사용하려면 timeout 옵션 없이 클러스터 리소스를 명시적으로 생성해야 합니다. 그렇지 않으면 다음 명령과 같이 클러스터 리소스를 업데이트하여 시간 초과 옵션을 제거해야 합니다.

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

예를 들어 모든 모니터링 작업에 대해 시간 제한 값의 전역 기본값을 설정하고 클러스터 리소스 VirtualIP 를 업데이트하여 모니터 작업의 시간 초과 값을 제거한 후, 시작에 대한 시간 제한 값을 갖게 됩니다. 그러면 리소스 VirtualIP 는 각각 20s, 40s 및240s의 작업에 대한 시간 초과 값을 갖습니다. 시간 초과 작업의 전역 기본값은 이전 명령에서 기본 시간 제한 옵션이 제거된 모니터 작업에서만 적용됩니다.

```
# 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. 리소스 집합에 대한 리소스 작업의 기본값 변경

pcs resource op defaults set create 명령을 사용하여 여러 리소스 작업 기본값 세트를 생성할 수 있으므로 리소스 및 작업 식이 포함된 규칙을 지정할 수 있습니다. Pacemaker에서 지원하는 모든 규칙 표현식은 허용됩니다.

이 명령을 사용하면 특정 유형의 모든 리소스에 대한 기본 리소스 작업 값을 구성할 수 있습니다. 예를 들어 번들이 사용 중일 때 Pacemaker에서 생성한 암시적 podman 리소스를 구성할 수 있습니다.

다음 명령은 모든 podman 리소스에 대한 모든 작업에 대해 기본 시간 초과 값을 90s로 설정합니다. 이 예에서 ::podman 은 podman 유형의 모든 클래스의 리소스를 나타냅니다.

리소스 작업 기본값의 이름을 지정하는 id 옵션은 필수가 아닙니다. 이 옵션을 설정하지 않으면 pcs 가 ID를 자동으로 생성합니다. 이 값을 설정하면 보다 설명적인 이름을 제공할 수 있습니다.

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

다음 명령은 모든 리소스에 대한 중지 작업에 대한 기본 타임아웃 값 **120s**를 설정합니다.

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

특정 유형의 모든 리소스에 대한 특정 작업에 대한 기본 시간 초과 값을 설정할 수 있습니다. 다음 예제에서는 모든 **podman** 리소스에 대한 중지 작업에 대한 기본 시간 초과 값을 **120s**로 설정합니다.

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

21.2.3. 현재 구성된 리소스 작업 기본값 표시

pcs resource op defaults 명령은 사용자가 지정한 규칙을 포함하여 현재 구성된 리소스 작업에 대한 기본값 목록을 표시합니다.

다음 명령은 모든 **podman** 리소스에 대한 기본 시간 초과 값 **90s** 및 리소스 작업 기본값 세트의 **ID**가 **podman -timeout** 으로 설정된 클러스터의 기본 작업 값을 표시합니다.

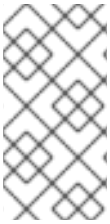
```
# pcs resource op defaults
Meta Attrs: podman-timeout
  timeout=90s
  Rule: boolean-op=and score=INFINITY
    Expression: resource ::podman
```

다음 명령은 모든 **podman** 리소스에 대한 중지 작업 **120s**의 기본 시간 제한 값으로 구성된 클러스터와 리소스 작업 기본값 세트의 **ID**가 **podman-stop-timeout** 으로 설정된 클러스터의 기본 작업 값을 표시합니다.

```
# 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. 여러 모니터링 작업 구성

리소스 에이전트가 지원하는 만큼의 모니터 작업을 사용하여 단일 리소스를 구성할 수 있습니다. 이렇게하면 매 분마다 초급한 건강 검사를 할 수 있으며 더 높은 간격으로 점진적으로 더 강렬한 건강 검사를 할 수 있습니다.



참고

여러 모니터 작업을 구성할 때 동일한 간격으로 두 작업이 수행되지 않는지 확인해야 합니다.

*다른 수준에서 더 심층적인 검사를 지원하는 리소스에 대한 추가 모니터링 작업을 구성하려면 **OCF_CHECK_LEVEL=n** 옵션을 추가합니다.*

*예를 들어 다음 **IPaddr2** 리소스를 구성하는 경우 기본적으로 이렇게 하면 간격이 **10**초이고 시간 초과 값이 **20**초인 모니터링 작업이 생성됩니다.*

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

*가상 **IP**에서 깊이가 **10**인 다른 검사를 지원하는 경우 다음 명령을 수행하면 **Pacemaker**가 일반 가상 **IP** 확인 **10**초마다 **60**초마다 고급 모니터링 검사를 수행합니다. **(**참고로, **10**초 간격으로 추가 모니터링 작업을 구성해서는 안 됩니다.**)***

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

22장. PACEMAKER 클러스터 속성

클러스터 속성은 클러스터 작업 중에 발생할 수 있는 상황에서 클러스터가 작동하는 방식을 제어합니다.

22.1. 클러스터 속성 및 옵션 요약

다음 표에서는 **Pacemaker** 클러스터 속성을 요약하여 해당 속성에 설정할 수 있는 속성의 기본값과 가능한 값을 표시합니다.

펜싱 동작을 결정하는 추가 클러스터 속성이 있습니다. 이러한 속성에 대한 자세한 내용은 펜싱 *장치의 일반 속성에서 펜싱 동작을 결정하는 클러스터 속성* 표를 참조하십시오.

참고

이 표에 설명된 속성 외에도 클러스터 소프트웨어에 의해 노출되는 추가 클러스터 속성이 있습니다. 이러한 속성의 경우 기본값에서 값을 변경하지 않는 것이 좋습니다.

표 22.1. 클러스터 속성

옵션	기본값	설명
batch-limit	0	클러스터가 병렬로 실행할 수 있는 리소스 작업 수입니다. "correct" 값은 네트워크 및 클러스터 노드의 속도 및 로드에 따라 달라집니다. 기본값은 0이면 노드의 CPU 로드가 높은 경우 클러스터에서 제한을 동적으로 적용합니다.
migration-limit	-1(제한 없음)	클러스터에서 병렬로 실행할 수 있는 마이그레이션 작업 수입니다.

옵션	기본값	설명
no-quorum-policy	중지	클러스터에 쿼럼이 없으면 어떻게 해야 합니까. 허용되는 값: *Ignore - 모든 리소스 관리 계속 * freeze - 계속 리소스 관리하지만 영향을 받는 파티션에 없는 노드에서 리소스를 복구하지 않습니다. * stop - 영향을 받는 클러스터 파티션의 모든 리소스를 중지합니다. *심각 - 영향을 받는 클러스터 파티션의 모든 노드를 펜싱합니다. * 데모 - 클러스터 파티션이 쿼럼을 손실하면 승격된 리소스를 시연하고 다른 모든 리소스를 중지합니다.
symmetric-cluster	true	기본적으로 리소스를 모든 노드에서 실행할 수 있는지 여부를 나타냅니다.
cluster-delay	60s	네트워크를 통해 왕복이 지연됩니다(작업 실행 제외). "correct" 값은 네트워크 및 클러스터 노드의 속도 및 로드에 따라 달라집니다.
dc-deadtime	20s	시작하는 동안 다른 노드의 응답을 대기하는 시간입니다. "올바른" 값은 네트워크의 속도와 로드와 사용된 스위치 유형에 따라 달라집니다.
stop-orphan-resources	true	삭제된 리소스를 중지할지 여부를 나타냅니다.
stop-orphan-actions	true	삭제된 작업을 취소할지 여부를 나타냅니다.

옵션	기본값	설명
start-failure-is-fatal	true	특정 노드에서 리소스를 시작하지 못하는 경우 해당 노드에서 추가 시작 시도를 방지합니다. **false** 로 설정하면 클러스터는 리소스의 현재 실패 수 및 마이그레이션 임계값에 따라 동일한 노드에서 다시 시작할지 여부를 결정합니다. 리소스에 대한 **migration-threshold** 옵션 설정에 대한 자세한 내용은 리소스 메타 옵션 구성을 참조하십시오. **start-failure-is-fatal** 을 **false** 로 설정하면 리소스가 모든 종속 작업을 보유할 수 없는 한 개의 결함이 있는 노드가 허용될 위험이 있습니다. 따라서 **start-failure-is-fatal** 의 기본값은 true입니다. 낮은 마이그레이션 임계값을 설정하여 **start-failure-is-fatal=false** 를 설정할 위험이 완화되어 여러 번의 실패 후에도 다른 작업이 진행될 수 있습니다.
pe-error-series-max	-1 (all)	스케줄러 입력 수로 인해 저장할 ERROR가 발생합니다. 문제를 보고할 때 사용합니다.
pe-warn-series-max	-1 (all)	스케줄러 입력 수로 인해 경고가 저장됩니다. 문제를 보고할 때 사용합니다.
pe-input-series-max	-1 (all)	저장할 "일반" 스케줄러 입력 수입니다. 문제를 보고할 때 사용합니다.
cluster-infrastructure		Pacemaker가 현재 실행 중인 메시징 스택입니다. 정보 및 진단을 위해 사용되며 사용자가 구성할 수 없습니다.
dc-version		클러스터 설계 컨트롤러 (DC)의 Pacemaker 버전입니다. 진단 목적으로 사용되며, 사용자 구성이 불가능합니다.
cluster-recheck-interval	15분	Pacemaker는 주로 이벤트 중심이며, 장애 시간 초과 및 대부분의 시간 기반 규칙에 대해 클러스터를 재확인할 시기를 미리 확인합니다. Pacemaker는 이 속성에서 지정한 비활성 기간 후에 클러스터를 다시 확인합니다. 이 클러스터 재확인에는 두 가지 용도가 있습니다. **date-spec** 규칙을 자주 확인하며 일부 종류의 스케줄러 버그에 대해 실패 안전으로 사용됩니다. 값이 0이면 이 폴링을 비활성화합니다. 양수 값은 시간 간격을 나타냅니다.

옵션	기본값	설명
maintenance-mode	false	유지 관리 모드는 클러스터에 "오프" 모드로 전환하도록 지시하고 달리 지시될 때까지 서비스를 시작하거나 중지하지 않도록 합니다. 유지 관리 모드가 완료되면 클러스터는 서비스의 현재 상태에 대한 온전한 확인을 수행한 다음 필요한 서비스를 중지하거나 시작합니다.
shutdown-escalation	20min	시간 후 정상적으로 종료하려고 하고 방금 종료하려고 합니다. 고급 사용 전용입니다.
stop-all-resources	false	클러스터에서 모든 리소스를 중지해야 합니다.
enable-acl	false	**pcs acl** 명령과 함께 설정된 대로 클러스터에서 액세스 제어 목록을 사용할 수 있는지 여부를 나타냅니다.
placement-strategy	default	클러스터 노드에 리소스 배치를 결정할 때 클러스터의 속성을 고려할지 여부와 방법을 고려합니다.
node-health-strategy	none	상태 리소스 에이전트와 함께 사용하는 경우 Pacemaker에서 노드 상태 변경에 응답하는 방법을 제어합니다. 허용되는 값: * **없음** - 노드 상태를 추적하지 마십시오. * **migrate-on-red** - 에이전트가 모니터링하는 로컬 조건에 따라 상태 에이전트가 노드의 상태가 **빨간색** 인 것으로 확인되는 모든 노드로 리소스가 이동합니다. * **only-green** - 에이전트가 모니터링하는 로컬 조건에 따라 상태 에이전트가 노드의 상태가 **노란색** 또는 **빨간색** 인 것으로 확인되는 모든 노드로 리소스가 이동합니다. * **Progressive**,**사용자 지정** - 상태 특성의 내부 숫자 값에 따라 상태 상태에 대한 클러스터의 대응을 세밀하게 제어하는 고급 노드 상태 전략입니다.

22.2. 클러스터 속성 설정 및 제거

*클러스터 속성 값을 설정하려면 다음 **pcs** 명령을 사용합니다.*

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

예를 들어 **symmetric-cluster** 값을 **false** 로 설정하려면 다음 명령을 사용합니다.

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

다음 명령을 사용하여 구성에서 클러스터 속성을 제거할 수 있습니다.

```
pcs property unset property
```

또는 **pcs property set** 명령의 **value** 필드를 비워 구성에서 클러스터 속성을 제거할 수 있습니다. 이렇게 하면 해당 속성이 기본값으로 복원됩니다. 예를 들어, **symmetric-cluster** 속성을 이전에 **false** 로 설정한 경우 다음 명령은 구성에서 설정한 값을 제거하고 **symmetric-cluster** 값을 기본값인 **true** 로 복원합니다.

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

22.3. 클러스터 속성 설정 쿼리

대부분의 경우 **pcs** 명령을 사용하여 다양한 클러스터 구성 요소의 값을 표시할 때 **pcs list** 또는 **pcs show** 를 서로 바꿔 사용할 수 있습니다. 다음 예제에서 **pcs list** 는 둘 이상의 속성에 대한 모든 설정의 전체 목록을 표시하는 데 사용되는 형식이며 **pcs show** 는 특정 속성의 값을 표시하는 데 사용되는 형식입니다.

클러스터에 설정된 속성 설정 값을 표시하려면 다음 **pcs** 명령을 사용합니다.

```
pcs property list
```

명시적으로 설정되지 않은 속성 설정의 기본값을 포함하여 클러스터의 속성 설정 값을 모두 표시하려면 다음 명령을 사용합니다.

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

특정 클러스터 속성의 현재 값을 표시하려면 다음 명령을 사용합니다.

```
pcs property show property
```

예를 들어 **cluster-infrastructure** 속성의 현재 값을 표시하려면 다음 명령을 실행합니다.

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

정보를 제공하기 위해 다음 명령을 사용하여 기본값 이외의 값으로 설정되었는지 여부에 관계없이 속성에 대한 모든 기본값 목록을 표시할 수 있습니다.

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

22.4. 클러스터 속성을 PCS 명령으로 내보내기

Red Hat Enterprise Linux 9.3부터 **pcs property config** 명령의 **--output-format=cmd** 옵션을 사용하여 다른 시스템에서 구성된 클러스터 속성을 다시 생성하는 데 사용할 수 있는 **pcs** 명령을 표시할 수 있습니다.

다음 명령은 **migration-limit cluster** 속성을 **10**으로 설정합니다.

```
# pcs property set migration-limit=10
```

클러스터 속성을 설정한 후 다음 명령은 다른 시스템에서 클러스터 속성을 설정하는 데 사용할 수 있는 **pcs** 명령을 표시합니다.

```
# pcs property config --output-format=cmd
pcs property set --force -- \
 migration-limit=10 \
 placement-strategy=minimal
```

23장. 클린 노드 종료 시 중지된 상태로 유지되도록 리소스 구성

클러스터 노드가 종료되면 Pacemaker의 기본 응답은 종료가 완전히 종료된 경우에도 해당 노드에서 실행 중인 모든 리소스를 중지하고 다른 위치에서 복구하는 것입니다. 노드가 완전히 종료되면 노드에 연결된 리소스가 노드에 잠겨 종료된 노드가 클러스터에 다시 참여할 때 다시 시작될 때까지 다른 곳에서 시작할 수 없도록 Pacemaker를 구성할 수 있습니다. 이렇게 하면 노드 리소스가 클러스터의 다른 노드로 장애 조치되지 않도록 서비스 중단이 허용되는 경우 유지보수 기간 중에 노드의 전원을 끄면 노드의 전원을 끕니다.

23.1. 클린 노드 종료 시 중지된 상태를 유지하도록 리소스를 구성하는 클러스터 속성

정리 노드 종료 시 리소스가 페일오버되지 않도록 하는 기능은 다음 클러스터 속성을 통해 구현됩니다.

shutdown-lock

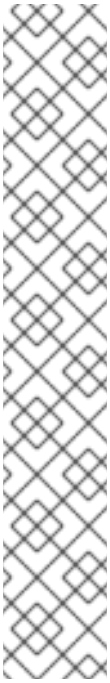
이 클러스터 속성을 기본값 **false** 로 설정하면 클러스터는 노드에서 활성 상태인 리소스를 완전히 종료합니다. 이 속성이 **true** 로 설정되면 클러스터에 다시 참여한 후 노드에서 다시 시작할 때까지 노드에서 활성 상태인 리소스를 다른 위치에서 시작할 수 없습니다.

shutdown-lock 속성은 클러스터 노드 또는 원격 노드에서 작동하지만 게스트 노드는 사용할 수 없습니다.

shutdown-lock 이 **true** 로 설정된 경우 다음 명령을 사용하여 노드에서 수동 새로 고침을 수행하여 노드가 다운될 때 한 클러스터 리소스에서 잠금을 제거할 수 있습니다.

pcs resource refresh resource node=nodename

리소스의 잠금이 해제되면 클러스터는 리소스를 다른 위치로 이동할 수 있습니다. 리소스에 대한 고정 값 또는 위치 기본 설정을 사용하여 이러한 상황이 발생할 가능성을 제어할 수 있습니다.



참고

수동 새로 고침은 다음 명령을 처음 실행하는 경우에만 원격 노드에서 작동합니다.

1. 원격 노드에서 **systemctl stop pacemaker_remote** 명령을 실행하여 노드를 중지합니다.
2. **pcs resource disable remote-connection-resource** 명령을 실행합니다.

그런 다음 원격 노드에서 수동 새로 고침을 수행할 수 있습니다.

shutdown-lock-limit

이 클러스터 속성이 기본값인 **0**이 아닌 시간으로 설정된 경우 종료가 시작된 이후 노드가 지정된 시간 내에 다시 참여하지 않는 경우 다른 노드에서 리소스를 복구할 수 있습니다.

참고

shutdown-lock-limit 속성은 다음 명령을 처음 실행하는 경우에만 원격 노드에서 작동합니다.

1. 원격 노드에서 **systemctl stop pacemaker_remote** 명령을 실행하여 노드를 중지합니다.
2. **pcs resource disable remote-connection-resource** 명령을 실행합니다.

이러한 명령을 실행하면 **shutdown-lock-limit** 로 지정된 시간이 경과한 경우 원격 노드에서 실행 중인 리소스를 다른 노드에서 복구할 수 있습니다.

23.2. SHUTDOWN-LOCK 클러스터 속성 설정

다음 예제에서는 클러스터 예제에서 **shutdown-lock** 클러스터 속성을 **true** 로 설정하고 노드가 종료되고 다시 시작될 때 이 효과를 보여줍니다. 이 예제 클러스터는 **z1.example.com,z2.example.com,**

z3.example.com 의 세 가지 노드로 구성됩니다.

절차

1. *shutdown-lock 속성을 true 로 설정하고 해당 값을 확인합니다. 이 예에서 shutdown-lock-limit 속성은 기본값 0을 유지합니다.*

```
[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. *클러스터 상태를 확인합니다. 이 예에서 세 번째 및 다섯 번째 리소스는 z1.example.com 에서 실행됩니다.*

```
[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 을 종료합니다. 그러면 해당 노드에서 실행 중인 리소스를 중지합니다.*

```
[root@z3 ~] # pcs cluster stop z1.example.com
Stopping Cluster (pacemaker)...
Stopping Cluster (corosync)...
```

4. *pcs status 명령을 실행하면 z1.example.com 노드가 오프라인 상태이고 노드가 다운되는 동안 z1.example.com 에서 실행 중인 리소스가 LOCKED 임을 보여줍니다.*

```
[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. *z1.example.com 에서 클러스터 서비스를 다시 시작하여 클러스터에 다시 참여합니다. 잠긴 리소스는 해당 노드에서 시작해야 하지만, 시작한 후에는 동일한 노드에 남아 있지 않습니다.*

```
[root@z3 ~]# pcs cluster start z1.example.com
Starting Cluster...
```

6. *이 예에서 resouces third and fiveth 는 z1.example.com 노드에서 복구됩니다.*

```
[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

...
```

24장. 노드 배치 전략 구성

Pacemaker는 모든 노드의 리소스 할당 점수에 따라 리소스를 배치할 위치를 결정합니다. 리소스는 리소스의 점수가 가장 높은 노드에 할당됩니다. 이 할당 점수는 리소스 제약 조건, resource-stickiness 설정, 각 노드에 있는 리소스의 사전 실패 내역, 각 노드의 사용률 등 요인의 조합에서 파생됩니다.

모든 노드의 리소스 할당 점수가 동일하면 기본 배치 전략 Pacemaker에서 로드 밸런싱을 위해 할당된 리소스 수가 가장 적은 노드를 선택합니다. 각 노드의 리소스 수가 동일하면 CIB에 나열된 첫 번째 적격 노드가 리소스를 실행하도록 선택됩니다.

그러나 다른 리소스가 노드 용량(예: 메모리 또는 I/O)의 비율을 크게 다르게 사용하는 경우가 많습니다. 노드에 할당된 리소스 수만 고려하여 적절하게 부하를 분산할 수는 없습니다. 또한 결합된 요구 사항이 제공된 용량을 초과하도록 리소스를 배치하면 완전히 시작되지 않거나 성능이 저하된 상태로 실행될 수 있습니다. 이러한 요인을 고려하여 Pacemaker를 사용하면 다음 구성 요소를 구성할 수 있습니다.

- 특정 노드가 제공하는 용량
- 특정 리소스에 필요한 용량
- 리소스 배치를 위한 전체 전략

24.1. 사용률 속성 및 배치 전략

노드에서 제공하거나 리소스에 필요한 용량을 구성하려면 노드 및 리소스에 사용 속성을 사용할 수 있습니다. 리소스에 대한 사용률 변수를 설정하고 해당 변수에 값을 할당하여 리소스가 필요한 사항을 나타내는 다음 해당 노드에 대해 동일한 사용률 변수를 설정하고 해당 노드에서 제공하는 내용을 나타내는 값을 해당 변수에 할당합니다.

기본 설정에 따라 사용률 속성의 이름을 지정하고 구성 요구 사항에 따라 많은 이름과 값 쌍을 정의할 수 있습니다. 사용률 속성 값은 정수여야 합니다.

24.1.1. 노드 및 리소스 용량 구성

다음 예제에서는 두 개의 노드에 대한 CPU 용량의 utilization 속성을 구성하여 이 속성을 cpu 로 설정합니다. 또한 이 속성을 변수 메모리 로 설정하여 RAM 용량의 사용률 속성을 구성합니다. 이 예제에서는 다음을 수행합니다.

- 노드 1은 두 개의 CPU 용량과 RAM 용량이 2048인으로 정의됩니다.

- 노드 2는 CPU 용량을 4개로 제공하고 RAM 용량이 2048인 것으로 정의됩니다.

```
# pcs node utilization node1 cpu=2 memory=2048
# pcs node utilization node2 cpu=4 memory=2048
```

다음 예제에서는 세 가지 다른 리소스에 필요한 것과 동일한 사용률 속성을 지정합니다. 이 예제에서는 다음을 수행합니다.

- 리소스 dummy-small 에는 CPU 용량이 1이고 RAM 용량이 1024입니다.

- 리소스 dummy-medium 에는 CPU 용량이 2이고 RAM 용량이 2048입니다.

- 리소스 dummy-large 에는 CPU 용량이 1이고 RAM 용량이 3072입니다.

```
# 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
```

사용률 속성에 정의된 대로 리소스의 요구 사항을 충족할 수 있는 충분한 용량이 있는 경우 노드는 리소스에 적합한 것으로 간주됩니다.

24.1.2. 배치 전략 구성

노드가 제공하는 용량과 리소스에 필요한 용량을 구성한 후 placement-strategy 클러스터 속성을 설정해야 합니다. 그렇지 않으면 용량 구성이 적용되지 않습니다.

placement-strategy 클러스터 속성에 네 가지 값을 사용할 수 있습니다.

- 기본값 - 사용률 값이 전혀 고려되지 않습니다. 리소스는 할당 점수에 따라 할당됩니다. 점수가 동일하면 리소스가 노드 간에 균등하게 분산됩니다.

- utilization - 사용률 값은 노드가 적합한 것으로 간주되는지 여부를 결정할 때만 고려합니다

(즉, 리소스 요구 사항을 충족할 수 있는 여유 용량이 있는지 여부). 노드에 할당된 리소스 수에 따라 로드 밸런싱이 계속 수행됩니다.

- **balanced - Utilization** 값은 노드가 리소스를 제공할 수 있는지 여부를 결정할 때 고려되므로 리소스 성능을 최적화하는 방식으로 리소스를 분산하려고 시도합니다.

- 최소 - 사용률 값은 노드가 리소스를 제공할 수 있는지 여부를 결정할 때만 고려합니다. 로드 밸런싱의 경우 가능한 한 적은 수의 노드에 리소스를 집중시키려고 하므로 나머지 노드에서 전력을 절감할 수 있습니다.

다음 예제 명령은 **placement-strategy** 값을 **balanced** 로 설정합니다. 이 명령을 실행한 후 **Pacemaker**는 복잡한 공동 배치 제약 조건 세트가 필요 없이 클러스터 전체에서 리소스의 부하를 균등하게 분배하도록 합니다.

```
# pcs property set placement-strategy=balanced
```

24.2. PACEMAKER 리소스 할당

Pacemaker는 노드 기본 설정, 노드 용량 및 리소스 할당 기본 설정에 따라 리소스를 할당합니다.

24.2.1. 노드 기본 설정

Pacemaker는 다음 전략에 따라 리소스를 할당할 때 기본 노드를 결정합니다.

- 노드 가중치가 가장 높은 노드가 먼저 사용됩니다. 노드 가중치는 노드 상태를 나타내기 위해 클러스터에서 유지 관리하는 점수입니다.

- 여러 노드에 노드 가중치가 동일한 경우:

  - **placement-strategy** 클러스터 속성이 **default** 또는 **utilization** 인 경우:

    - 할당된 리소스의 수가 가장 적은 노드를 먼저 사용합니다.

    - 할당된 리소스 수가 동일하면 **CIB**에 나열된 첫 번째 적격 노드가 먼저 사용됩니다.

- *placement-strategy 클러스터 속성이 균형 잡힌 경우:*
  - *사용 가능한 용량이 있는 노드를 먼저 사용합니다.*
  - *노드의 여유 용량이 동일하면 할당된 리소스 수가 가장 적은 노드가 먼저 사용됩니다.*
  - *노드의 여유 용량이 동일하고 할당된 리소스 수가 동일하면 CIB에 나열된 첫 번째 적합한 노드가 먼저 사용됩니다.*
- *placement-strategy 클러스터 속성이 최소 인 경우 CIB에 나열된 첫 번째 적격 노드가 먼저 소비됩니다.*

24.2.2. 노드 용량

Pacemaker는 다음 설정에 따라 사용 가능한 용량이 있는 노드를 결정합니다.

- *사용률 속성의 한 유형만 정의한 경우 사용 가능한 용량은 간단한 숫자 비교입니다.*
- *여러 가지 유형의 사용률 속성이 정의된 경우 가장 많은 특성 유형에서 숫자가 가장 높은 노드에 가장 많은 용량이 있습니다. 예를 들면 다음과 같습니다.*
  - *NodeA에 더 많은 사용 가능한 CPU가 있고 NodeB에 더 많은 여유 메모리가 있는 경우 사용 가능한 용량이 동일합니다.*
  - *NodeA에 더 많은 여유 CPU가 있는 경우 NodeB는 사용 가능한 메모리 및 스토리지가 더 많은 경우 NodeB에 더 많은 여유 용량이 있습니다.*

24.2.3. 리소스 할당 기본 설정

Pacemaker는 다음 전략에 따라 먼저 할당된 리소스를 결정합니다.

- *우선순위가 가장 높은 리소스가 먼저 할당됩니다. 리소스를 생성할 때 리소스의 우선 순위를*

설정할 수 있습니다.

- 리소스 우선순위가 동일하면 리소스 셔플링을 방지하기 위해 실행 중인 노드에서 점수가 가장 높은 리소스가 먼저 할당됩니다.

- 리소스가 실행 중인 노드의 리소스 점수가 같거나 리소스가 실행되지 않는 경우 기본 노드에서 가장 높은 점수가 먼저 할당됩니다. 기본 노드의 리소스 점수가 이 경우 **CIB**에 나열된 첫 번째 실행 가능 리소스가 먼저 할당됩니다.

24.3. 리소스 배치 전략 지침

리소스에 대한 **Pacemaker** 배치 전략이 가장 효율적으로 작동하도록 하려면 시스템을 구성할 때 다음 사항을 고려해야 합니다.

- 충분한 물리적 용량이 있는지 확인하십시오.

  노드의 물리적 용량을 정상적인 조건에서 최대 미만으로 사용하는 경우 장애 조치 중에 문제가 발생할 수 있습니다. 사용률 기능이 없어도 시간 초과 및 보조 오류가 발생할 수 있습니다.

- 노드에 대해 구성하는 기능에 일부 버퍼를 빌드합니다.

  Pacemaker 리소스에서 항상 구성된 **CPU** 양, 메모리 등의 **100%**를 사용하지 않는다는 가정하에 실제 보유보다 약간 더 많은 노드 리소스를 알립니다. 이 방법을 오버 커밋이라고 합니다.

- 리소스 우선순위를 지정합니다.

  클러스터가 서비스를 희생하는 경우 최소한에 대해 주의하는 서비스여야 합니다. 가장 중요한 리소스가 먼저 예약되도록 리소스 우선순위가 올바르게 설정되어 있는지 확인합니다.

24.4. NODEUTILIZATION 리소스 에이전트

NodeUtilization Resoure 에이전트는 사용 가능한 **CPU**, 호스트 메모리 가용성 및 하이퍼 바이저 메모리 가용성의 시스템 매개변수를 감지하고 이러한 매개변수를 **CIB**에 추가할 수 있습니다. 에이전트를 복제 리소스로 실행하여 각 노드에서 이러한 매개변수를 자동으로 채울 수 있습니다.

NodeUtilization 리소스 에이전트 및 이 에이전트의 리소스 옵션에 대한 정보를 보려면 **pcs resource**

***describe NodeUtilization** 명령을 실행합니다.*

25장. 가상 도메인을 리소스로 구성

libvirt 가상화 프레임워크에서 관리하는 가상 도메인을 **pcs resource create** 명령을 사용하여 클러스터 리소스로 구성하고 **VirtualDomain** 을 리소스 유형으로 지정할 수 있습니다.

가상 도메인을 리소스로 구성할 때 다음 사항을 고려하십시오.

- 가상 도메인을 클러스터 리소스로 구성하기 전에 중지해야 합니다.
- 가상 도메인이 클러스터 리소스이면 클러스터 툴을 제외한 시작, 중지 또는 마이그레이션해서는 안 됩니다.
- 호스트가 부팅될 때 시작하기 위해 클러스터 리소스로 구성한 가상 도메인을 구성하지 마십시오.
- 가상 도메인을 실행하도록 허용된 모든 노드는 해당 가상 도메인에 필요한 구성 파일 및 스토리지 장치에 액세스할 수 있어야 합니다.

클러스터가 가상 도메인 자체 내에서 서비스를 관리하려면 가상 도메인을 게스트 노드로 구성할 수 있습니다.

25.1. 가상 도메인 리소스 옵션

다음 표에서는 **VirtualDomain** 리소스에 대해 구성할 수 있는 리소스 옵션에 대해 설명합니다.

표 25.1. 가상 도메인 리소스에 대한 리소스 옵션

필드	기본값	설명
config		(필수) 이 가상 도메인의 **libvirt** 구성 파일에 대한 Absolute 경로입니다.
hypervisor	시스템에 따라	연결할 하이퍼바이저 URI입니다. **virsh --quiet** octets 명령을 실행하여 시스템의 기본 URI를 확인할 수 있습니다.

필드	기본값	설명
force_stop	**0**	중지 시 도메인을 강제로 종료합니다. 기본 동작은 정상 종료 시도가 실패한 경우에만 강제 종료에 의존하는 것입니다. 가상 도메인(또는 가상화 백엔드)이 정상 종료를 지원하지 않는 경우에만 이 값을 **true** 로 설정해야 합니다.
migration_transport	시스템에 따라	마이그레이션하는 동안 원격 하이퍼바이저에 연결하는 데 사용되는 전송입니다. 이 매개 변수를 생략하면 리소스에서 **libvirt** 의 기본 전송을 사용하여 원격 하이퍼바이저에 연결합니다.
migration_network_suffix		전용 마이그레이션 네트워크를 사용합니다. 마이그레이션 URI는 이 매개변수의 값을 노드 이름의 끝에 추가하여 구성됩니다. 노드 이름이 FQDN(정규화된 도메인 이름)인 경우 FQDN에서 첫 번째 마침표(.) 바로 앞에 접미사를 삽입합니다. 이 구성된 호스트 이름이 로컬에서 확인할 수 있고, 권장되는 네트워크를 통해 연결된 IP 주소에 연결할 수 있는지 확인합니다.
monitor_scripts		가상 도메인 내의 서비스를 추가로 모니터링하려면 모니터링할 스크립트 목록과 함께 이 매개 변수를 추가합니다. *참고:* 모니터 스크립트를 사용하면 모든 모니터 스크립트가 성공적으로 완료된 경우에만 **start** 및 **migrate_from** 작업이 완료됩니다. 이 지연을 수용하도록 이러한 작업의 시간 초과를 설정해야 합니다.
autoset_utilization_cpu	**true**	**true** 로 설정하면 에이전트는 **virsh** 에서 **domainU** 의 **vCPU**s 수를 감지하고 모니터가 실행될 때 리소스의 CPU 사용률에 배치합니다.
autoset_utilization_hv_memory	**true**	true를 설정하면 에이전트는 **virsh** 에서 **Max 메모리** 수를 탐지하고 모니터가 실행될 때 소스의 **hv_memory** 사용률에 배치합니다.
migrateport	random highport	이 포트는 **qemu** 마이그레이션 URI에 사용됩니다. 설정되지 않으면 포트는 임의의 고포트가 됩니다.

필드	기본값	설명
snapshot		가상 머신 이미지를 저장할 스냅샷 디렉터리의 경로입니다. 이 매개 변수를 설정하면 중지된 경우 스냅샷 디렉터리의 파일에 가상 시스템의 RAM 상태가 저장됩니다. 시작 시 도메인에 대한 상태 파일이 있는 경우 도메인이 마지막으로 중지되기 전에 동일한 상태로 복원됩니다. 이 옵션은 **force_stop** 옵션과 호환되지 않습니다.

***VirtualDomain** 리소스 옵션 외에도 리소스의 실시간 마이그레이션을 통해 다른 노드로 마이그레이션할 수 있도록 **allow-migrate metadata** 옵션을 구성할 수 있습니다. 이 옵션을 **true** 로 설정하면 상태가 손실되지 않고 리소스를 마이그레이션할 수 있습니다. 이 옵션이 기본 상태인 **false** 로 설정되면 가상 도메인이 첫 번째 노드에서 종료된 다음 한 노드에서 다른 노드로 이동하면 가상 도메인이 종료됩니다.*

25.2. 가상 도메인 리소스 생성

*다음 절차에서는 이전에 생성한 가상 머신에 대해 클러스터에 **VirtualDomain** 리소스를 생성합니다.*

절차

1. *가상 머신 관리를 위해 **VirtualDomain** 리소스 에이전트를 생성하려면 **Pacemaker**에서 가상 머신의 **xml** 구성 파일을 디스크의 파일에 덤프해야 합니다. 예를 들어 **guest1** 이라는 가상 머신을 생성한 경우 게스트를 실행할 수 있는 클러스터 노드 중 하나의 파일에 **xml** 파일을 덤프합니다. 선택한 파일 이름을 사용할 수 있습니다. 이 예제에서는 **/etc/pacemaker/guest1.xml** 을 사용합니다.*

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

2. *가상 머신의 **xml** 구성 파일을 각 노드의 동일한 위치에 게스트를 실행할 수 있는 다른 클러스터 노드에 복사합니다.*

3. *가상 도메인을 실행하도록 허용된 모든 노드가 해당 가상 도메인에 필요한 스토리지 장치에 액세스할 수 있는지 확인합니다.*

4. *가상 도메인을 실행할 각 노드에서 가상 도메인을 시작하고 중지할 수 있는지 여부를 별도로 테스트합니다.*

5.

*실행 중인 경우 게스트 노드를 종료합니다. **Pacemaker**가 클러스터에 구성되면 노드를 시작합니다. 호스트가 부팅될 때 자동으로 시작하도록 가상 시스템을 구성해서는 안 됩니다.*

6.

***pcs resource create** 명령을 사용하여 **VirtualDomain** 리소스를 구성합니다. 예를 들어 다음 명령은 **VM** 이라는 **VirtualDomain** 리소스를 구성합니다. **allow-migrate** 옵션이 **true** 로 설정되었으므로 **pcs resource move VM nodeX** 명령은 실시간 마이그레이션으로 수행됩니다.*

*이 예에서는 **migration_transport** 가 **ssh** 로 설정됩니다. **SSH** 마이그레이션이 제대로 작동하려면 키 없는 로깅이 노드 간에 작동해야 합니다.*

```
# pcs resource create VM VirtualDomain config=/etc/pacemaker/guest1.xml
migration_transport=ssh meta allow-migrate=true
```

26장. 클러스터 쿼럼 구성

Red Hat Enterprise Linux High Availability Add-On 클러스터는 분리되는 문제를 방지하기 위해 펜싱과 함께 **votequorum** 서비스를 사용합니다. 클러스터의 각 시스템에 투표가 할당되고 대부분의 투표가 있는 경우에만 클러스터 작업을 진행할 수 있습니다. 서비스는 모든 노드에 로드되어야 합니다. 클러스터 노드의 하위 집합으로 로드되면 결과를 예측할 수 없습니다. **votequorum** 서비스 구성 및 운영에 대한 자세한 내용은 **votequorum(5)** 매뉴얼 페이지를 참조하십시오.

26.1. 쿼럼 옵션 구성

pcs cluster setup 명령으로 클러스터를 생성할 때 설정할 수 있는 쿼럼 구성의 특수 기능이 있습니다. 다음 표에는 이러한 옵션이 요약되어 있습니다.

표 **26.1.** 쿼럼 옵션

옵션	설명
auto_tie_breaker	활성화하면 클러스터가 결정적 방식으로 노드의 최대 50%가 동시에 실패할 수 있습니다. 클러스터 파티션 또는 **auto_tie_breaker_node** 에 구성된 **nodeid** 와 여전히 연결된 노드(또는 설정되지 않은 경우 가장 낮은 **nodeid**)는 quorate로 유지됩니다. 다른 노드는 inquorate가 됩니다. **auto_tie_breaker** 옵션은 클러스터가 균등하게 분할된 상태에서도 작업을 계속할 수 있으므로 노드 수가 짝수인 클러스터에 주로 사용됩니다. 다중, 일관되지 않은 분할과 같은 더 복잡한 오류를 보려면 쿼럼 장치를 사용하는 것이 좋습니다. **auto_tie_breaker** 옵션은 쿼럼 장치와 호환되지 않습니다.
wait_for_all	활성화하면 모든 노드가 동시에 한 번 이상 표시되는 경우에만 처음 클러스터가 정족수에 달합니다. **wait_for_all** 옵션은 주로 2-노드 클러스터와 쿼럼 장치 **lms** (last man stand) 알고리즘을 사용하는 even-node 클러스터에 사용됩니다. **wait_for_all** 옵션은 클러스터에 두 개의 노드가 있고 쿼럼 장치를 사용하지 않고 **auto_tie_breaker** 가 비활성화된 경우 자동으로 활성화됩니다. **wait_for_all** 을 0으로 명시적으로 설정하여 이 문제를 덮어쓸 수 있습니다.
last_man_standing	활성화하면 클러스터는 특정 상황에서 **expected_votes** 및 쿼럼을 동적으로 재계산할 수 있습니다. 이 옵션을 활성화하면 **wait_for_all** 을 활성화해야 합니다. **last_man_standing** 옵션은 쿼럼 장치와 호환되지 않습니다.
last_man_standing_window	클러스터가 노드를 손실한 후 **expected_votes** 및 쿼럼을 다시 계산하기 전에 대기하는 시간(밀리초)입니다.

이러한 옵션을 구성하고 사용하는 방법에 대한 자세한 내용은 **votequorum(5)** 도움말 페이지를 참조하십시오.

26.2. 쿼럼 옵션 수정

pcs quorum update 명령을 사용하여 클러스터의 일반 쿼럼 옵션을 수정할 수 있습니다. 실행 중인 시스템에서 **quorum.two_node** 및 **quorum.expected_votes** 옵션을 수정할 수 있습니다. 다른 모든 쿼럼 옵션의 경우 이 명령을 실행하려면 클러스터를 중지해야 합니다. 쿼럼 옵션에 대한 자세한 내용은 **votequorum(5)** 도움말 페이지를 참조하십시오.

pcs quorum update 명령의 형식은 다음과 같습니다.

```
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]]
```

다음 일련의 명령은 **wait_for_all** 쿼럼 옵션을 수정하고 옵션의 업데이트된 상태를 표시합니다. 클러스터가 실행되는 동안 시스템에서 이 명령을 실행할 수 없습니다.

```
[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. 쿼럼 구성 및 상태 표시

클러스터가 실행되면 다음 클러스터 쿼럼 명령을 입력하여 쿼럼 구성 및 상태를 표시할 수 있습니다.

다음 명령은 쿼럼 구성을 보여줍니다.

```
pcs quorum [config]
```

다음 명령은 쿼럼 런타임 상태를 보여줍니다.

```
pcs quorum status
```

26.4. INQUORATE 클러스터 실행

클러스터가 장기간 중단되고 해당 노드의 손실이 발생하면 **pcs quorum expected-votes** 명령을 사용하여 라이브 클러스터의 **expected_votes** 매개변수 값을 변경할 수 있습니다. 이를 통해 쿼럼이 없는 경우 클러스터가 작업을 계속할 수 있습니다.

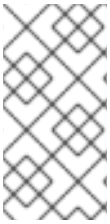
주의

라이브 클러스터에서 예상되는 투표를 변경하는 것은 매우 주의해야 합니다. 예상 투표를 수동으로 변경했기 때문에 클러스터의 **50%** 미만이 실행되는 경우 클러스터의 다른 노드를 별도로 시작하고 클러스터 서비스를 실행할 수 있으므로 데이터 손상 및 기타 예기치 않은 결과가 발생할 수 있습니다. 이 값을 변경하는 경우 **wait_for_all** 매개변수가 활성화되어 있는지 확인해야 합니다.

다음 명령은 라이브 클러스터의 예상 투표를 지정된 값으로 설정합니다. 이는 라이브 클러스터에만 영향을 미치며 구성 파일을 변경하지 않습니다. **expected_votes** 의 값은 다시 로드될 때 구성 파일의 값으로 재설정됩니다.

```
pcs quorum expected-votes votes
```

클러스터가**quorate**라는 것을 알고 있지만 클러스터를 리소스 관리로 진행하려는 경우 **pcs quorum unblock** 명령을 사용하여 쿼럼을 설정할 때 클러스터가 모든 노드를 기다리지 못하도록 할 수 있습니다.



참고

이 명령은 매우 주의해서 사용해야 합니다. 이 명령을 실행하기 전에 현재 클러스터에 없는 노드가 꺼지고 공유 리소스에 대한 액세스 권한이 없는지 확인해야 합니다.

```
# pcs quorum unblock
```

27장. 쿼럼 장치 구성

클러스터가 클러스터의 타사 중재 장치 역할을 하는 별도의 쿼럼 장치를 구성하여 허용하는 표준 쿼럼 규칙보다 더 많은 노드 오류를 유지할 수 있습니다. 노드 수가 짝수인 클러스터에는 쿼럼 장치가 권장됩니다. **two-node** 클러스터를 사용하면 쿼럼 장치를 사용하면 분할된 상황에서 발생하는 노드를 더 잘 결정할 수 있습니다.

쿼럼 장치를 구성할 때 다음 사항을 고려해야 합니다.

- 쿼럼 장치는 쿼럼 장치를 사용하는 클러스터와 동일한 사이트의 다른 물리적 네트워크에서 실행되는 것이 좋습니다. 쿼럼 장치 호스트는 기본 클러스터와 별도의 랙에 있거나 적어도 별도의 **psU**에 있어야 하며, **corosync** 링 또는 링과 동일한 네트워크 세그먼트에 있지 않아야 합니다.
- 클러스터에서 두 개 이상의 쿼럼 장치를 동시에 사용할 수 없습니다.
- 클러스터에서 두 개 이상의 쿼럼 장치를 동시에 사용할 수는 없지만 여러 클러스터에서 단일 쿼럼 장치를 동시에 사용할 수 있습니다. 쿼럼 장치를 사용하는 각 클러스터는 클러스터 노드 자체에 저장되므로 서로 다른 알고리즘과 쿼럼 옵션을 사용할 수 있습니다. 예를 들어, 단일 쿼럼 장치는 **ffsplit (fifty split)** 알고리즘을 사용하고 **lms (last man stand)** 알고리즘을 사용하는 두 번째 클러스터에서 사용할 수 있습니다.
- 쿼럼 장치는 기존 클러스터 노드에서 실행되지 않아야 합니다.

27.1. 쿼럼 장치 패키지 설치

클러스터의 쿼럼 장치를 구성하려면 다음 패키지를 설치해야 합니다.

- 기존 클러스터의 노드에 **corosync-qdevice** 를 설치합니다.

```
[root@node1:~]# dnf install corosync-qdevice
[root@node2:~]# dnf install corosync-qdevice
```

- 쿼럼 장치 호스트에 **pcs** 및 **corosync-qnetd** 를 설치합니다.

```
[root@qdevice:~]# dnf install pcs corosync-qnetd
```

- *pcsd 서비스를 시작하고 쿼럼 장치 호스트에서 시스템을 시작할 때 pcsd 를 활성화합니다.*

```
[root@qdevice:~]# systemctl start pcsd.service
[root@qdevice:~]# systemctl enable pcsd.service
```

27.2. 쿼럼 장치 구성

쿼럼 장치를 구성하고 다음 절차에 따라 클러스터에 추가합니다.

이 예제에서는 다음을 수행합니다.

- *쿼럼 장치에 사용되는 노드는 qdevice 입니다.*
- *쿼럼 장치 모델은 현재 지원되는 유일한 모델인 net 입니다. 네트워크 모델은 다음과 같은 알고리즘을 지원합니다.*
  - *FFSplit: fifty split 입니다. 이는 가장 많은 활성 노드 수를 갖는 파티션에 정확히 하나의 투표를 제공합니다.*
  - *LMS: 마지막에 있습니다. 노드가 클러스터에 남아 있는 유일한 서버인 경우 qnetd 서버를 볼 수 있는 노드가 투표를 반환합니다.*

주의

LMS 알고리즘을 사용하면 클러스터는 남은 하나의 노드로도 정족수이지만, 쿼럼 장치의 투표 성능이 number_of_nodes - 1과 같기 때문에 유용합니다. 쿼럼 장치와의 연결이 손실되면 number_of_nodes - 1 vote가 손실된다는 것을 의미합니다. 즉, 모든 노드가 활성인 클러스터만 정지 상태로 유지될 수 있습니다. 다른 클러스터는 inquorate가 됩니다.

이러한 알고리즘 구현에 대한 자세한 내용은 corosync-qdevice(8) 매뉴얼 페이지를 참조하십시오.

- 클러스터 노드는 **node1** 및 **node2** 입니다.

절차

1. 쿼럼 장치를 호스팅하는 데 사용할 노드에서 다음 명령을 사용하여 쿼럼 장치를 구성합니다. 이 명령은 쿼럼 장치 모델 **net** 을 구성 및 시작하고 부팅 시 시작되도록 장치를 구성합니다.

```
[root@qdevice:~]# pcs qdevice setup model net --enable --start
Quorum device 'net' initialized
quorum device enabled
Starting quorum device...
quorum device started
```

쿼럼 장치를 구성한 후 상태를 확인할 수 있습니다. 그러면 **corosync-qnetd** 데몬이 실행 중이며 이 데몬에 연결된 클라이언트가 없습니다. **--full** 명령 옵션은 자세한 출력을 제공합니다.

```
[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. 다음 명령을 사용하여 **firewalld** 에서 고가용성 서비스를 활성화하여 **pcsd** 데몬과 **net quorum** 장치에 필요한 방화벽의 포트를 활성화합니다.

```
[root@qdevice:~]# firewall-cmd --permanent --add-service=high-availability
[root@qdevice:~]# firewall-cmd --add-service=high-availability
```

3. 기존 클러스터의 노드 중 하나에서 쿼럼 장치를 호스팅하는 노드에서 **hacluster** 사용자를 인증합니다. 이를 통해 클러스터의 **pcs** 가 **qdevice** 호스트의 **pcs** 에 연결할 수 있지만 클러스터의 **pcs** 에 연결할 수 없습니다.

```
[root@node1:~] # pcs host auth qdevice
Username: hacluster
Password:
qdevice: Authorized
```

4. 쿼럼 장치를 클러스터에 추가합니다.

쿼럼 장치를 추가하기 전에 현재 구성 및 쿼럼 장치의 상태를 확인하여 나중에 비교할 수 있습니다. 이러한 명령의 출력은 클러스터가 쿼럼 장치를 아직 사용하지 않았으며 각 노드의 **Qdevice**

*멤버십 상태가 **NR (등록되지 않음)**임을 나타냅니다.*

```
[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
```

*다음 명령은 이전에 생성한 쿼럼 장치를 클러스터에 추가합니다. 클러스터에서 두 개 이상의 쿼럼 장치를 동시에 사용할 수 없습니다. 그러나 여러 클러스터에서 동시에 하나의 쿼럼 장치를 사용할 수 있습니다. 이 예제 명령은 **ffsplit** 알고리즘을 사용하도록 쿼럼 장치를 구성합니다. 쿼럼 장치의 구성 옵션에 대한 자세한 내용은 **corosync-qdevice(8)** 매뉴얼 페이지를 참조하십시오.*

```
[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. *쿼럼 장치의 구성 상태를 확인합니다.*

클러스터 측에서 다음 명령을 실행하여 구성이 어떻게 변경되었는지 확인할 수 있습니다.

pcs quorum 구성에 는 구성된 쿼럼 장치가 표시됩니다.

```
[root@node1:~]# pcs quorum config
Options:
Device:
  Model: net
    algorithm: ffsplit
    host: qdevice
```

pcs quorum status 명령은 쿼럼 런타임 상태를 표시하며 쿼럼 장치가 사용 중임을 나타냅니다. 각 클러스터 노드의 **Qdevice** 멤버십 정보 상태 값의 의미는 다음과 같습니다.

- **A/NA -** 쿼럼 장치는 활성 상태이며 **qdevice** 와 **corosync** 사이에 하트비트가 있는지 여부를 나타냅니다. 쿼럼 장치가 활성 상태임을 나타냅니다.

- **V/NV - V** 는 쿼럼 장치가 노드에 투표를 할 때 설정됩니다. 이 예에서는 서로 통신할 수 있으므로 두 노드 모두 **V** 로 설정됩니다. 클러스터가 두 개의 단일 노드 클러스터로 분할된 경우 노드 중 하나가 **V** 로 설정되고 다른 노드 중 하나가 **NV** 으로 설정됩니다.

- **MW/NMW -** 내부 쿼럼 장치 플래그가 설정되어 있거나 (MW) 설정되지 않았습니다 (NMW). 기본적으로 플래그는 설정되지 않으며 값은 **NMW** 입니다.

```
[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 장치 상태에 쿼럼 장치 런타임 상태가 표시됩니다.

```
[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
```

*쿼럼 장치 측에서 **corosync-qnetd** 데몬의 상태를 표시하는 다음 **status** 명령을 실행할 수 있습니다.*

```
[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. 쿼럼 장치 서비스 관리

PCS는 다음 예제 명령과 같이 로컬 호스트(corosync-qnetd)에서 쿼럼 장치 서비스를 관리하는 기능을 제공합니다. 이러한 명령은 corosync-qnetd 서비스에만 영향을 미칩니다.

```
[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. 클러스터에서 쿼럼 장치 관리

클러스터에서 쿼럼 장치 설정을 변경하고 쿼럼 장치를 비활성화하며 쿼럼 장치를 제거하는 데 사용할 수 있는 다양한 pcs 명령이 있습니다.

27.4.1. 쿼럼 장치 설정 변경

pcs quorum device update 명령을 사용하여 쿼럼 장치의 설정을 변경할 수 있습니다.

주의

쿼럼 장치 모델 net 의 호스트 옵션을 변경하려면 pcs quorum 장치 remove 및 pcs quorum device add 명령을 사용하여 이전 호스트와 새 호스트가 동일한 시스템이 아닌 경우 구성을 올바르게 설정합니다.

다음 명령은 쿼럼 장치 알고리즘을 lms 로 변경합니다.

```
[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. 쿼럼 장치 제거

다음 명령은 클러스터 노드에 구성된 쿼럼 장치를 제거합니다.

```
[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
```

쿼럼 장치를 제거한 후 쿼럼 장치 상태를 표시할 때 다음과 같은 오류 메시지가 표시됩니다.

```
[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. 쿼럼 장치 삭제

다음 명령은 쿼럼 장치 호스트에서 쿼럼 장치를 비활성화 및 중지하고 모든 구성 파일을 삭제합니다.

```
[root@qdevice:~]# pcs qdevice destroy net
Stopping quorum device...
quorum device stopped
quorum device disabled
Quorum device 'net' configuration files removed
```

28장. 클러스터 이벤트에 대한 스크립트 트리거

Pacemaker 클러스터는 이벤트 중심 시스템입니다. 이벤트가 리소스 또는 노드 오류, 구성 변경 또는 시작 또는 중지 리소스일 수 있습니다. 클러스터 이벤트가 경고 에이전트를 통해 발생할 때 일부 외부 작업을 수행하도록 Pacemaker 클러스터 경고를 구성할 수 있습니다. 이는 클러스터 호출 리소스 에이전트가 리소스 구성 및 작업을 처리하는 것과 동일한 방식으로 클러스터가 호출하는 외부 프로그램입니다.

클러스터는 환경 변수를 통해 이벤트에 대한 정보를 에이전트에 전달합니다. 에이전트는 이메일 메시지 전송 또는 파일에 로그 또는 모니터링 시스템 업데이트와 같은 이 정보를 사용하여 모든 작업을 수행할 수 있습니다.

- Pacemaker는 기본적으로 /usr/share/pacemaker/alerts 에 설치된 여러 샘플 경고 에이전트를 제공합니다. 이러한 샘플 스크립트는 그대로 복사 및 사용할 수 있거나 목적에 맞게 편집할 템플릿으로 사용할 수 있습니다. 지원하는 모든 특성 세트는 샘플 에이전트의 소스 코드를 참조하십시오.

- 샘플 경고 에이전트가 요구 사항을 충족하지 않으면 Pacemaker 경고에 대해 자체 경고 에이전트를 작성할 수 있습니다.

28.1. 샘플 경고 에이전트 설치 및 구성

샘플 경고 에이전트 중 하나를 사용하는 경우 스크립트를 검토하여 필요에 맞게 스크립트를 검토해야 합니다. 이러한 샘플 에이전트는 특정 클러스터 환경에 대한 사용자 지정 스크립트의 시작점으로 제공됩니다. Red Hat은 경고 에이전트가 Pacemaker와 통신하는 데 사용하는 인터페이스를 지원하지만 Red Hat은 사용자 정의 에이전트 자체를 지원하지 않습니다.

샘플 경고 에이전트 중 하나를 사용하려면 클러스터의 각 노드에 에이전트를 설치해야 합니다. 예를 들어 다음 명령은 alert_file.sh.sample 스크립트를 alert_file.sh 로 설치합니다.

```
# install --mode=0755 /usr/share/pacemaker/alerts/alert_file.sh.sample
/var/lib/pacemaker/alert_file.sh
```

스크립트를 설치한 후 스크립트를 사용하는 경고를 만들 수 있습니다.

다음 예제에서는 설치된 alert_file.sh 경고 에이전트를 사용하여 이벤트를 파일에 기록하는 경고를 구성합니다. 경고 에이전트는 최소한의 권한 집합이 있는 hacluster 사용자로 실행됩니다.

이 예제에서는 이벤트를 기록하는 데 사용할 로그 파일 pcs mk_alert_file.log 를 생성합니다. 그런 다

음 경고 에이전트를 만들고 수신자로 로그 파일에 경로를 추가합니다.

```
# 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
```

*다음 예제에서는 **alert_snmp.sh.sample** 스크립트를 **alert_snmp.sh** 로 설치하고 설치된 **alert_snmp.sh** 경고 에이전트를 사용하여 클러스터 이벤트를 **SNMP** 트랩으로 보내는 경고를 구성합니다. 기본적으로 스크립트는 성공적인 **monitor** 호출을 **SNMP** 서버로 보내는 것을 제외한 모든 이벤트를 보냅니다. 이 예제에서는 타임스탬프 형식을 메타 옵션으로 구성합니다. 경고를 구성한 후 이 예제에서는 경고에 대한 수신자를 구성하고 경고 구성을 표시합니다.*

```
# 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)
```

*다음 예제에서는 **alert_smtp.sh** 에이전트를 설치한 다음 설치된 경고 에이전트를 사용하여 클러스터 이벤트를 이메일 메시지로 보내는 경고를 구성합니다. 경고를 구성한 후 이 예제에서는 수신자를 구성하고 경고 구성을 표시합니다.*

```
# 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. 클러스터 경고 생성

*다음 명령은 클러스터 경고를 생성합니다. 구성하는 옵션은 추가 환경 변수로 지정하는 경로에서 경고 에이전트 스크립트에 전달되는 에이전트별 구성 값입니다. **id** 값을 지정하지 않으면 다음이 생성됩니다.*

```
pcs alert create path=path [id=alert-id] [description=description] [options [option=value]...]
[meta [meta-option=value]...]
```

여러 경고 에이전트를 구성할 수 있습니다. 클러스터는 각 이벤트에 대해 모든 경고를 호출합니다. 경고 에이전트는 클러스터 노드에서만 호출됩니다. Pacemaker 원격 노드와 관련된 이벤트에 대해 호출되지만 해당 노드에서는 호출되지 않습니다.

다음 예제에서는 각 이벤트에 대해 myscript.sh 를 호출하는 간단한 경고를 생성합니다.

```
# pcs alert create id=my_alert path=/path/to/myscript.sh
```

28.3. 클러스터 경고 표시, 수정 및 제거

클러스터 경고를 표시, 수정 및 제거하는 데 사용할 수 있는 다양한 pcs 명령이 있습니다.

다음 명령은 구성된 모든 경고와 구성된 옵션 값을 표시합니다.

```
pcs alert [config|show]
```

다음 명령은 지정된 alert-id 값으로 기존 경고를 업데이트합니다.

```
pcs alert update alert-id [path=path] [description=description] [options [option=value]...] [meta
[meta-option=value]...]
```

다음 명령은 지정된 alert-id 값을 사용하여 경고를 제거합니다.

```
pcs alert remove alert-id
```

또는 pcs alert remove 명령과 동일한 pcs alert delete 명령을 실행할 수 있습니다. pcs alert delete 및 pcs alert remove 명령을 사용하면 둘 이상의 경고를 삭제할 수 있습니다.

28.4. 클러스터 경고 수신자 구성

일반적으로 경고는 수신자에게 전송됩니다. 따라서 각 경보는 하나 이상의 수신자로 추가로 구성될 수 있습니다. 클러스터는 각 수신자에 대해 에이전트를 별도로 호출합니다.

수신자는 경고 에이전트가 인식할 수 있는 모든 항목일 수 있습니다. 즉, IP 주소, 이메일 주소, 파일 이름 또는 특정 에이전트가 지원하는 모든 항목이 될 수 있습니다.

다음 명령은 지정된 경고에 새 수신자를 추가합니다.

```
pcs alert recipient add alert-id value=recipient-value [id=recipient-id] [description=description] [options [option=value]...] [meta [meta-option=value]...]
```

다음 명령은 기존 경고 수신자를 업데이트합니다.

```
pcs alert recipient update recipient-id [value=recipient-value] [description=description] [options [option=value]...] [meta [meta-option=value]...]
```

다음 명령은 지정된 경고 수신자를 제거합니다.

```
pcs alert recipient remove recipient-id
```

또는 **pcs alert recipient remove** 명령과 동일한 **pcs alert recipient delete** 명령을 실행할 수 있습니다. **pcs alert recipient remove** 및 **pcs alert recipient delete** 명령을 사용하면 두 개 이상의 경고 수신자를 제거할 수 있습니다.

다음 예제 명령은 **my-recipient-id** 의 수신자 ID가 있는 경고 수신자 **my-alert-recipient -id**를 **my-alert** 경고에 추가합니다. 그러면 수신자 **some-address** 를 환경 변수로 전달하여 각 이벤트에 대해 **my-alert** 에 대해 구성된 경고 스크립트를 호출하도록 클러스터가 구성됩니다.

```
# pcs alert recipient add my-alert value=my-alert-recipient id=my-recipient-id options value=some-address
```

28.5. 경고 메타 옵션

리소스 에이전트와 마찬가지로 경고 에이전트가 **Pacemaker**에서 호출하는 방법에 영향을 주기 위해 메타 옵션을 구성할 수 있습니다. 다음 표에서는 경고 메타 옵션을 설명합니다. 메타 옵션은 수신자별로 및 경고 에이전트별로 구성할 수 있습니다.

표 **28.1.** 경고 메타 옵션

meta-Attribute	기본값	설명

meta-Attribute	기본값	설명
enabled	**true**	(RHEL 9.3 이상) 경고에 대해 **false** 로 설정하면 경고가 사용되지 않습니다. 해당 경고의 특정 수신자에 대해 경고 및 **false** 에 대해 **true** 로 설정하면 해당 수신자가 사용되지 않습니다.
timestamp-format	%H:%M:%S.%06N	이벤트의 타임 스탬프를 에이전트로 보낼 때 클러스터가 사용됩니다. **date**(1) 명령과 함께 사용되는 문자열입니다.
timeout	30s	경고 에이전트가 이 시간 내에 완료되지 않으면 종료됩니다.

*다음 예제에서는 **myscript.sh** 스크립트를 호출하는 경고를 구성한 다음 두 수신자를 경고에 추가합니다. 첫 번째 수신자는 **my-alert-recipient1 ID**를 가지며 두 번째 수신자는 **my-alert-recipient2** 의 **ID**를 갖습니다. 이 스크립트는 각 이벤트에 대해 두 번 호출되며 각 호출은 **15**초의 시간 초과를 사용합니다. 하나의 호출은 **%D %H:%M** 형식의 타임스탬프를 사용하여 수신자 **someuser@example.com** 에 전달되고, 다른 호출은 **%c** 형식의 타임 스탬프를 사용하여 수신자 **otheruser@example.com** 로 전달됩니다.*

```
# 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. 클러스터 경고 구성 명령 예

다음 순차적 예제에서는 경고를 만들고 수신자를 추가하고 구성된 경고를 표시하는 데 사용할 형식을 표시하는 몇 가지 기본 경고 구성 명령을 보여줍니다.

*클러스터의 각 노드에 경고 에이전트를 설치해야 하는 동안 **pcs** 명령을 한 번만 실행해야 합니다.*

다음 명령은 간단한 경고를 만들고, 경고에 두 개의 수신자를 추가하고 구성된 값을 표시합니다.

- *경고 **ID** 값이 지정되지 않았으므로 시스템은 경고의 경고 **ID** 값을 생성합니다.*

- *첫 번째 수신자 생성 명령은 **rec_value** 의 수신자를 지정합니다. 이 명령은 수신자 **ID**를 지정하지 않으므로 **alert-recipient** 값은 수신자 **ID**로 사용됩니다.*

- *두 번째 수신자 생성 명령은 **rec_value2** 의 수신자를 지정합니다. 이 명령은 수신자에 대한 **my-recipient** 의 수신자 **ID**를 지정합니다.*

```
# 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)
```

*다음 명령은 해당 경고에 대한 두 번째 경고 및 수신자를 추가합니다. 두 번째 경고의 경고 **ID**는 **my-alert** 이고 수신자 값은 **my-other-recipient** 입니다. 수신자 **ID**가 지정되지 않았으므로 시스템은 **my-alert-recipient** 의 수신자 **ID**를 제공합니다.*

```
# 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)
```

*다음 명령은 **my-alert** 경고 및 수신자 **my-alert -recipient** 에 대한 경고 값을 수정합니다.*

```
# 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
```

*다음 명령은 경고 에서 **my-alert-recipient** 수신자를 제거합니다.*

```
# 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
```

*다음 명령은 구성에서 **myalert** 를 제거합니다.*

```
# pcs alert remove myalert
# pcs alert
Alerts:
 Alert: alert (path=/my/path)
  Recipients:
   Recipient: alert-recipient (value=rec_value)
```

28.7. 클러스터 경고 에이전트 작성

*노드 경고, 펜싱 경고, 리소스 경고의 세 가지 유형의 **Pacemaker** 클러스터 경고가 있습니다. 경고 에이전트에 전달되는 환경 변수는 경고 유형에 따라 다를 수 있습니다. 다음 표에서는 경고 에이전트에 전달되는 환경 변수를 설명하고 환경 변수가 특정 경고 유형과 연결된 시기를 지정합니다.*

*표 **28.2.** 경고 에이전트로 전달되는 환경 변수*

환경 변수	설명
CRM_alert_kind	경고 유형(node, fencing 또는 resource)
CRM_alert_version	경고를 전송하는 Pacemaker 버전

환경 변수	설명
CRM_alert_recipient	구성된 수신자
CRM_alert_node_sequence	로컬 노드에서 경고를 발행할 때마다 시퀀스 번호가 증가했으며, 이는 Pacemaker에서 경보를 발행한 순서를 참조하는 데 사용할 수 있습니다. 나중에 발생한 이벤트에 대한 경고는 이전 이벤트에 대한 경고보다 시퀀스 번호가 더 높습니다. 이 숫자에는 클러스터 전체의 의미가 없습니다.
CRM_alert_timestamp	**timestamp-format** meta 옵션에 지정된 형식으로 에이전트를 실행하기 전에 생성된 타임 스탬프입니다. 이를 통해 에이전트는 에이전트 자체를 호출한 시점과 관계없이 이벤트가 발생한 시점(시스템 로드 또는 기타 상황으로 인해 지연될 수 있음)의 안정적이고 고정밀한 시간을 가질 수 있습니다.
CRM_alert_node	영향을 받는 노드의 이름
CRM_alert_desc	이벤트에 대한 세부 정보입니다. 노드 경고의 경우 노드의 현재 상태(member 또는 lost)입니다. 펜싱 경고의 경우, 해당하는 경우 origin, target 및 fencing 작업 오류 코드를 포함하여 요청된 펜싱 작업에 대한 요약입니다. 리소스 경고의 경우 이 문자열은 Gantt_**alert_status** 와 동일한 읽기 가능한 문자열입니다.
CRM_alert_nodeid	상태가 변경된 노드의 ID(노드 경고만 제공)
CRM_alert_task	요청된 펜싱 또는 리소스 작업(라이그레이션 및 리소스 경고만 제공)
CRM_alert_rc	수치에는 펜싱 또는 리소스 작업의 반환 코드 (fencing 및 리소스 경고만 제공)
CRM_alert_rsc	영향을 받는 리소스의 이름(리소스 경고만 해당)
CRM_alert_interval	리소스 작업의 간격(리소스 경고만 해당)
CRM_alert_target_rc	작업의 예상 수치 반환 코드 (리소스 경고만 해당)
CRM_alert_status	Pacemaker에서 작업 결과를 나타내는 데 사용하는 숫자 코드 (리소스 경고만 해당)

경고 에이전트를 작성할 때는 다음과 같은 우려 사항을 고려해야 합니다.

- ***경고 에이전트는 수신자 없이 호출할 수 있으므로(구성된 경우) 에이전트는 해당 경우에만 종료하더라도 이 상황을 처리할 수 있어야 합니다. 사용자는 단계를 통해 구성을 수정하고 나중에***

수신자를 추가할 수 있습니다.

- *경고에 대해 둘 이상의 수신자가 구성된 경우 수신자당 경고 에이전트가 한 번 호출됩니다. 에이전트가 동시에 실행할 수 없는 경우 단일 수신자만 사용하여 구성해야 합니다. 그러나 에이전트는 수신자를 목록으로 해석하기 위한 자유롭다.*

- *클러스터 이벤트가 발생하면 별도의 프로세스와 동시에 모든 경고가 종료됩니다. 얼마나 많은 경고와 수신자가 구성되어 있으며 경고 에이전트 내에서 수행되는 작업에 따라 상당한 로드 버스트가 발생할 수 있습니다. 에이전트가 이를 고려하여(예: 리소스 집약적인 작업을 직접 실행하는 대신 다른 인스턴스에 작업을 대기열에 추가함)를 작성할 수 있습니다.*

- *경고 에이전트는 최소한의 권한 집합이 있는 **hacluster** 사용자로 실행됩니다. 에이전트에 추가 권한이 필요한 경우 에이전트가 필요한 명령을 적절한 권한이 있는 다른 사용자로 실행하도록 **sudo** 를 구성하는 것이 좋습니다.*

- ***CRM_alert_timestamp** 와 같은 사용자 구성 매개변수의 유효성을 검증하고 분리하십시오(사용자 구성 타임 포맷(사용자 구성 **timestamp-format**), **CRM_alert_recipient** 및 모든 경고 옵션에 의해 지정됨). 이는 구성 오류로부터 보호하는 데 필요합니다. 또한 일부 사용자가 클러스터 노드에 대한 **hacluster-level** 액세스 권한이 없어도 **CIB**를 수정할 수 있는 경우 잠재적인 보안 문제이기도 하며 코드 삽입 가능성을 피해야 합니다.*

- *클러스터에 **on-fail** 매개변수가 **fence** 로 설정된 작업이 포함된 리소스가 포함된 경우 실패에 대한 여러 개의 펜스 알림이 있으며, 이 매개 변수가 하나의 추가 알림을 더 추가하여 각 리소스에 대해 하나씩 추가 알림이 설정됩니다. **pacemaker-fenced** 및 **pacemaker-controld** 둘 다 알림을 보냅니다. 그러나 **Pacemaker**는 전송되는 알림 수에 관계없이 하나의 실제 펜스 작업만 수행합니다.*

참고

*경고 인터페이스는 **ocf:pacemaker:ClusterMon** 리소스에서 사용하는 외부 스크립트 인터페이스와 역호환되도록 설계되었습니다. 이 호환성을 유지하기 위해 경고 에이전트에 전달된 환경 변수는 **CRM_notify_** 및 **CRM_alert_** 앞에 제공됩니다. 호환성의 한 가지 중단은 **ClusterMon** 리소스가 **root** 사용자로 외부 스크립트를 실행했지만 경고 에이전트는 **hacluster** 사용자로 실행된다는 것입니다.*

29장. 다중 사이트 PACEMAKER 클러스터

클러스터가 둘 이상의 사이트에 걸쳐 있으면 사이트 간 네트워크 연결 문제로 인해 분할된 상황이 발생할 수 있습니다. 연결이 떨어지면 한 사이트에 노드가 다른 사이트의 노드가 실패했는지 또는 실패한 사이트 상호 연결로 작동하는지 확인할 수있는 방법이 없습니다. 또한 동기를 유지하기 위해 너무 멀리 떨어져 있는 두 사이트에서 고가용성 서비스를 제공하는 것이 문제가 될 수 있습니다. 이러한 문제를 해결하기 위해 **Pacemaker**는 **Booth** 클러스터 티켓 관리자를 사용하여 여러 사이트에 걸쳐 고가용성 클러스터를 구성하는 기능을 완벽하게 지원합니다.

29.1. BOOTH 클러스터 티켓 관리자 개요

Booth 티켓 관리자 는 특정 사이트에서 클러스터 노드를 연결하는 네트워크와 다른 물리적 네트워크에서 실행되어야 하는 분산 서비스입니다. 사이트의 일반 클러스터 위에 있는 다른 연결이 느릴 수 있는 클러스터, 부스 형성을 제공합니다. 이렇게 집계된 통신 계층은 개별 부스 티켓에 대한 합의 기반 의사 결정 프로세스를 용이하게 합니다.

Booth 티켓 은 **Booth formation**의 **singleton**이며 시간에 민감한 권한 부여 단위를 나타냅니다. 특정 티켓을 실행하기 위해 리소스를 구성할 수 있습니다. 이를 통해 한 번에 하나의 사이트에서만 리소스를 실행할 수 있으며 티켓 또는 티켓이 부여되었습니다.

Booth 형성은 서로 다른 사이트에서 실행되는 클러스터로 구성된 오버레이 클러스터로 간주할 수 있습니다. 여기서 모든 원래 클러스터는 서로 독립적입니다. 티켓이 부여되었는지 여부에 관계없이 클러스터와 통신하는 **Booth** 서비스이며 **Pacemaker** 티켓 제약 조건을 기반으로 클러스터에서 리소스를 실행할지 여부를 결정하는 **Pacemaker**입니다. 즉, 티켓 관리자를 사용할 때 각 클러스터는 자체 리소스뿐만 아니라 공유 리소스를 실행할 수 있습니다. 예를 들어, 하나의 클러스터에서만 **A, B, C,** 리소스 **D, E, F**가 다른 클러스터에서만 실행되고 있는 리소스, 티켓에 따라 두 클러스터 중 하나에서 실행되는 **G** 및 **H** 리소스가 있을 수 있습니다. 또한 별도의 티켓에 따라 두 클러스터 중 하나에서 실행할 수 있는 추가 리소스 **J**도 있을 수 있습니다.

29.2. PACEMAKER를 사용하여 다중 사이트 클러스터 구성

다음 절차에 따라 **Booth** 티켓 관리자를 사용하는 다중 사이트 구성을 구성할 수 있습니다.

이러한 예제 명령은 다음 배열을 사용합니다.

- 클러스터 **1**은 **cluster1-node1** 및 **cluster1-node2**노드로 구성됩니다.
- 클러스터 **1**의 유동 **IP** 주소가 **192.168.11.100**입니다.

- 클러스터 2는 cluster2-node1 및 cluster2-node2로 구성됩니다.
- 클러스터 2에는 192.168.22.100의 유동 IP 주소가 할당되어 있습니다.
- 중재자 노드는 IP 주소가 192.168.99.100인 arbitrator-node 입니다.
- 이 구성이 사용하는 Booth 티켓의 이름은 apacheticket입니다.

이 예제 명령은 Apache 서비스의 클러스터 리소스가 각 클러스터에 대해 리소스 그룹 apachegroup의 일부로 구성되어 있다고 가정합니다. 각 클러스터의 Pacemaker 인스턴스가 독립적이지만 일반적인 장애 조치(failover) 시나리오이므로 리소스 및 리소스 그룹이 각 클러스터에서 동일하게 설정할 필요는 없습니다.

설정 절차에 언제든지 pcs booth config 명령을 입력하여 현재 노드 또는 클러스터의 booth 구성 또는 pcs booth status 명령을 입력하여 로컬 노드에 현재 booth 상태를 표시할 수 있습니다.

절차

1. 두 클러스터의 각 노드에 booth-site Booth 티켓 관리자 패키지를 설치합니다.

```
[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. 중재자 노드에 pcs,booth-core, booth-arbitrator 패키지를 설치합니다.

```
[root@arbitrator-node ~]# dnf install -y pcs booth-core booth-arbitrator
```

3. firewalld 데몬을 실행하는 경우 두 클러스터의 모든 노드에서 다음 명령을 실행하고 중재자 노드에서 다음 명령을 실행하여 Red Hat High Availability Add-On에 필요한 포트를 활성화합니다.

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

로컬 조건에 맞게 열려 있는 포트를 수정해야 할 수 있습니다. **Red Hat High-Availability Add-On**에 필요한 포트에 대한 자세한 내용은 고가용성 애드온의 포트 활성화를 참조하십시오.

4. 하나의 클러스터 노드 하나에 **Booth** 구성을 생성합니다. 각 클러스터에 지정한 주소 및 중재자의 주소는 **IP** 주소여야 합니다. 각 클러스터에 유동 **IP** 주소를 지정합니다.

```
[cluster1-node1 ~] # pcs booth setup sites 192.168.11.100 192.168.22.100 arbitrators
192.168.99.100
```

이 명령은 실행 중인 노드에 구성 파일 **/etc/booth/booth.conf** 및 **/etc/booth/booth.key** 를 생성합니다.

5. **Booth** 구성에 대한 티켓을 생성합니다. 이 티켓은 이 티켓이 클러스터에 부여된 경우에만 리소스를 실행할 수 있도록 허용하는 리소스 제약 조건을 정의하는 데 사용할 티켓입니다.

이 기본 장애 조치 구성 절차에서는 하나의 티켓만 사용하지만 각 티켓이 다른 리소스 또는 리소스와 연결된 더 복잡한 시나리오에 대한 티켓을 생성할 수 있습니다.

```
[cluster1-node1 ~] # pcs booth ticket add apacheticket
```

6. **Booth** 구성을 현재 클러스터의 모든 노드와 동기화합니다.

```
[cluster1-node1 ~] # pcs booth sync
```

7. 중재자 노드에서 **Booth** 구성을 **arbitrator**로 가져옵니다. 아직 수행하지 않은 경우 먼저 구성을 가져오는 노드에 **pcs** 를 인증해야 합니다.

```
[arbitrator-node ~] # pcs host auth cluster1-node1
[arbitrator-node ~] # pcs booth pull cluster1-node1
```

8. **Booth** 구성을 다른 클러스터로 가져와서 해당 클러스터의 모든 노드와 동기화합니다. 중재자와 마찬가지로 이전에 수행하지 않은 경우, 먼저 구성을 가져오는 노드에 **pcs** 를 인증해야 합니다.

```
[cluster2-node1 ~] # pcs host auth cluster1-node1
[cluster2-node1 ~] # pcs booth pull cluster1-node1
[cluster2-node1 ~] # pcs booth sync
```

9. *중재기에서 **Booth**를 시작하고 활성화합니다.*

참고

***Booth**가 해당 클러스터에서 **Pacemaker** 리소스로 실행되므로 클러스터 노드에서 **Booth**를 수동으로 시작하거나 활성화할 수 없습니다.*

```
[arbitrator-node ~] # pcs booth start
[arbitrator-node ~] # pcs booth enable
```

10. *각 클러스터에 할당된 유동 **IP** 주소를 사용하여 두 클러스터 사이트에서 클러스터 리소스로 실행되도록 **Booth**를 구성합니다. 그러면 **booth-ip** 및 **booth-service** 가 해당 그룹의 멤버로 포함된 리소스 그룹이 생성됩니다.*

```
[cluster1-node1 ~] # pcs booth create ip 192.168.11.100
[cluster2-node1 ~] # pcs booth create ip 192.168.22.100
```

11. *각 클러스터에 대해 정의한 리소스 그룹에 티켓 제약 조건을 추가합니다.*

```
[cluster1-node1 ~] # pcs constraint ticket add apacheticket apachegroup
[cluster2-node1 ~] # pcs constraint ticket add apacheticket apachegroup
```

다음 명령을 입력하여 현재 구성된 티켓 제약 조건을 표시할 수 있습니다.

```
pcs constraint ticket [show]
```

12. *이 설정에 대해 만든 티켓을 첫 번째 클러스터에 부여합니다.*

*티켓을 부여하기 전에 정의 된 티켓 제약 조건을 가질 필요가 없습니다. 처음에 클러스터에 대한 티켓이 부여되면 **pcs booth** 티켓 취소 명령을 사용하여 수동으로 이 값을 재정의하지 않는 한 **Booth**는 티켓 관리를 통과합니다. **pcs booth** 관리 명령에 대한 자세한 내용은 **pcs booth** 명령의 **PCS** 도움말 화면을 참조하십시오.*

```
[cluster1-node1 ~] # pcs booth ticket grant apacheticket
```

이 절차를 완료한 후에도 언제든지 티켓을 추가하거나 제거할 수 있습니다. 그러나 티켓을 추가하거나 제거한 후 설정 파일을 다른 노드 및 클러스터와 동기화하고 중재자와 이 프로세스에 표시된 대로 티켓을 부여해야 합니다.

***Booth** 구성 파일, 티켓 및 리소스를 정리 및 제거하는 데 사용할 수 있는 추가 **Booth** 관리 명령에 대한 자세한 내용은 **pcs booth** 명령의 **PCS** 도움말 화면을 참조하십시오.*

30장. 비COROSYNC 노드를 클러스터에 통합: PACEMAKER_REMOTE 서비스

pacemaker_remote 서비스를 사용하면 노드에서 corosync 를 실행하지 않고 클러스터가 실제 클러스터 노드인 것처럼 해당 리소스를 클러스터에 통합할 수 있습니다.

pacemaker_remote 서비스에서 제공하는 기능은 다음과 같습니다.

- pacemaker_remote 서비스를 사용하면 Red Hat 노드 32개 지원 제한을 초과할 수 있습니다.
- pacemaker_remote 서비스를 사용하면 가상 환경을 클러스터 리소스로 관리하고 가상 환경 내의 개별 서비스를 클러스터 리소스로 관리할 수도 있습니다.

다음 용어는 pacemaker_remote 서비스를 설명하는 데 사용됩니다.

- 클러스터 노드 - 고가용성 서비스(pacemaker 및 corosync)를 실행하는 노드입니다.
- 원격 노드 - corosync 클러스터 멤버십 없이도 pacemaker_remote 를 실행하여 클러스터에 원격으로 통합할 수 있습니다. 원격 노드는 ocf:pacemaker:remote 리소스 에이전트를 사용하는 클러스터 리소스로 구성됩니다.
- 게스트 노드 - pacemaker_remote 서비스를 실행하는 가상 게스트 노드 가상 게스트 리소스는 클러스터에서 관리하며 둘 다 클러스터에 의해 시작되고 원격 노드로 클러스터에 통합됩니다.
- pacemaker_remote - Pacemaker 클러스터 환경에서 원격 노드 및 KVM 게스트 노드에서 원격 애플리케이션 관리를 수행할 수 있는 서비스 데몬. 이 서비스는 corosync를 실행하지 않는 노드에서 원격으로 리소스를 관리할 수 있는 Pacemaker의 로컬 executor 데몬(pacemaker-execd)의 향상된 버전입니다.

pacemaker_remote 서비스를 실행하는 Pacemaker 클러스터에는 다음과 같은 특징이 있습니다.

- 원격 노드 및 게스트 노드는 pacemaker_remote 서비스를 실행합니다(가상 머신 측에 필요한 구성이 거의 없음).
-

클러스터 노드에서 실행되는 클러스터 스택(**pacemaker** 및 **corosync**)은 원격 노드의 **pacemaker_remote** 서비스에 연결하여 클러스터에 통합할 수 있습니다.

- 클러스터 노드에서 실행되는 클러스터 스택(**pacemaker** 및 **corosync**)은 게스트 노드를 시작하고 게스트 노드의 **pacemaker_remote** 서비스에 즉시 연결하여 클러스터에 통합할 수 있습니다.

클러스터 노드가 관리하는 클러스터 노드와 원격 노드 간의 주요 차이점은 원격 및 게스트 노드가 클러스터 스택을 실행하지 않는다는 것입니다. 즉, 원격 및 게스트 노드에 다음과 같은 제한 사항이 있습니다.

- 쿼럼에서 발생하지 않습니다.
- 펜싱 장치 작업을 실행하지 않습니다.
- 클러스터의 **Designated Controller (DC)**가 될 수 없습니다.
- **pcs** 명령의 전체 범위를 자체적으로 실행하지 않습니다.

반면 원격 노드 및 게스트 노드는 클러스터 스택과 관련된 확장성 제한에 바인딩되지 않습니다.

이러한 제한 사항 외에도 원격 및 게스트 노드는 리소스 관리와 관련하여 클러스터 노드와 마찬가지로 작동하며 원격 및 게스트 노드는 자체적으로 펜싱될 수 있습니다. 클러스터는 각 원격 및 게스트 노드에서 리소스를 완전히 관리하고 모니터링할 수 있습니다. 이에 대한 제약 조건을 빌드하거나 대기 상태로 설정하거나 **pcs** 명령으로 클러스터 노드에서 수행하는 기타 작업을 수행할 수 있습니다. 클러스터 노드에서와 마찬가지로 원격 및 게스트 노드가 클러스터 상태 출력에 표시됩니다.

30.1. PACEMAKER_REMOTE 노드의 호스트 및 게스트 인증

클러스터 노드와 **pacemaker_remote** 간 연결은 사전 공유 키(**PSK**) 암호화 및 **TCP**를 통한 인증(기본적으로 포트 **3121** 사용)과 함께 **TLS(Transport Layer Security)**를 사용하여 보호됩니다. 즉 **pacemaker_remote** 를 실행하는 노드와 클러스터 노드 모두 동일한 개인 키를 공유해야 합니다. 기본적으로 이 키는 클러스터 노드와 원격 노드의 **/etc/pacemaker/authkey** 에 배치해야 합니다.

pcs cluster node add-guest 명령은 게스트 노드의 **authkey** 를 설정하고 **pcs cluster node add-remote** 명령은 원격 노드의 **authkey** 를 설정합니다.

30.2. KVM 게스트 노드 구성

Pacemaker 게스트 노드는 **pacemaker_remote** 서비스를 실행하는 가상 게스트 노드입니다. 가상 게스트 노드는 클러스터에서 관리합니다.

30.2.1. 게스트 노드 리소스 옵션

게스트 노드로 작동하도록 가상 머신을 구성할 때 가상 머신을 관리하는 **VirtualDomain** 리소스를 생성합니다. **VirtualDomain** 리소스에 대해 설정할 수 있는 옵션에 대한 설명은 가상 도메인 리소스 옵션의 "가상 도메인 리소스에 대한 리소스 옵션 " 표를 참조하십시오.

metadata 옵션은 **VirtualDomain** 리소스 옵션 외에도 리소스를 게스트 노드로 정의하고 연결 매개 변수를 정의합니다. **pcs cluster node add-guest** 명령을 사용하여 이러한 리소스 옵션을 설정합니다. 다음 표에서는 이러한 메타데이터 옵션을 설명합니다.

표 **30.1. KVM** 리소스를 원격 노드로 설정하기 위한 메타데이터 옵션

필드	기본값	설명
remote-node	<none>	이 리소스에서 정의하는 게스트 노드의 이름입니다. 이 둘 다 리소스를 게스트 노드로 활성화하고 게스트 노드를 식별하는 데 사용되는 고유 이름을 정의합니다. *경고*: 이 값은 리소스 또는 노드 ID와 중복될 수 없습니다.
remote-port	3121	**pacemaker_remote**에 대한 게스트 연결에 사용할 사용자 정의 포트를 설정합니다.
remote-addr	**pcs host auth** 명령에 제공된 주소	연결할 IP 주소 또는 호스트 이름입니다.
remote-connect-timeout	60s	보류 중인 게스트 연결 전의 시간(시간)

30.2.2. 가상 머신을 게스트 노드로 통합

다음 절차에서는 **Pacemaker**에서 가상 머신을 시작하고 **libvirt** 및 **KVM** 가상 게스트를 사용하여 해당 머신을 게스트 노드로 통합할 수 있도록 수행하는 단계에 대한 간략한 개요입니다.

절차

1. *VirtualDomain 리소스를 구성합니다.*

2. *pacemaker_remote 패키지를 설치할 모든 가상 시스템에 다음 명령을 입력하고 pcsd 서비스를 시작한 후 시작 시 실행할 수 있도록 활성화한 다음 방화벽을 통해 TCP 포트 3121을 허용합니다.*

```
# 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. *각 가상 머신에 모든 노드에 알고 있어야 하는 정적 네트워크 주소와 고유한 호스트 이름을 지정합니다.*

4. *아직 수행하지 않은 경우 quest 노드로 통합하려는 노드에 pcs 를 인증합니다.*

```
# pcs host auth nodename
```

5. *다음 명령을 사용하여 기존 VirtualDomain 리소스를 게스트 노드로 변환합니다. 이 명령은 추가 중인 게스트 노드가 아닌 클러스터 노드에서 실행해야 합니다. 리소스를 변환하는 것 외에도 이 명령은 /etc/pacemaker/authkey 를 게스트 노드에 복사하고 guest 노드에서 pacemaker_remote 데몬을 시작하고 활성화합니다. 임의로 정의할 수 있는 게스트 노드의 노드 이름은 노드의 호스트 이름과 다를 수 있습니다.*

```
# pcs cluster node add-guest nodename resource_id [options]
```

6. *VirtualDomain 리소스를 생성한 후 클러스터의 다른 노드를 처리하는 것처럼 게스트 노드를 처리할 수 있습니다. 예를 들어, 리소스를 생성하고 클러스터 노드에서 실행되는 다음 명령과 같이 게스트 노드에서 실행할 리소스 제약 조건을 리소스에 배치할 수 있습니다. 게스트 노드를 그룹에 추가하여 스토리지 장치, 파일 시스템 및 VM을 그룹화할 수 있습니다.*

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

30.3. PACEMAKER 원격 노드 구성

원격 노드는 ocf:pacemaker:remote 를 리소스 에이전트로 사용하여 클러스터 리소스로 정의됩니다. pcs cluster node add-remote 명령을 사용하여 이 리소스를 생성합니다.

30.3.1. 원격 노드 리소스 옵션

다음 표에서는 원격 리소스에 대해 구성할 수 있는 리소스 옵션을 설명합니다.

표 30.2. 원격 노드의 리소스 옵션

필드	기본값	설명
reconnect_interval	0	원격 노드에 대한 활성 연결이 심각해진 후 원격 노드에 다시 연결하기 전에 대기하는 시간(초)입니다. 이 대기는 반복입니다. 대기 기간 후에 다시 연결하는 데 실패하면 대기 시간을 관찰한 후 새로운 다시 연결 시도가 수행됩니다. 이 옵션이 사용 중인 경우 Pacemaker에서 계속 연결을 시도하고 각 대기 간격 후에 원격 노드에 무기한 연결합니다.
server	**pcs host auth** 명령으로 지정된 주소	연결할 서버입니다. 이는 IP 주소 또는 호스트 이름일 수 있습니다.
port		연결할 TCP 포트입니다.

30.3.2. 원격 노드 구성 개요

*다음 절차에서는 **Pacemaker** 원격 노드를 구성하고 해당 노드를 기존 **Pacemaker** 클러스터 환경에 통합하기 위한 단계를 간략하게 설명합니다.*

절차

1. *원격 노드로 구성할 노드에서 로컬 방화벽을 통해 클러스터 관련 서비스를 허용합니다.*

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

참고

iptables 를 직접 사용하거나 firewalld 이외의 다른 방화벽 솔루션을 사용하는 경우 간단히 다음 포트를 엽니다. TCP 포트 2224 및 3121.

2. 원격 노드에 **pacemaker_remote** 데몬을 설치합니다.

```
# dnf install -y pacemaker-remote resource-agents pcs
```

3. 원격 노드에서 **pcsd** 를 시작하고 활성화합니다.

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

4. 아직 수행하지 않은 경우 원격 노드로 추가할 노드에 **pcs** 를 인증합니다.

```
# pcs host auth remote1
```

5. 다음 명령을 사용하여 원격 노드 리소스를 클러스터에 추가합니다. 이 명령은 관련된 모든 구성 파일을 새 노드에 동기화하고 노드를 시작한 다음 부팅 시 **pacemaker_remote** 를 시작하도록 구성합니다. 이 명령은 추가 중인 원격 노드가 아닌 클러스터 노드에서 실행해야 합니다.

```
# pcs cluster node add-remote remote1
```

6. 원격 리소스를 클러스터에 추가한 후 클러스터의 다른 노드를 처리하는 것처럼 원격 노드를 처리할 수 있습니다. 예를 들어, 리소스를 생성하고 클러스터 노드에서 실행되는 다음 명령과 같이 원격 노드에서 실행할 리소스에 리소스 제약 조건을 배치할 수 있습니다.

```
# pcs resource create webserver apache configfile=/etc/httpd/conf/httpd.conf op
monitor interval=30s
# pcs constraint location webserver prefers remote1
```

주의

리소스 그룹, 공동 배치 제약 조건 또는 순서 제약 조건에 원격 노드 연결 리소스를 포함하지 마십시오.

7. 원격 노드의 펜싱 리소스를 구성합니다. 클러스터 노드와 동일한 방식으로 원격 노드가 펜싱됩니다. 클러스터 노드와 마찬가지로 원격 노드에서 사용할 펜싱 리소스를 구성합니다. 그러나 원

격 노드는 펜싱 작업을 시작할 수 없습니다. 클러스터 노드만 다른 노드에 대해 펜싱 작업을 실제로 실행할 수 있습니다.

30.4. 기본 포트 위치 변경

Pacemaker 또는 pacemaker_remote 의 기본 포트 위치를 변경해야 하는 경우 이러한 데몬에 영향을 주는 PCMK_remote_port 환경 변수를 설정할 수 있습니다. 이 환경 변수는 다음과 같이 /etc/sysconfig/pacemaker 파일에 배치하여 활성화할 수 있습니다.

```
\#==#==# Pacemaker Remote
...
#
# Specify a custom port for Pacemaker Remote connections
PCMK_remote_port=3121
```

특정 게스트 노드 또는 원격 노드에서 사용하는 기본 포트를 변경할 때 해당 노드의 /etc/sysconfig/pacemaker 파일에서 PCMK_remote_ port 변수를 설정해야 하며, 게스트 노드 또는 원격 노드 연결을 생성하는 클러스터 리소스도 동일한 포트 번호(게스트 노드에 대해 remote-port 메타데이터 옵션 사용)로 구성해야 합니다.

30.5. PACEMAKER_REMOTE 노드로 시스템 업그레이드

Pacemaker _remote 서비스가 활성 Pacemaker 원격 노드에서 중지되면 노드를 중지하기 전에 클러스터에서 노드에서 리소스를 정상적으로 마이그레이션합니다. 이를 통해 클러스터에서 노드를 제거하지 않고도 소프트웨어 업그레이드 및 기타 일상적인 유지 관리 절차를 수행할 수 있습니다. pacemaker_remote 가 종료되면 즉시 클러스터가 다시 연결을 시도합니다. 리소스 모니터 시간 제한 내에서 pacemaker_remote 가 재시작되지 않으면 클러스터는 모니터 작업을 실패로 간주합니다.

pacemaker_remote 서비스가 활성 Pacemaker 원격 노드에서 중지될 때 오류를 모니터링하지 않으려면 pacemaker_remote 를 중지할 수 있는 시스템 관리를 수행하기 전에 다음 절차를 사용하여 클러스터를 중단할 수 있습니다.

절차

1. *pcs resource disable resourcename 명령을 사용하여 노드의 연결 리소스를 중지하여 모든 서비스를 노드에서 이동합니다. 연결 리소스는 원격 노드의 ocf:pacemaker:remote 리소스이거나 일반적으로 게스트 노드의 ocf:heartbeat:VirtualDomain 리소스입니다. 게스트 노드의 경우 이 명령은 VM도 중지되므로 모든 유지 관리를 수행하려면 VM을 클러스터 외부에서 시작해야 합니다(예: virsh를 사용).*

```
pcs resource disable resourcename
```

2. *필요한 유지 관리를 수행합니다.*

3. *노드를 클러스터로 반환할 준비가 되면* ***pcs resource enable*** *명령을 사용하여 리소스를 다시 활성화합니다.*

```
pcs resource enable resourcename
```

31장. 클러스터 유지보수 수행

클러스터 노드에서 유지보수를 수행하려면 해당 클러스터에서 실행되는 리소스 및 서비스를 중지하거나 이동해야 할 수 있습니다. 또는 서비스를 그대로 두지 않고 클러스터 소프트웨어를 중지해야 할 수도 있습니다. Pacemaker는 시스템 유지보수를 수행하는 다양한 방법을 제공합니다.

- 다른 노드의 해당 클러스터에서 실행 중인 서비스를 계속 제공하는 동안 클러스터에서 노드를 중지해야 하는 경우 클러스터 노드를 **standby** 모드가 되게 할 수 있습니다. 대기 모드에 있는 노드는 더 이상 리소스를 호스팅할 수 없습니다. 현재 노드에서 현재 활성화된 리소스가 다른 노드로 이동되거나 리소스를 실행할 수 있는 다른 노드가 없는 경우 중지됩니다. 대기 모드에 대한 자세한 내용은 노드 Putting a node into standby mode 를 참조하십시오.

- 해당 리소스를 중지하지 않고 현재 실행 중인 노드에서 개별 리소스를 이동해야 하는 경우 **pcs resource move** 명령을 사용하여 리소스를 다른 노드로 이동할 수 있습니다.

  pcs resource move 명령을 실행하면 리소스에 제약 조건이 추가되어 현재 실행 중인 노드에서 실행되지 않습니다. 리소스를 다시 이동할 준비가 되면 **pcs resource clear** 또는 **pcs constraint delete** 명령을 실행하여 제약 조건을 제거할 수 있습니다. 이 작업은 리소스가 원래 노드로 다시 이동하는 것은 아니지만, 해당 시점에서 리소스를 실행할 수 있는 위치는 처음에 리소스를 구성한 방법에 따라 달라집니다. **pcs resource relocate run** 명령을 사용하여 리소스를 기본 노드로 재배치할 수 있습니다.

- 실행 중인 리소스를 완전히 중지하고 클러스터가 다시 시작되지 않도록 하려면 **pcs resource disable** 명령을 사용할 수 있습니다. **pcs resource disable** 명령에 대한 자세한 내용은 클러스터 리소스 비활성화, 활성화 및 금지를 참조하십시오.

- **Pacemaker**에서 리소스에 대한 작업을 수행하지 못하도록 하려면 (예: 리소스에서 유지 관리를 수행하는 동안 복구 작업을 비활성화하거나 **/etc/sysconfig/pacemaker** 설정을 다시 로드해야 하는 경우), 다음과 같이 **pcs resource unmanage** 명령을 사용합니다. **Pacemaker** 원격 연결 리소스는 관리되지 않아야 합니다.

- 서비스를 시작하거나 중지하지 않는 상태로 클러스터를 배치해야 하는 경우 **maintenance-mode** 클러스터 속성을 설정할 수 있습니다. 클러스터를 유지보수 모드로 전환하면 모든 리소스가 자동으로 관리되지 않습니다. 클러스터를 유지 관리 모드로 배치하는 방법에 대한 자세한 내용은 클러스터 Putting a cluster in maintenance mode 를 참조하십시오.

- **RHEL** 고가용성 및 복구 스토리지 애드온을 구성하는 패키지를 업데이트해야 하는 경우 RHEL 고가용성 클러스터를 업데이트하는 데 요약된 대로 한 번에 또는 전체 클러스터에서 패키지를 업데이트할 수 있습니다.

-

Pacemaker 원격 노드에서 유지보수를 수행해야 하는 경우 원격 노드 및 게스트 노드 업그레이드에 설명된 대로 원격 노드 리소스를 비활성화하여 클러스터에서 해당 노드를 제거할 수 있습니다.

- **RHEL** 클러스터에서 **VM**을 마이그레이션해야 하는 경우 먼저 클러스터에서 노드를 제거한 다음 마이그레이션을 수행한 후 클러스터를 다시 시작합니다. **RHEL** 클러스터에서 **VM** 마이그레이션에 설명된 대로 **VM**에서 클러스터 서비스를 중지해야 합니다.

31.1. 노드를 대기 모드로 전환

클러스터 노드가 대기 모드이면 노드가 더 이상 리소스를 호스팅할 수 없습니다. 현재 노드에서 활성 상태인 모든 리소스가 다른 노드로 이동합니다.

다음 명령은 지정된 노드를 대기 모드로 전환합니다. **--all** 을 지정하면 이 명령을 실행하면 모든 노드가 **standby** 모드가 됩니다.

이 명령은 리소스의 패키지를 업데이트할 때 사용할 수 있습니다. 구성을 테스트할 때 이 명령을 사용하여 실제로 노드를 종료하지 않고 복구를 시뮬레이션할 수도 있습니다.

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

다음 명령은 대기 모드에서 지정된 노드를 제거합니다. 이 명령을 실행한 후 지정된 노드가 리소스를 호스팅할 수 있습니다. **--all** 을 지정하면 이 명령은 대기 모드에서 모든 노드를 제거합니다.

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

pcs node standby 명령을 실행하면 표시된 노드에서 리소스가 실행되지 않습니다. **pcs node unstandby** 명령을 실행하면 표시된 노드에서 리소스를 실행할 수 있습니다. 이는 리소스가 표시된 노드로 다시 이동하는 것은 아닙니다. 해당 시점에서 리소스를 실행할 수 있는 위치는 처음에 리소스를 구성한 방법에 따라 달라집니다.

31.2. 수동으로 클러스터 리소스 이동

클러스터를 재정의하고 리소스를 현재 위치에서 이동하도록 강제할 수 있습니다. 이 작업을 수행하려는 경우 두 가지 경우가 있습니다.

- 노드가 유지보수 상태에 있고 해당 노드에서 실행 중인 모든 리소스를 다른 노드로 이동해야 합니다.

- 개별적으로 지정된 리소스를 이동해야 하는 경우

노드에서 실행 중인 모든 리소스를 다른 노드로 이동하려면 노드를 **standby** 모드가 되게 합니다.

다음 방법 중 하나로 개별적으로 지정된 리소스를 이동할 수 있습니다.

- **pcs resource move** 명령을 사용하여 리소스를 현재 실행 중인 노드에서 이동할 수 있습니다.
- **pcs resource relocate run** 명령을 사용하여 현재 클러스터 상태, 제약 조건, 리소스 위치 및 기타 설정에 따라 리소스를 기본 노드로 이동할 수 있습니다.

31.2.1. 현재 노드에서 리소스 이동

리소스를 현재 실행 중인 노드에서 이동하려면 다음 명령을 사용하여 정의된 리소스의 **resource_id** 를 지정합니다. 이동 중인 리소스를 실행할 노드를 지정하려면 **destination_node** 를 지정합니다.

```
pcs resource move resource_id [destination_node] [--promoted] [--strict] [--wait[=n]]
```

pcs resource move 명령을 실행하면 리소스에 제약 조건이 추가되어 현재 실행 중인 노드에서 실행되지 않습니다. 기본적으로 명령이 생성하는 위치 제한 조건은 리소스가 이동되면 자동으로 제거됩니다. 제약 조건을 제거하면 리소스의 **resource-stickiness** 값이 **0**인 경우 **pcs resource move** 명령이 실패할 수 있으므로 리소스가 원래 노드로 다시 이동할 수 있습니다. 리소스를 이동하고 결과 제약 조건을 그대로 두려면 **pcs resource move-with-constraint** 명령을 사용합니다.

pcs resource move 명령의 **--promoted** 매개변수를 지정하면 제약 조건이 승격된 리소스 인스턴스에만 적용됩니다.

pcs resource move 명령의 **--strict** 매개변수를 지정하면 명령에 지정된 리소스 이외의 다른 리소스가 영향을 받는 경우 명령이 실패합니다.

리소스가 아직 시작되지 않은 경우 **0**을 반환하기 전에 대상 노드에서 리소스가 시작될 때까지 대기하는 시간(초)을 나타내는 **--wait[=n]** 매개변수를 선택적으로 구성할 수 있습니다. **n**을 지정하지 않으면 기본값은 **60**분입니다.

31.2.2. 리소스를 기본 노드로 이동

장애 조치(failover) 또는 관리자가 수동으로 노드를 이동하게 되면 장애 조치(failover)를 유발한 상황이 수정된 경우에도 리소스가 원래 노드로 다시 이동되지는 않습니다.After a resource has moved, either due to a failover or to an administrator manually moving the node, it will not necessarily move back to its original node even after the situation that caused the failover have been corrected. 리소스를 기본 노드로 재배치하려면 다음 명령을 사용합니다. 기본 노드는 현재 클러스터 상태, 제약 조건, 리소스 위치 및 기타 설정으로 결정되며 시간이 지남에 따라 변경될 수 있습니다.

```
pcs resource relocate run [resource1] [resource2] ...
```

리소스를 지정하지 않으면 모든 리소스가 기본 노드로 재배치됩니다.

이 명령은 리소스 고정성을 무시하는 동안 각 리소스에 대해 기본 노드를 계산합니다. 기본 노드를 계산한 후 위치 제한 조건을 생성하여 리소스가 기본 노드로 이동합니다. 리소스가 이동되면 제약 조건이 자동으로 삭제됩니다. **pcs resource relocate run** 명령으로 생성된 모든 제약 조건을 제거하려면 **pcs resource relocate clear** 명령을 입력합니다. 현재 리소스 상태와 리소스 고정을 무시하는 최적의 노드를 표시하려면 **pcs resource relocate show** 명령을 입력합니다.

31.3. 클러스터 리소스 비활성화, 활성화 및 금지

pcs resource move 및 **pcs resource relocate** 명령 외에도 클러스터 리소스의 동작을 제어하는 데 사용할 수 있는 다른 다양한 명령이 있습니다.

클러스터 리소스 비활성화

다음 명령을 사용하여 실행 중인 리소스를 수동으로 중지하고 클러스터가 다시 시작되지 않도록 할 수 있습니다. 나머지 구성(**constraints, options, failures** 등)에 따라 리소스가 시작될 수 있습니다. **--wait** 옵션을 지정하면 **pcs** 가 리소스가 중지될 때까지 **'n'**초까지 기다린 다음 리소스가 중지되지 않은 경우 **0**을 반환하거나 **1**을 반환합니다. **'n'**을 지정하지 않으면 기본값은 **60**분입니다.

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

리소스를 비활성화해도 다른 리소스에 영향을 미치지 않는 경우에만 리소스를 비활성화하도록 지정할 수 있습니다. 복잡한 리소스 관계가 설정될 때 이에 의해 수행되는 것이 불가능한지 확인합니다.

- **pcs resource disable --simulate** 명령은 클러스터 구성을 변경하지 않고 리소스를 비활성화하는 영향을 보여줍니다.
- **pcs resource disable --safe** 명령은 한 노드에서 다른 노드로 마이그레이션하는 것과 같은 어떤 방식으로든 다른 리소스가 영향을 받지 않는 경우에만 리소스를 비활성화합니다. **pcs**

resource safe-disable 명령은 pcs resource disable --safe 명령의 별칭입니다.

- *pcs resource disable --safe --no-strict 명령은 다른 리소스가 중지되거나 시연되지 않는 경우에만 리소스를 비활성화합니다.*

pcs resource disable --safe 명령에 --brief 옵션을 지정하여 오류만 출력할 수 있습니다. safe disable 작업에 영향을 받는 리소스 ID가 포함된 경우 pcs resource disable --safe 명령이 생성한다고 오류 보고합니다. 리소스를 비활성화하여 영향을 받는 리소스의 리소스 ID만 알아야 하는 경우 전체 시뮬레이션 결과를 제공하지 않는 --brief 옵션을 사용합니다.

클러스터 리소스 활성화

다음 명령을 사용하여 클러스터가 리소스를 시작할 수 있도록 허용합니다. 나머지 구성에 따라 리소스가 중지된 상태로 유지될 수 있습니다. --wait 옵션을 지정하면 pcs 가 리소스가 시작될 때까지 'n'초까지 기다린 다음 리소스가 시작되지 않으면 0을 반환하거나 리소스가 시작되지 않은 경우 1을 반환합니다. 'n'을 지정하지 않으면 기본값은 60분입니다.

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

특정 노드에서 리소스가 실행되지 않도록 방지

다음 명령을 사용하여 지정된 노드에서 또는 노드가 지정되지 않은 경우 현재 노드에서 리소스가 실행되지 않도록 합니다.

```
pcs resource ban resource_id [node] [--promoted] [lifetime=lifetime] [--wait[=n]]
```

pcs resource forbidden 명령을 실행하면 지정된 노드에서 실행되지 않도록 리소스에 -INFINITY 위치 제약 조건이 추가됩니다. pcs resource clear 또는 pcs constraint delete 명령을 실행하여 제약 조건을 제거할 수 있습니다. 이는 리소스가 표시된 노드로 다시 이동하는 것은 아닙니다. 해당 시점에서 리소스를 실행할 수 있는 위치는 처음에 리소스를 구성한 방법에 따라 달라집니다.

pcs resource prohibited 명령 의 --promoted 매개변수를 지정하는 경우 제약 조건의 범위가 승격 된 역할로 제한되며 resource_id 대신 promotetable_id 를 지정해야 합니다.

제약 조건이 남아 있어야 하는 기간을 표시하도록 pcs 리소스 금지 명령에 대한 수명 매개 변수를 선택적으로 구성할 수 있습니다.

리소스가 아직 시작되지 않은 경우 0을 반환하기 전에 대상 노드에서 리소스가 시작될 때까지 대기하는 시간(초)을 나타내는 --wait[=n] 매개변수를 선택적으로 구성할 수 있습니다. n을 지정하지 않으면 기본 리소스 시간 초과가 사용됩니다.

현재 노드에서 리소스를 강제로 시작

pcs resource 명령의 **debug-start** 매개변수를 사용하여 지정된 리소스가 현재 노드에서 시작되도록 하고 클러스터 권장 사항을 무시하고 리소스를 시작할 때 출력을 출력하도록 합니다. 이는 주로 리소스를 디버깅하는 데 사용됩니다. 클러스터에서 리소스를 시작하는 것은 항상 **pcs** 명령으로 직접 **Pacemaker**에서 수행합니다. 리소스를 시작하지 않는 경우 일반적으로 리소스의 잘못된 구성(시스템 로그에서 디버그), 리소스를 시작하지 못하게 하는 제약 조건 또는 비활성화 중인 리소스 때문입니다. 이 명령을 사용하여 리소스 구성을 테스트할 수 있지만 일반적으로 클러스터에서 리소스를 시작하는 데 사용해서는 안 됩니다.

debug-start 명령의 형식은 다음과 같습니다.

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

31.4. 관리되지 않는 모드로 리소스 설정

리소스가 관리되지 않는 모드이면 리소스가 여전히 구성에 있지만 **Pacemaker**에서 리소스를 관리하지 않습니다.

다음 명령은 표시된 리소스를 관리되지 않는 모드로 설정합니다.

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

다음 명령은 리소스를 기본 상태인 관리 모드로 설정합니다.

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

pcs resource manage 또는 **pcs resource unmanage** 명령을 사용하여 리소스 그룹의 이름을 지정할 수 있습니다. 명령은 그룹의 모든 리소스에 대해 작동하므로 그룹의 모든 리소스를 단일 명령으로 관리 또는 관리 되지 않는 모드로 설정한 다음 포함된 리소스를 개별적으로 관리할 수 있습니다.

31.5. 클러스터를 유지보수 모드로 전환

클러스터가 유지보수 모드에 있을 때 클러스터가 달리 지시될 때까지 서비스를 시작하거나 중지하지 않습니다. 유지 관리 모드가 완료되면 클러스터는 서비스의 현재 상태에 대한 온전한 확인을 수행한 다음 필요한 서비스를 중지하거나 시작합니다.

클러스터를 유지보수 모드로 전환하려면 다음 명령을 사용하여 **maintenance-mode** 클러스터 속성을 **true** 로 설정합니다.

```
# pcs property set maintenance-mode=true
```

유지보수 모드에서 클러스터를 제거하려면 다음 명령을 사용하여 **maintenance-mode** 클러스터 속성을 **false** 로 설정합니다.

```
# pcs property set maintenance-mode=false
```

다음 명령을 사용하여 구성에서 클러스터 속성을 제거할 수 있습니다.

```
pcs property unset property
```

또는 **pcs property set** 명령의 **value** 필드를 비워 구성에서 클러스터 속성을 제거할 수 있습니다. 이렇게 하면 해당 속성이 기본값으로 복원됩니다. 예를 들어, **symmetric-cluster** 속성을 이전에 **false** 로 설정한 경우 다음 명령은 구성에서 설정한 값을 제거하고 **symmetric-cluster** 값을 기본값인 **true** 로 복원합니다.

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

31.6. RHEL 고가용성 클러스터 업데이트

RHEL 고가용성 및 복구 스토리지 애드온을 개별적으로 또는 전체적으로 구성하는 패키지를 두 가지 일반적인 방법 중 하나로 수행할 수 있습니다.

- 롤링 업데이트: 서비스에서 한 번에 하나의 노드를 제거하고 소프트웨어를 업데이트한 다음 클러스터에 다시 통합합니다. 이를 통해 클러스터는 각 노드가 업데이트되는 동안 서비스 및 리소스를 계속 제공할 수 있습니다.
- 전체 클러스터 업데이트: 전체 클러스터를 중지하고 업데이트를 모든 노드에 적용한 다음 클러스터를 다시 시작합니다.

주의

Red Hat Enterprise Linux High Availability 및 **Resilient Storage** 클러스터에 대한 소프트웨어 업데이트 절차를 수행할 때는 업데이트가 시작되기 전에 클러스터의 활성 멤버가 아닌 노드가 클러스터의 활성 멤버가 아닌지 확인하는 것이 중요합니다.

이러한 각 방법과 업데이트에 대한 절차에 대한 자세한 내용은 **RHEL** 고가용성 또는 복구 스토리지 클러스터에 소프트웨어 업데이트 적용 방법을 참조하십시오.

31.7. 원격 노드 및 게스트 노드 업그레이드

pacemaker_remote 서비스가 활성 원격 노드 또는 게스트 노드에서 중지되면 클러스터는 노드를 중지하기 전에 노드에서 리소스를 정상적으로 마이그레이션합니다. 이를 통해 클러스터에서 노드를 제거하지 않고도 소프트웨어 업그레이드 및 기타 일상적인 유지 관리 절차를 수행할 수 있습니다. **pacemaker_remote** 가 종료되면 즉시 클러스터가 다시 연결을 시도합니다. 리소스 모니터 시간 제한 내에서 **pacemaker_remote** 가 재시작되지 않으면 클러스터는 모니터 작업을 실패로 간주합니다.

pacemaker_remote 서비스가 활성 **Pacemaker** 원격 노드에서 중지될 때 오류를 모니터링하지 않으려면 **pacemaker_remote** 를 중지할 수 있는 시스템 관리를 수행하기 전에 다음 절차를 사용하여 클러스터를 중단할 수 있습니다.

절차

1. **pcs resource disable resourcename** 명령을 사용하여 노드의 연결 리소스를 중지하여 모든 서비스를 노드에서 이동합니다. 연결 리소스는 원격 노드의 **ocf:pacemaker:remote** 리소스이거나 일반적으로 게스트 노드의 **ocf:heartbeat:VirtualDomain** 리소스입니다. 게스트 노드의 경우 이 명령은 **VM**도 중지되므로 모든 유지 관리를 수행하려면 **VM**을 클러스터 외부에서 시작해야 합니다(예: **virsh**를 사용).

```
pcs resource disable resourcename
```

2. 필요한 유지 관리를 수행합니다.

3. 노드를 클러스터로 반환할 준비가 되면 **pcs resource enable** 명령을 사용하여 리소스를 다시 활성화합니다.

```
pcs resource enable resourcename
```

31.8. RHEL 클러스터에서 VM 마이그레이션

Red Hat은 RHEL 고가용성 클러스터에 대한 지원 정책인 가상화된 클러스터 멤버를 보유한 일반 조건인 하이퍼바이저 또는 호스트에서 활성 클러스터 노드의 실시간 마이그레이션을 지원하지 않습니다. 실시간 마이그레이션을 수행해야 하는 경우 먼저 VM에서 클러스터 서비스를 중지하여 클러스터에서 노드를 제거한 다음 마이그레이션을 수행한 후 클러스터를 다시 시작해야 합니다. 다음 단계에서는 클러스터에서 VM을 제거하고, VM을 마이그레이션하며, VM을 클러스터로 복원하는 절차를 간략하게 설명합니다.

다음 단계에서는 클러스터에서 VM을 제거하고, VM을 마이그레이션하며, VM을 클러스터로 복원하는 절차를 간략하게 설명합니다.

이 절차는 특정 예방 조치 없이 실시간 마이그레이션될 수 있는 클러스터 리소스(게스트 노드로 사용되는 VM 포함)로 관리되는 VM에 전체 클러스터 노드로 사용되는 VM에 적용됩니다. RHEL High Availability 및 Resilient Storage 애드온을 개별적으로 또는 전체적으로 구성하는 패키지를 업데이트하는 데 필요한 전체 프로시저에 대한 일반 정보는 RHEL High Availability 또는 Resilient Storage Cluster에 소프트웨어 업데이트 적용을 위한 권장 사례를 참조하십시오.

참고

이 절차를 수행하기 전에 클러스터 노드 제거의 클러스터 쿼럼에 미치는 영향을 고려하십시오. 예를 들어 3-노드 클러스터가 있고 하나의 노드를 제거하는 경우 클러스터의 노드 장애를 견딜 수 없습니다. 이는 3-노드 클러스터의 한 노드가 이미 다운된 경우 두 번째 노드를 제거하면 쿼럼이 손실되기 때문입니다.

절차

1. 마이그레이션할 VM에서 실행 중인 리소스 또는 소프트웨어를 중지하거나 이동하기 전에 준비를 수행해야 합니다.

2. VM에서 다음 명령을 실행하여 VM에서 클러스터 소프트웨어를 중지합니다.

```
# pcs cluster stop
```

3. VM의 실시간 마이그레이션을 수행합니다.

4. VM에서 클러스터 서비스를 시작합니다.

```
# pcs cluster start
```

31.9. UUID로 클러스터 확인

Red Hat Enterprise Linux 9.1부터 클러스터를 생성할 때 관련 **UUID**가 있습니다. 클러스터 이름은 고유한 클러스터 식별자가 아니므로 이름이 동일한 여러 클러스터를 관리하는 구성 관리 데이터베이스와 같은 타사 툴은 **UUID**를 사용하여 클러스터를 고유하게 식별할 수 있습니다. 출력에 클러스터 **UUID**가 포함된 **pcs** 클러스터 구성 **[show]** 명령을 사용하여 현재 클러스터 **UUID**를 표시할 수 있습니다.

기존 클러스터에 **UUID**를 추가하려면 다음 명령을 실행합니다.

```
# pcs cluster config uuid generate
```

기존 **UUID**를 사용하여 클러스터의 **UUID**를 재생성하려면 다음 명령을 실행합니다.

```
# pcs cluster config uuid generate --force
```

32장. 재해 복구 클러스터 구성

고가용성 클러스터에 재해 복구를 제공하는 한 가지 방법은 두 개의 클러스터를 구성하는 것입니다. 그런 다음 하나의 클러스터를 기본 사이트 클러스터로 구성하고 두 번째 클러스터를 재해 복구 클러스터로 구성할 수 있습니다.

정상적인 상황에서는 기본 클러스터가 프로덕션 모드에서 리소스를 실행합니다. 재해 복구 클러스터에는 모든 리소스가 구성되어 있으며 데모 모드에서 실행 중이거나 실행되지 않습니다. 예를 들어, 승격 모드의 기본 클러스터에서 실행 중인 데이터베이스가 있고 데모 모드에서 재해 복구 클러스터에서 실행 중인 데이터베이스가 있을 수 있습니다. 이 설정의 데이터베이스는 데이터가 주에서 재해 복구 사이트로 동기화되도록 구성됩니다. 이 작업은 **pcs** 명령 인터페이스를 거치지 않고 데이터베이스 구성 자체를 통해 수행됩니다.

기본 클러스터가 중단되면 **pcs** 명령 인터페이스를 사용하여 리소스를 재해 복구 사이트로 수동으로 실패할 수 있습니다. 그런 다음 재해 사이트에 로그인하고 해당 리소스를 승격 및 시작할 수 있습니다. 기본 클러스터가 복구되면 **pcs** 명령 인터페이스를 사용하여 리소스를 기본 사이트로 수동으로 이동할 수 있습니다.

pcs 명령을 사용하여 한 사이트의 단일 노드에서 기본 및 재해 복구 사이트 클러스터의 상태를 표시할 수 있습니다.

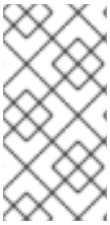
32.1. 재해 복구 클러스터에 대한 고려 사항

pcs 명령 인터페이스를 사용하여 재해 복구 사이트를 계획 및 구성할 때 다음 사항을 고려하십시오.

- 재해 복구 사이트는 클러스터여야 합니다. 이렇게 하면 기본 사이트와 동일한 도구 및 유사한 절차를 사용하여 구성할 수 있습니다.
- 기본 및 재해 복구 클러스터는 독립적인 **pcs cluster setup** 명령으로 생성됩니다.
- 데이터가 동기화되고 장애 조치가 가능하도록 클러스터 및 해당 리소스를 구성해야 합니다.
- 복구 사이트의 클러스터 노드는 기본 사이트의 노드와 동일한 이름을 가질 수 없습니다.
- **pcs** 명령을 실행할 노드의 두 클러스터에 있는 각 노드에 대해 **pcs** 사용자 **hacluster** 를 인증해야 합니다.

32.2. 복구 클러스터 상태 표시

두 클러스터의 상태를 표시할 수 있도록 주 및 재해 복구 클러스터를 구성하려면 다음 절차를 수행하십시오.



참고

재해 복구 클러스터를 설정하면 리소스를 자동으로 구성하거나 데이터를 복제하지 않습니다. 이러한 항목은 사용자가 수동으로 구성해야 합니다.

이 예제에서는 다음을 수행합니다.

- 기본 클러스터 이름은 **PrimarySite** 로 지정되며 **z1.example.com** 노드와 **z2.example.com** 으로 구성됩니다.
- 재해 복구 사이트 클러스터의 이름은 **DRsite** 이며 **z3.example.com** 및 **z4.example.com** 노드로 구성됩니다.

이 예에서는 리소스 또는 펜싱이 구성되지 않은 기본 클러스터를 설정합니다.

절차

1. 두 클러스터에 모두 사용할 모든 노드를 인증합니다.

```
[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. 기본 클러스터로 사용할 클러스터를 생성하고 클러스터의 클러스터 서비스를 시작합니다.

```
[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. 재해 복구 클러스터로 사용할 클러스터를 생성하고 클러스터의 클러스터 서비스를 시작합니다.

```
[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. 기본 클러스터의 노드에서 두 번째 클러스터를 복구 사이트로 설정합니다. 복구 사이트는 노드 중 하나의 이름으로 정의됩니다.

```
[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. 재해 복구 구성을 확인합니다.

```
[root@z1 ~]# pcs dr config
Local site:
  Role: Primary
Remote site:
  Role: Recovery
  Nodes:
    z3.example.com
    z4.example.com
```

6. 기본 클러스터의 노드와 기본 클러스터의 재해 복구 클러스터의 상태를 확인합니다.

```
[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
```

재해 복구 구성에 대한 추가 표시 옵션은 **pcs dr** *명령의 도움말 화면을 참조하십시오.*

33장. 리소스 에이전트 OCF 반환 코드 해석

Pacemaker 리소스 에이전트는 OVS(Open Cluster Framework) 리소스 에이전트 API를 준수합니다. 다음 표에서는 OCF 반환 코드 및 Pacemaker에서 해석하는 방법을 설명합니다.

에이전트가 코드를 반환할 때 클러스터가 수행하는 첫 번째 작업은 예상되는 결과에 대해 반환 코드를 확인하는 것입니다. 결과가 예상 값과 일치하지 않으면 작업이 실패한 것으로 간주되고 복구 작업이 시작됩니다.

모든 호출의 경우 리소스 에이전트는 호출된 작업의 결과를 호출자로 알리는 정의된 반환 코드로 종료해야 합니다.

다음 표에 설명된 대로 실패 복구에는 세 가지 유형이 있습니다.

표 33.1. 클러스터에서 수행하는 복구 유형

유형	설명	클러스터에 의한 작업
소프트	일시적인 오류가 발생했습니다.	리소스를 다시 시작하거나 새 위치로 이동합니다.
하드	현재 노드에 고유할 수 있는 일시적이지 않은 오류입니다.	리소스를 다른 위치로 이동하고 현재 노드에서 재시도하지 못하도록 합니다.
치명적	발생하는 모든 클러스터 노드에 공통된 일시적이지 않은 오류(예: 잘못된 구성이 지정됨).	리소스를 중지하고 클러스터 노드에서 시작되지 않도록 합니다.

다음 표에서는 OCF 반환 코드를 제공하고 실패 코드가 수신될 때 클러스터가 시작될 때 클러스터가 시작됩니다.0이 예상 반환 값이 아닌 경우 0이 실패한 것으로 간주되는 경우 0을 반환하는 작업도 0(OCF 별칭 OCF_SUCCESS)으로 간주될 수 있습니다.

표 33.2. OCF 반환 코드

코드 반환	OCF 레이블	설명
0	**OCF_SUCCESS**	* 작업이 성공적으로 완료되었습니다. 성공적인 start, stop, promote 및 demote 명령에 대해 예상되는 반환 코드입니다. * 예기치 않은 경우 유형: soft

코드 반환	OCF 레이블	설명
1	**OCF_ERR_GENERIC**	*이 작업은 일반적인 오류를 반환했습니다. * 유형: soft * 리소스 관리자는 리소스를 복구하거나 새 위치로 이동하려고 시도합니다.
2	**OCF_ERR_ARGS**	* 리소스의 구성이 이 컴퓨터에서 유효하지 않습니다. 예를 들어 노드에서 찾을 수 없는 위치를 참조합니다. * 유형: hard * 리소스 관리자가 리소스를 다른 위치로 이동하여 현재 노드에서 다시 수행되지 않도록 합니다.
3	**OCF_ERR_UNIMPLEMENTED**	* 요청된 작업이 구현되지 않았습니다. * 유형: hard
4	**OCF_ERR_PERM**	* 리소스 에이전트에 작업을 완료하는 데 충분한 권한이 없습니다. 예를 들어 에이전트에서 특정 파일을 열 수 없거나 특정 소켓에서 수신 대기하거나 디렉토리에 쓸 수 없기 때문일 수 있습니다. * 유형: hard * 별도로 구성하지 않는 한 리소스 관리자는 다른 노드에서 리소스를 다시 시작하여 이 오류로 인해 실패한 리소스를 복구하려고 합니다(권한 문제가 없을 수 있음).
5	**OCF_ERR_INSTALLED**	* 작업이 실행된 노드에서 필수 구성 요소가 누락되었습니다. 이는 필수 바이너리가 실행 불가능하거나, 필수 구성 파일을 읽을 수 없기 때문일 수 있습니다. * 유형: hard * 별도로 구성하지 않는 한 리소스 관리자는 다른 노드에서 리소스를 다시 시작하여 이 오류로 인해 실패한 리소스를 복구하려고 시도합니다(필수 파일 또는 바이너리가 있을 수 있음).
6	**OCF_ERR_CONFIGURED**	* 로컬 노드의 리소스 구성이 유효하지 않습니다. * 유형: fatal *이 코드가 반환되면 Pacemaker는 서비스 구성이 다른 노드에서 유효하더라도 클러스터의 노드에서 리소스가 실행되지 않도록 합니다.

코드 반환	OCF 레이블	설명
7	**OCF_NOT_RUNNING**	* 리소스가 안전하게 중지됩니다. 즉, 리소스가 정상적으로 종료되었거나 시작되지 않았음을 의미합니다. * 예기치 않은 경우 유형: soft * 클러스터는 작업을 위해 이를 반환하는 리소스를 중지하지 않습니다.
8	**OCF_RUNNING_PROMOTED**	* 리소스가 승격된 역할에서 실행되고 있습니다. * 예기치 않은 경우 유형: soft
9	**OCF_FAILED_PROMOTED**	* 리소스는 승격된 역할에서(또는 있을 수 있음)이지만 실패했습니다. * 유형: soft * 리소스는 강등, 중지 후 다시 시작(및 승격될 수 있음)됩니다.
190		* 서비스가 제대로 작동하는 것으로 밝혀지지만, 이러한 조건에서 향후 오류가 발생할 가능성이 높습니다.
191		* 리소스 에이전트는 역할을 지원하고 서비스는 승격된 역할에서 올바르게 활성화되지만, 이러한 조건에서 향후 오류가 발생할 가능성이 높습니다.
기타	해당 없음	사용자 지정 오류 코드.

34장. IBM Z/VM 인스턴스를 클러스터 멤버로 사용하여 RED HAT HIGH AVAILABILITY 클러스터 구성

Red Hat은 z/VM 가상 시스템에서 실행되는 Red Hat High Availability 클러스터를 설계, 구성 및 관리할 때 유용할 수 있는 여러 기사를 제공합니다.

- RHEL High Availability Clusters용 설계 가이드 - IBM z/VM Instances as Cluster Members
- RHEL 고가용성 클러스터용 관리 절차 - RHEL 7 또는 8 IBM z Systems 클러스터 멤버용 fence_zvmip로 z/VM SMAPI Fencing 구성
- IBM z Systems의 RHEL 고가용성 클러스터 노드인 STONITH 장치 시간 초과
- RHEL 고가용성 클러스터에 대한 관리자 작업 - IBM z Systems 사용자가 사용할 수 있도록 dasd 스토리지 장치 준비

Red Hat High Availability 클러스터를 일반적으로 설계할 때 유용한 문서도 확인할 수 있습니다.

- RHEL 고가용성 클러스터에 대한 지원 정책
- RHEL 고가용성 클러스터의 개념 탐색 - Fencing/STONITH