Red Hat Enterprise Linux 8

논리 볼륨 구성 및 관리

RHEL에서 LVM 구성 및 관리

Last Updated: 2024-06-26

Red Hat Enterprise Linux 8 논리 볼륨 구성 및 관리

RHEL에서 LVM 구성 및 관리

법적 공지

초록

LVM(Logical Volume Management)은 물리 스토리지에 대한 추상화 계층을 생성하여 파일 시스템, 데이터베이스 또는 애플리케이션에서 사용할 수 있는 가상 블록 스토리지 장치인 논리 스토리지 볼륨을 생성합니다. PV(물리 볼륨)는 파티션 또는 전체 디스크입니다. 이러한 PV를 사용하면 볼륨 그룹(VG)을 생성하여 사용 가능한 스토리지에서 논리 볼륨(LV)의 디스크 공간 풀을 생성할 수 있습니다. 물리 볼륨을 볼륨 그룹에 결합하여 논리 볼륨(LV)을 생성할 수 있습니다. LV는 물리 스토리지를 사용하는 것보다 더 많은 유연성을 제공하며 생성된 LV는 물리 장치를 다시 파티셔닝하거나 다시 포맷하지 않고도 확장하거나 줄일 수 있습니다. 또한 씬 프로비저닝된 논리 볼륨 생성, 원래 볼륨, RAID 볼륨, 캐시 볼륨, 스트라이핑된 논리 볼륨 생성과 같은 여러 고급 작업을 수행할 수 있습니다.

차례

RED HAT 문서에 관한 피드백 제공 5

1장. 논리 볼륨 관리 개요 6
1.1. LVM 아키텍처 6
1.2. LVM의 이점 7

2장. RHEL 시스템 역할을 사용하여 로컬 스토리지 관리 9
2.1. 스토리지 RHEL 시스템 역할 소개 9
2.2. 스토리지 RHEL 시스템 역할을 사용하여 블록 장치에서 XFS 파일 시스템 생성 10
2.3. 스토리지 RHEL 시스템 역할을 사용하여 파일 시스템을 영구적으로 마운트 11
2.4. 스토리지 RHEL 시스템 역할을 사용하여 논리 볼륨 관리 12
2.5. 스토리지 RHEL 시스템 역할을 사용하여 온라인 블록 삭제 활성화 13
2.6. 스토리지 RHEL 시스템 역할을 사용하여 EXT4 파일 시스템 생성 및 마운트 14
2.7. 스토리지 RHEL 시스템 역할을 사용하여 EXT3 파일 시스템 생성 및 마운트 14
2.8. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 기존 파일 시스템 크기 조정 16
2.9. 스토리지 RHEL 시스템 역할을 사용하여 스왑 볼륨 생성 17
2.10. 스토리지 RHEL 시스템 역할을 사용하여 RAID 볼륨 구성 18
2.11. 스토리지 RHEL 시스템 역할을 사용하여 RAID로 LVM 풀 구성 19
2.12. 스토리지 RHEL 시스템 역할을 사용하여 RAID LVM 볼륨의 스트라이프 크기 구성 20
2.13. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 VDO 볼륨 압축 및 중복 제거 21
2.14. 스토리지 RHEL 시스템 역할을 사용하여 LUKS2 암호화된 볼륨 생성 22
2.15. 스토리지 RHEL 시스템 역할을 사용하여 풀 볼륨 크기를 백분율로 표현 24

3장. LVM 물리 볼륨 관리 26
3.1. 물리 볼륨 개요 26
3.2. 디스크의 여러 파티션 27
3.3. LVM 물리 볼륨 생성 28
3.4. LVM 물리 볼륨 제거 30
3.5. 추가 리소스 30

4장. LVM 볼륨 그룹 관리 31
4.1. LVM 볼륨 그룹 생성 31
4.2. LVM 볼륨 그룹 결합 33
4.3. 볼륨 그룹에서 물리 볼륨 제거 33
4.4. LVM 볼륨 그룹 분할 35
4.5. 볼륨 그룹을 다른 시스템으로 이동 36
4.6. LVM 볼륨 그룹 제거 38

5장. LVM 논리 볼륨 관리 40
5.1. 논리 볼륨 개요 40
5.2. LVM 논리 볼륨 생성 41
5.3. RAID0 스트라이핑 논리 볼륨 생성 43
5.4. LVM 논리 볼륨 이름 44
5.5. 논리 볼륨에서 디스크 제거 45
5.6. LVM 논리 볼륨 제거 46
5.7. RHEL 시스템 역할을 사용하여 LVM 논리 볼륨 관리 47
5.8. LVM 볼륨 그룹 제거 49

6장. 논리 볼륨의 크기 수정 51
6.1. 논리 볼륨 및 파일 시스템 확장 51
6.2. 논리 볼륨 및 파일 시스템 감소 53
6.3. 스트라이핑된 논리 볼륨 확장 54

7장. LVM 보고서 사용자 정의 **57**
7.1. LVM 디스플레이의 제어 형식 57
7.2. LVM 보고서 디스플레이의 단위 지정 58
7.3. LVM 구성 파일 사용자 정의 60
7.4. LVM 선택 기준 정의 61

8장. 공유 스토리지에서 LVM 구성 **64**
8.1. VM 디스크에 대한 LVM 구성 64
8.2. 한 시스템에서 SAN 디스크를 사용하도록 LVM 구성 64
8.3. 장애 조치에 SAN 디스크를 사용하도록 LVM 구성 65
8.4. 여러 시스템에서 SAN 디스크를 공유하도록 LVM 구성 66
8.5. 스토리지 RHEL 시스템 역할을 사용하여 공유 LVM 장치 생성 67

9장. RAID 논리 볼륨 구성 **69**
9.1. RAID 논리 볼륨 69
9.2. RAID 수준 및 선형 지원 70
9.3. LVM RAID 세그먼트 유형 72
9.4. RAID 논리 볼륨 생성 73
9.5. RAID0 스트라이핑 논리 볼륨 생성 74
9.6. 스토리지 RHEL 시스템 역할을 사용하여 RAID LVM 볼륨의 스트라이프 크기 구성 76
9.7. RAID0을 생성하는 매개변수 77
9.8. 소프트 데이터 손상 78
9.9. DM 무결성으로 RAID LV 생성 79
9.10. 최소 및 최대 I/O 속도 옵션 81
9.11. 선형 장치를 RAID 논리 볼륨으로 변환 82
9.12. LVM RAID1 논리 볼륨을 LVM 선형 논리 볼륨으로 변환 83
9.13. 미러링된 LVM 장치를 RAID1 논리 볼륨으로 변환 84
9.14. RAID 논리 볼륨의 크기를 조정하는 명령 85
9.15. 기존 RAID1 장치의 이미지 수 변경 86
9.16. RAID 이미지를 별도의 논리 볼륨으로 분할 88
9.17. RAID 이미지 분할 및 병합 89
9.18. RAID 오류 정책 설정 91
9.19. 논리 볼륨에서 RAID 장치 교체 94
9.20. RAID 논리 볼륨에서 데이터 일관성 확인 99
9.21. RAID 논리 볼륨을 다른 RAID 수준으로 변환 101
9.22. RAID1 논리 볼륨에서 I/O 작업 103
9.23. RAID 볼륨 교체 104
9.24. RAID 논리 볼륨에서 영역 크기 변경 106

10장. 논리 볼륨의 스냅샷 **110**
10.1. 스냅샷 볼륨 개요 110
10.2. 원래 볼륨의 스냅샷 생성 111
10.3. 스냅샷을 원래 볼륨에 병합 114
10.4. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 생성 114
10.5. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 마운트 해제 116
10.6. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 확장 118
10.7. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 복원 121
10.8. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 제거 123

11장. 씬 프로비저닝 볼륨 생성 및 관리(THIN VOLUMES) **126**
11.1. 씬 프로비저닝 개요 126
11.2. 씬 프로비저닝된 논리 볼륨 생성 127
11.3. 청크 크기 개요 132
11.4. 씬 프로비저닝된 스냅샷 볼륨 133

11.5. 씬 프로비저닝된 스냅샷 볼륨 생성 134

12장. 캐싱을 활성화하여 논리 볼륨 성능 개선 138
12.1. LVM의 캐싱 방법 138
12.2. LVM 캐싱 구성 요소 138
12.3. 논리 볼륨에 대한 DM-CACHE 캐싱 활성화 140
12.4. 논리 볼륨의 CACHEPOOL을 사용하여 DM-CACHE 캐싱 활성화 142
12.5. 논리 볼륨에 대한 DM-WRITECACHE 캐싱 활성화 144
12.6. 논리 볼륨의 캐싱 비활성화 147

13장. 논리 볼륨 활성화 149
13.1. 논리 볼륨 및 볼륨 그룹의 자동 활성화 제어 149
13.2. 논리 볼륨 활성화 제어 151
13.3. 공유 논리 볼륨 활성화 152
13.4. 누락된 장치가 있는 논리 볼륨 활성화 153

14장. LVM 장치 가시성 및 사용 제한 154
14.1. LVM 필터링을 위한 영구 식별자 154
14.2. LVM 장치 필터 154

15장. LVM 할당 제어 158
15.1. 지정된 장치의 확장 영역 할당 158
15.2. LVM 할당 정책 161
15.3. 물리 볼륨에서 할당 방지 163

16장. 태그가 있는 LVM 오브젝트 그룹화 164
16.1. LVM 오브젝트 태그 164
16.2. LVM 태그 나열 164
16.3. LVM 오브젝트에 태그 추가 165
16.4. LVM 오브젝트에서 태그 제거 166
16.5. LVM 호스트 태그 정의 166
16.6. 태그를 사용하여 논리 볼륨 활성화 제어 167

17장. LVM 문제 해결 168
17.1. LVM에서 진단 데이터 수집 168
17.2. 실패한 LVM 장치에 대한 정보 표시 169
17.3. 볼륨 그룹에서 손실된 LVM 물리 볼륨 제거 171
17.4. 누락된 LVM 물리 볼륨의 메타데이터 찾기 172
17.5. LVM 물리 볼륨에서 메타데이터 복원 173
17.6. LVM 출력에서 오류 반올림 175
17.7. LVM 볼륨을 만들 때 반올림 오류 방지 175
17.8. LVM 메타데이터 및 디스크의 위치 177
17.9. 디스크에서 VG 메타데이터 추출 178
17.10. 추출된 메타데이터를 파일에 저장 180
17.11. PVCREATE 및 CRYOSTATCFGRESTORE 명령을 사용하여 손상된 LVM 헤더 및 메타데이터로 디스크 복구 181
17.12. PVCK 명령을 사용하여 손상된 LVM 헤더 및 메타데이터로 디스크 복구 183
17.13. LVM RAID 문제 해결 184
17.14. 다중 경로 LVM 장치에 대한 중복 물리 볼륨 경고 문제 해결 189

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견에 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. **요약** 필드에 설명 제목을 입력합니다.
4. **설명** 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. 대화 상자 하단에서 **생성** 을 클릭합니다.

1장. 논리 볼륨 관리 개요

LVM(Logical Volume Management)은 물리 스토리지보다 추상화 계층을 생성하므로 논리 스토리지 볼륨을 생성할 수 있습니다. 이를 통해 물리적 스토리지를 직접 사용하는 것보다 여러 가지 면에서 유연성이 향상됩니다.

또한 하드웨어 스토리지 구성은 소프트웨어에서 숨겨져 있으므로 애플리케이션을 중지하거나 파일 시스템을 마운트 해제하지 않고도 크기를 조정하고 이동할 수 있습니다. 이는 운영 비용을 줄일 수 있습니다.

1.1. LVM 아키텍처

다음은 LVM의 구성 요소입니다.

물리 볼륨

PV(물리 볼륨)는 LVM 사용을 위해 지정된 파티션 또는 전체 디스크입니다. 자세한 내용은 LVM 물리 볼륨 관리를 참조하십시오.

볼륨 그룹

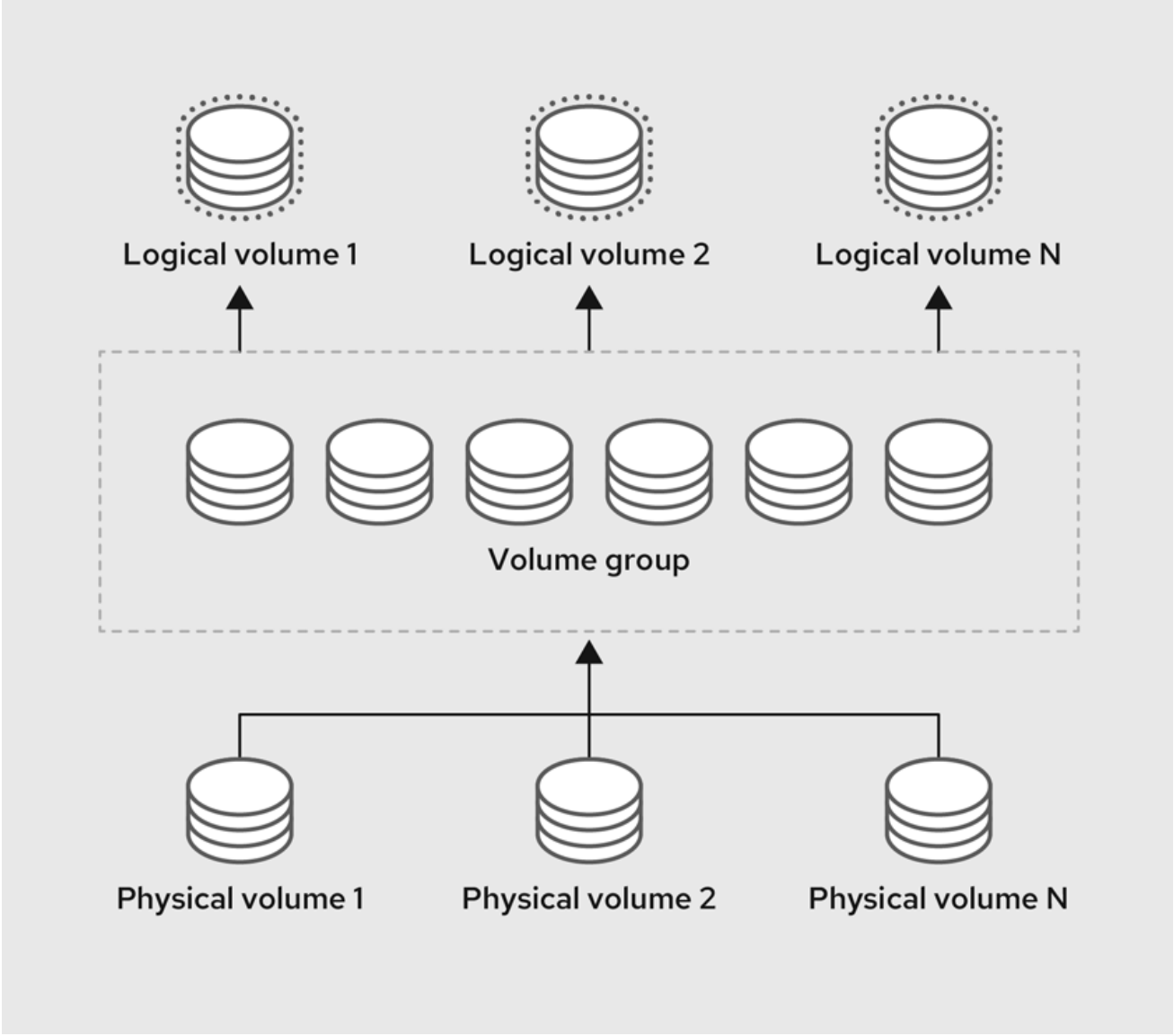
볼륨 그룹(VG)은 물리 볼륨(PV) 컬렉션으로, 논리 볼륨을 할당할 수 있는 디스크 공간 풀을 생성합니다. 자세한 내용은 LVM 볼륨 그룹 관리를 참조하십시오.

논리 볼륨

논리 볼륨은 마운트 가능한 스토리지 장치를 나타냅니다. 자세한 내용은 LVM 논리 볼륨 관리를 참조하십시오.

다음 다이어그램에서는 LVM의 구성 요소를 보여줍니다.

그림 1.1. LVM 논리 볼륨 구성 요소



1.2. LVM의 이점

논리 볼륨은 물리적 스토리지를 직접 사용하는 것보다 다음과 같은 이점을 제공합니다.

유연한 용량

논리 볼륨을 사용하는 경우 장치와 파티션을 단일 논리 볼륨으로 집계할 수 있습니다. 이 기능을 사용하면 파일 시스템을 하나의 큰 장치처럼 여러 장치에 확장할 수 있습니다.

편리한 장치 이름 지정

논리 스토리지 볼륨은 사용자 정의 및 사용자 지정 이름으로 관리할 수 있습니다.

크기 조정 가능한 스토리지 볼륨

기본 장치를 다시 포맷하고 다시 분할하지 않고 간단한 소프트웨어 명령으로 논리 볼륨을 확장하거나 크기를 줄일 수 있습니다. 자세한 내용은 논리 볼륨의 크기 수정을 참조하십시오.

온라인 데이터 재배치

최신 스토리지 하위 시스템을 배포하려면 **pvmove** 명령을 사용하여 시스템을 활성화하는 동안 데이터를 이동할 수 있습니다. 디스크가 사용되는 동안 디스크에 데이터를 다시 정렬할 수 있습니다. 예를 들어 핫 스왑 가능 디스크를 제거하기 전에 비워 둘 수 있습니다.

데이터를 마이그레이션하는 방법에 대한 자세한 내용은 **pvmove** 도움말 페이지 및 볼륨 그룹에서 물리 볼륨 제거를 참조하십시오.

제거된 볼륨

두 개 이상의 장치에서 데이터를 스트라이핑하는 논리 볼륨을 생성할 수 있습니다. 이로 인해 처리량이 크게 증가할 수 있습니다. 자세한 내용은 스트라이핑된 논리 볼륨 확장을 참조하십시오.

RAID 볼륨

논리 볼륨은 데이터에 대한 RAID를 구성하는 편리한 방법을 제공합니다. 이렇게 하면 장치 오류에 대한 보호 기능과 성능이 향상됩니다. 자세한 내용은 RAID 논리 볼륨 구성 을 참조하십시오.

볼륨 스냅샷

일관된 백업을 위해 논리 볼륨의 시점 복사본인 스냅샷을 만들거나 실제 데이터에 영향을 주지 않고 변경 효과를 테스트할 수 있습니다. 자세한 내용은 논리 볼륨의 스냅샷 을 참조하십시오.

씬 볼륨

논리 볼륨은 씬 프로비저닝할 수 있습니다. 이를 통해 사용 가능한 물리 공간보다 큰 논리 볼륨을 만들 수 있습니다. 자세한 내용은 씬 프로비저닝된 볼륨(볼륨) 생성 및 관리를 참조하십시오.

볼륨 캐시

캐시 논리 볼륨은 SSD 드라이브와 같은 빠른 블록 장치를 사용하여 더 크고 느린 블록 장치의 성능을 향상시킵니다. 자세한 내용은 논리 볼륨 성능을 개선하기 위해 캐싱 활성화를 참조하십시오.

추가 리소스

- LVM 보고서 사용자 정의

2장. RHEL 시스템 역할을 사용하여 로컬 스토리지 관리

Ansible을 사용하여 LVM 및 로컬 파일 시스템(FS)을 관리하려면 RHEL 8에서 사용할 수 있는 RHEL 시스템 역할 중 하나인 **스토리지** 역할을 사용할 수 있습니다.

스토리지 역할을 사용하면 RHEL 7.7부터 여러 시스템의 디스크 및 논리 볼륨 및 모든 버전의 RHEL에서 파일 시스템을 자동으로 관리할 수 있습니다.

RHEL 시스템 역할 및 적용 방법에 대한 자세한 내용은 RHEL 시스템 역할 소개를 참조하십시오.

2.1. 스토리지 RHEL 시스템 역할 소개

스토리지 역할은 다음을 관리할 수 있습니다.

- 분할되지 않은 디스크의 파일 시스템
- 논리 볼륨 및 파일 시스템을 포함한 LVM 볼륨 그룹 완료
- MD RAID 볼륨 및 파일 시스템

스토리지 역할을 사용하면 다음 작업을 수행할 수 있습니다.

- 파일 시스템 생성
- 파일 시스템 제거
- 파일 시스템 마운트
- 파일 시스템 마운트 해제
- LVM 볼륨 그룹 만들기
- LVM 볼륨 그룹 제거
- 논리 볼륨 생성
- 논리 볼륨 제거
- RAID 볼륨 생성
- RAID 볼륨 제거
- RAID로 LVM 볼륨 그룹 생성
- RAID를 사용하여 LVM 볼륨 그룹 제거
- 암호화된 LVM 볼륨 그룹 생성
- RAID를 사용하여 LVM 논리 볼륨 생성

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.2. 스토리지 RHEL 시스템 역할을 사용하여 블록 장치에서 XFS 파일 시스템 생성

예제 Ansible 플레이북은 기본 매개 변수를 사용하여 블록 장치에서 XFS 파일 시스템을 생성하는 데 **storage** 역할을 적용합니다.

참고

스토리지 역할은 분할되지 않은 전체 디스크 또는 논리 볼륨(LV)에서만 파일 시스템을 생성할 수 있습니다. 파티션에 파일 시스템을 만들 수 없습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
```

   - 볼륨 이름(예의 ***barefs***)은 현재 임의로 사용할 수 있습니다. **스토리지** 역할은 **disks:** 속성 아래에 나열된 디스크 장치에서 볼륨을 식별합니다.
   - XFS는 RHEL 8의 기본 파일 시스템이므로 **fs_type: xfs** 행을 생략할 수 있습니다.
   - LV에 파일 시스템을 생성하려면 enclosing 볼륨 그룹을 포함하여 **disks:** 속성 아래에 LVM 설정을 제공합니다. 자세한 내용은 Storage RHEL 시스템 역할을 사용하여 논리 볼륨 관리를 참조하십시오.
     LV 장치의 경로를 제공하지 마십시오.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.3. 스토리지 RHEL 시스템 역할을 사용하여 파일 시스템을 영구적으로 마운트

예제 Ansible은 **스토리지** 역할을 XFS 파일 시스템에 즉시 영구적으로 마운트하는 데 적용합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - 이 플레이북은 파일 시스템을 **/etc/fstab** 파일에 추가하고 즉시 파일 시스템을 마운트합니다.
   - **/dev/sdb** 장치 또는 마운트 지점 디렉터리의 파일 시스템이 존재하지 않는 경우 플레이북에서 해당 시스템을 생성합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.4. 스토리지 RHEL 시스템 역할을 사용하여 논리 볼륨 관리

예제 Ansible 플레이북은 **storage** 역할을 적용하여 볼륨 그룹에 LVM 논리 볼륨을 생성합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt/dat
```

   - **myvg** 볼륨 그룹은 **/dev/sda**,**/dev/sdb**, **/dev/sdc** 디스크로 구성됩니다.
   - **myvg** 볼륨 그룹이 이미 있는 경우 Playbook은 볼륨 그룹에 논리 볼륨을 추가합니다.
   - **myvg** 볼륨 그룹이 없으면 플레이북에서 해당 그룹을 생성합니다.
   - 플레이북은 **mylv** 논리 볼륨에 Ext4 파일 시스템을 생성하고 **/mnt** 에 파일 시스템을 영구적으로 마운트합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.5. 스토리지 RHEL 시스템 역할을 사용하여 온라인 블록 삭제 활성화

예제 Ansible 플레이북은 **스토리지** 역할을 적용하여 온라인 블록 삭제가 활성화된 XFS 파일 시스템을 마운트합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
        mount_options: discard
```

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.6. 스토리지 RHEL 시스템 역할을 사용하여 EXT4 파일 시스템 생성 및 마운트

예제 Ansible 플레이북은 Ext4 파일 시스템을 생성하고 마운트하는 **스토리지** 역할을 적용합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext4
        fs_label: label-name
        mount_point: /mnt/data
```

   - Playbook은 **/dev/sdb** 디스크에 파일 시스템을 생성합니다.
   - 플레이북은 **/mnt/data** 디렉터리에 파일 시스템을 영구적으로 마운트합니다.
   - 파일 시스템의 레이블은 **label-name** 입니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.7. 스토리지 RHEL 시스템 역할을 사용하여 EXT3 파일 시스템 생성 및 마운트

예제 Ansible 플레이북은 Ext3 파일 시스템을 생성하고 마운트하는 **스토리지** 역할을 적용합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- hosts: all
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: ext3
        fs_label: label-name
        mount_point: /mnt/data
        mount_user: somebody
        mount_group: somegroup
        mount_mode: 0755
```

   - Playbook은 **/dev/sdb** 디스크에 파일 시스템을 생성합니다.
   - 플레이북은 **/mnt/data** 디렉터리에 파일 시스템을 영구적으로 마운트합니다.
   - 파일 시스템의 레이블은 **label-name** 입니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.8. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 기존 파일 시스템 크기 조정

예제 Ansible 플레이북은 **스토리지** RHEL 시스템 역할을 적용하여 파일 시스템으로 LVM 논리 볼륨의 크기를 조정합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Create LVM pool over three disks
  hosts: managed-node-01.example.com
  tasks:
    - name: Resize LVM logical volume with file system
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_pools:
          - name: myvg
            disks:
              - /dev/sda
              - /dev/sdb
              - /dev/sdc
            volumes:
              - name: mylv1
                size: 10 GiB
                fs_type: ext4
                mount_point: /opt/mount1
              - name: mylv2
                size: 50 GiB
                fs_type: ext4
                mount_point: /opt/mount2
```

   이 Playbook은 다음과 같은 기존 파일 시스템의 크기를 조정합니다.

   - **/opt/mount1** 에 마운트된 **mylv1** 볼륨의 Ext4 파일 시스템은 10GiB로 크기를 조정합니다.
   - **/opt/mount2** 에 마운트된 **mylv2** 볼륨의 Ext4 파일 시스템은 50GiB로 크기를 조정합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.9. 스토리지 RHEL 시스템 역할을 사용하여 스왑 볼륨 생성

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 기본 매개 변수를 사용하여 블록 장치에서 스왑 볼륨을 생성하거나 스왑 볼륨이 없는 경우 스왑 볼륨을 수정하는 데 **스토리지** 역할을 적용합니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Create a disk device with swap
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: swap_fs
        type: disk
        disks:
          - /dev/sdb
        size: 15 GiB
        fs_type: swap
```

   볼륨 이름(예의***swap_fs***)은 현재 임의적입니다. **스토리지** 역할은 **disks:** 속성 아래에 나열된 디스크 장치에서 볼륨을 식별합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

   이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.10. 스토리지 RHEL 시스템 역할을 사용하여 RAID 볼륨 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform 및 Ansible-Core를 사용하여 RHEL에서 RAID 볼륨을 구성할 수 있습니다. 매개변수를 사용하여 Ansible 플레이북을 생성하여 요구 사항에 맞게 RAID 볼륨을 구성합니다.

주의

장치 이름은 예를 들어 시스템에 새 디스크를 추가하는 경우와 같이 특정 상황에서 변경될 수 있습니다. 따라서 데이터 손실을 방지하려면 플레이북에서 특정 디스크 이름을 사용하지 마십시오.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Configure the storage
  hosts: managed-node-01.example.com
  tasks:
    - name: Create a RAID on sdd, sde, sdf, and sdg
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_safe_mode: false
        storage_volumes:
          - name: data
            type: raid
            disks: [sdd, sde, sdf, sdg]
            raid_level: raid0
            raid_chunk_size: 32 KiB
            mount_point: /mnt/data
            state: present
```

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리
- RAID 관리

2.11. 스토리지 RHEL 시스템 역할을 사용하여 RAID로 LVM 풀 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform을 사용하여 RHEL에서 RAID로 LVM 풀을 구성할 수 있습니다. 사용 가능한 매개 변수로 Ansible 플레이북을 설정하여 RAID로 LVM 풀을 구성할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Configure LVM pool with RAID
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
```

RAID를 사용하여 LVM 풀을 생성하려면 **raid_level** 매개변수를 사용하여 RAID 유형을 지정해야 합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리
- RAID 관리

2.12. 스토리지 RHEL 시스템 역할을 사용하여 RAID LVM 볼륨의 스트라이프 크기 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform을 사용하여 RHEL에서 RAID LVM 볼륨의 스트라이프 크기를 구성할 수 있습니다. 사용 가능한 매개 변수로 Ansible 플레이북을 설정하여 RAID로 LVM 풀을 구성할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
```

```
        raid_level: raid1
        raid_stripe_size: "256 KiB"
        state: present
```

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리
- RAID 관리

2.13. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 VDO 볼륨 압축 및 중복 제거

예제 Ansible 플레이북은 VDO(Virtual Data Optimizer)를 사용하여 **스토리지** RHEL 시스템 역할을 적용하여 LVM(Logical Volumes)의 압축 및 중복 제거를 활성화합니다.

참고

스토리지 시스템 역할은 LVM VDO를 사용하므로 풀당 하나의 볼륨만 압축 및 중복 제거를 사용할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
```

```
    - /dev/sdb
  volumes:
    - name: mylv1
      compression: true
      deduplication: true
      vdo_pool_size: 10 GiB
      size: 30 GiB
      mount_point: /mnt/app/shared
```

이 예제에서 **압축** 및 **중복 제거** 풀은 true로 설정되어 VDO가 사용되도록 지정합니다. 다음에서는 이러한 매개변수의 사용법을 설명합니다.

- **중복 제거** 는 스토리지 볼륨에 저장된 중복 데이터를 중복 제거하는 데 사용됩니다.
- 압축은 스토리지 볼륨에 저장된 데이터를 압축하여 더 많은 스토리지 용량을 만드는 데 사용됩니다.
- vdo_pool_size는 볼륨이 장치에서 사용하는 실제 크기를 지정합니다. VDO 볼륨의 가상 크기는 **size** 매개변수로 설정됩니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

2.14. 스토리지 RHEL 시스템 역할을 사용하여 LUKS2 암호화된 볼륨 생성

storage 역할을 사용하여 Ansible 플레이북을 실행하여 LUKS로 암호화된 볼륨을 생성하고 구성할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
```

```
- name: Create and configure a volume encrypted with LUKS
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: <password>
```

encryption_key ,**encryption_cipher**, **encryption_key _size , encryption_ luks** 와 같은 다른 암호화 매개변수를 플레이북 파일에 추가할 수도 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

검증

1. 암호화 상태를 확인합니다.

```
# cryptsetup status sdb

/dev/mapper/sdb is active and is in use.
type: LUKS2
cipher: aes-xts-plain64
keysize: 512 bits
key location: keyring
device: /dev/sdb
...
```

2. 생성된 LUKS 암호화된 볼륨을 확인합니다.

```
# cryptsetup luksDump /dev/sdb

Version:        2
Epoch:          6
Metadata area:  16384 [bytes]
Keyslots area:  33521664 [bytes]
UUID:           a4c6be82-7347-4a91-a8ad-9479b72c9426
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          allow-discards
```

```
Data segments:
  0: crypt
        offset: 33554432 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 4096 [bytes]
...
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리
- LUKS를 사용하여 블록 장치 암호화

2.15. 스토리지 RHEL 시스템 역할을 사용하여 풀 볼륨 크기를 백분율로 표현

예제 Ansible 플레이북은 **스토리지** 시스템 역할을 적용하여 논리 관리자 볼륨(LVM) 볼륨 크기를 풀의 총 크기의 백분율로 표시할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - /dev/sdb
        volumes:
          - name: data
            size: 60%
            mount_point: /opt/mount/data
          - name: web
            size: 30%
            mount_point: /opt/mount/web
          - name: cache
            size: 10%
            mount_point: /opt/cache/mount
```

이 예에서는 LVM 볼륨의 크기를 풀 크기의 백분율로 지정합니다. 예를 들면 다음과 같습니다. **60%**. 또는 사용자가 읽을 수 있는 파일 시스템의 크기(예: **10g** 또는 **50GiB**)에서 풀 크기의 백분율로 LVM 볼륨의 크기를 지정할 수도 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file
- **/usr/share/doc/rhel-system-roles/storage/** 디렉터리

3장. LVM 물리 볼륨 관리

PV(물리 볼륨)는 LVM 사용을 위해 지정된 파티션 또는 전체 디스크입니다. LVM 논리 볼륨에 장치를 사용하려면 장치를 물리 볼륨으로 초기화해야 합니다.

물리 볼륨에 전체 디스크 장치를 사용하는 경우 디스크에 파티션 테이블이 없어야 합니다. ECDHE 디스크 파티션의 경우 파티션 ID는ECDHE 또는 **cfdisk** 명령을 사용하여 0x8e로 설정되어야 합니다. **물리 볼륨에 전체 디스크 장치를 사용하는 경우 디스크에 파티션 테이블이 없어야 합니다. 기존 파티션 테이블을 삭제해야 합니다. 그러면 해당 디스크의 모든 데이터를 효과적으로 삭제합니다. delete fs -a <PhysicalVolume>' 명령을 root로 사용하여 기존 파티션 테이블을 제거할 수 있습니다.**

3.1. 물리 볼륨 개요

블록 장치를 물리 볼륨으로 초기화하면 장치 시작 가까이에 라벨이 배치됩니다. 다음은 LVM 레이블을 설명합니다.

- **LVM 레이블은 물리적 장치의 올바른 식별 및 장치 순서를 제공합니다. 레이블이 지정되지 않은 LVM 장치는 부팅 중에 시스템에서 검색한 순서에 따라 재부팅 시 이름을 변경할 수 있습니다. LVM 레이블은 재부팅과 클러스터 전체에서 계속 유지됩니다.**

- **LVM 레이블은 장치를 LVM 물리 볼륨으로 식별합니다. 여기에는 물리 볼륨의 UUID인 임의의 고유 식별자가 포함되어 있습니다. 또한 블록 장치의 크기를 바이트 단위로 저장하고 LVM 메타데이터가 장치에 저장될 위치를 기록합니다.**

- **기본적으로 LVM 레이블은 두 번째 512바이트 섹터에 배치됩니다. 물리 볼륨을 생성할 때 처음 4개 섹터 중 하나에 라벨을 배치하여 이 기본 설정을 덮어쓸 수 있습니다. 이를 통해 LVM 볼륨은 필요한 경우 이러한 섹터의 다른 사용자와 연결할 수 있습니다.**

다음은 LVM 메타데이터를 설명합니다.

- **LVM 메타데이터에는 시스템의 LVM 볼륨 그룹의 구성 세부 정보가 포함되어 있습니다. 기본적으로 동일한 메타데이터 복사본은 볼륨 그룹 내의 모든 물리 볼륨 영역에서 유지됩니다. LVM 메타데이터는 작으며 ASCII로 저장됩니다.**

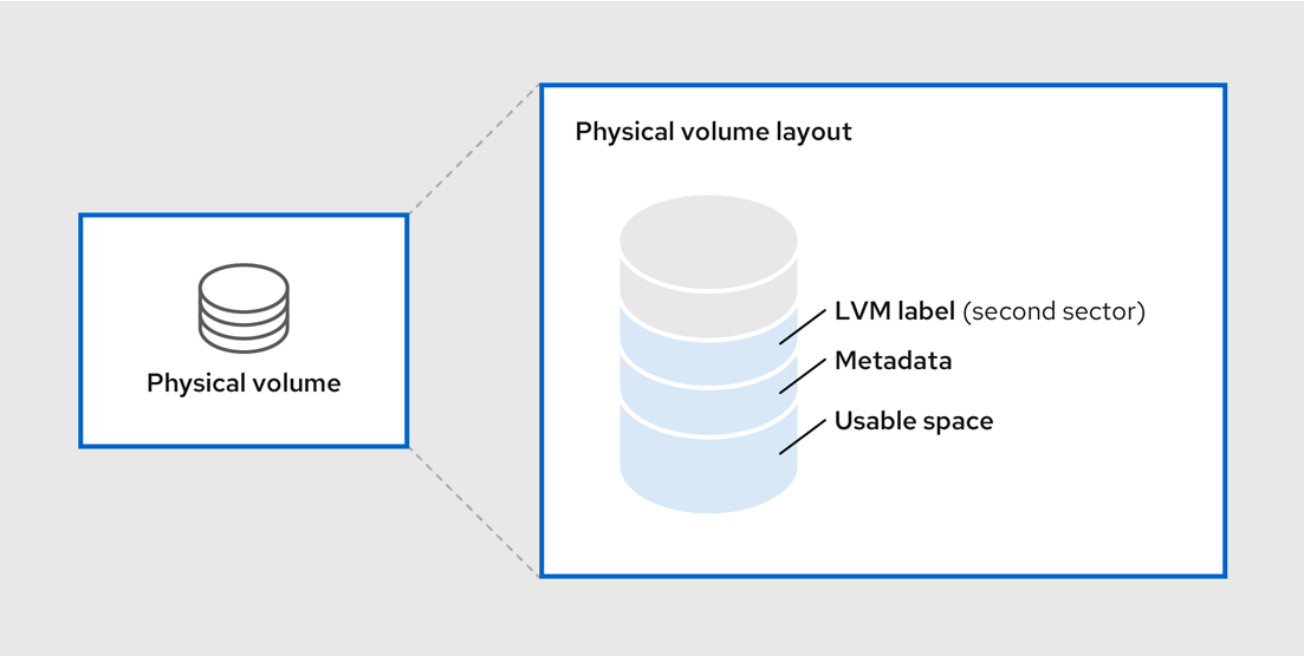
- **현재 LVM을 사용하면 각 물리 볼륨에 0, 1 또는 2개의 동일한 메타데이터 복사본을 저장할 수 있습니다. 기본값은 1 copy입니다. 물리 볼륨에서 메타데이터 복사본 수를 구성하면 나중에 해당 수를 변경할 수 없습니다. 첫 번째 복사본은 장치 시작 부분에 저장되며 레이블 바로 뒤에 저장됩니다. 두 번째 사본이 있는 경우 장치의 끝에 배치됩니다. 의도한 것과 다른 디스크에 작성하여 디스크 시작 부분에 있는 영역을 실수로 덮어 쓰면 장치 끝에 있는 메타데이터의 두 번째 복사본을 통해 메타데이터를 복구할 수 있습니다.**

다음 다이어그램에서는 LVM 물리 볼륨의 레이아웃을 보여줍니다. LVM 레이블은 두 번째 섹터에 있고, 메타데이터 영역과 그 뒤에 장치에서 사용 가능한 공간이 옵니다.

참고

Linux 커널 및 이 문서 전반에 걸쳐 섹터는 512바이트 크기의 것으로 간주됩니다.

그림 3.1. 물리 볼륨 레이아웃



추가 리소스

- 디스크의 여러 파티션

3.2. 디스크의 여러 파티션

LVM을 사용하여 디스크 파티션에서 PV(물리 볼륨)를 생성할 수 있습니다.

다음과 같은 이유로 전체 디스크를 LVM 물리 볼륨으로 레이블하는 단일 파티션을 생성하는 것이 좋습니다.

관리 편의성

각 실제 디스크가 한 번만 표시되면 시스템의 하드웨어를 추적하는 것이 더 쉽습니다. 특히 디스크가 실패하는 경우 그러합니다.

성능 제거

LVM에서는 두 개의 물리 볼륨이 동일한 물리 디스크에 있다고 표시할 수 없습니다. 두 개의 물리 볼륨이 동일한 물리 디스크에 있을 때 스트라이핑된 논리 볼륨을 생성하면 스트라이프가 동일한 디스크의 다른 파티션에 있을 수 있습니다. 이로 인해 성능이 향상되지 않고 성능이 저하됩니다.

RAID 중복

LVM에서는 두 개의 물리 볼륨이 동일한 장치에 있는지 확인할 수 없습니다. 두 개의 물리 볼륨이 동일한 장치에 있을 때 RAID 논리 볼륨을 생성하면 성능과 내결함성이 손실될 수 있습니다.

권장되지는 않지만 디스크를 별도의 LVM 물리 볼륨으로 분할해야 하는 경우 특정 상황이 발생할 수 있습니다. 예를 들어 디스크가 적은 시스템에서 기존 시스템을 LVM 볼륨으로 마이그레이션할 때 파티션을 중심으로 데이터를 이동해야 할 수 있습니다. 또한 매우 큰 디스크가 있고 관리 목적으로 두 개 이상의 볼륨 그룹을 사용하려면 디스크를 분할해야 합니다. 파티션이 두 개 이상 있고 해당 파티션이 모두 동일한 볼륨 그룹에 있는 디스크가 있는 경우 볼륨을 생성할 때 논리 볼륨에 포함할 파티션을 지정해야 합니다.

LVM에서는 분할되지 않은 디스크 사용을 물리 볼륨으로 지원하지만 파티션 없이 PV를 생성하는 것이 혼합 운영 체제 환경에서 문제가 발생할 수 있으므로 단일 디스크 파티션을 생성하는 것이 좋습니다. 다른 운영 체제는 장치를 사용 가능한 것으로 해석하고 드라이브 시작 시 PV 레이블을 덮어쓸 수 있습니다.

3.3. LVM 물리 볼륨 생성

이 절차에서는 LVM 물리 볼륨(PV)을 생성하고 레이블을 지정하는 방법을 설명합니다.

이 절차에서는 */dev/vdb1*,*/dev/vdb2*, */dev/vdb3* 을 시스템에서 사용 가능한 스토리지 장치로 교체합니다.

사전 요구 사항

- lvm2 패키지가 설치되어 있습니다.

절차

1. pvcreate 명령에 공백으로 구분된 장치 이름을 인수로 사용하여 여러 물리 볼륨을 생성합니다.

```
# pvcreate /dev/vdb1 /dev/vdb2 /dev/vdb3
  Physical volume "/dev/vdb1" successfully created.
  Physical volume "/dev/vdb2" successfully created.
  Physical volume "/dev/vdb3" successfully created.
```

그러면 */dev/vdb1*,*/dev/vdb2*, */dev/vdb3* 에 레이블이 배치되어 **LVM**에 속하는 물리 볼륨으로 표시됩니다.

2. 요구 사항에 따라 다음 명령 중 하나를 사용하여 생성된 물리 볼륨을 확인합니다.

a. 각 물리 볼륨에 대한 자세한 다중 줄 출력을 제공하는 **pvdisplay** 명령. 크기, 확장 영역, 볼륨 그룹 및 기타 옵션과 같은 물리적 속성을 고정된 형식으로 표시합니다.

```
# pvdisplay
--- NEW Physical volume ---
  PV Name               /dev/vdb1
  VG Name
  PV Size               1.00 GiB
[..]
--- NEW Physical volume ---
  PV Name               /dev/vdb2
  VG Name
  PV Size               1.00 GiB
[..]
--- NEW Physical volume ---
  PV Name               /dev/vdb3
  VG Name
  PV Size               1.00 GiB
[..]
```

b. **pvs** 명령은 물리적 볼륨 정보를 구성 가능한 형식으로 제공하여 물리 볼륨당 한 줄을 표시합니다.

```
# pvs
  PV         VG  Fmt    Attr   PSize     PFree
/dev/vdb1        lvm2          1020.00m   0
/dev/vdb2        lvm2          1020.00m   0
/dev/vdb3        lvm2          1020.00m   0
```

c. **pvscan** 명령은 물리 볼륨에 대해 시스템에서 지원되는 모든 **LVM** 블록 장치를 검사합니다. 이 명령으로 특정 물리 볼륨을 검사하지 않도록 **lvm.conf** 파일에서 필터를 정의할 수 있습니다.

```
# pvscan
  PV  /dev/vdb1                lvm2 [1.00 GiB]
  PV  /dev/vdb2                lvm2 [1.00 GiB]
  PV  /dev/vdb3                lvm2 [1.00 GiB]
```

추가 리소스

- pvcreate(8), pvdisplay(8), pvs(8), pvscan(8), lvm(8) 매뉴얼 페이지

3.4. LVM 물리 볼륨 제거

LVM에서 장치를 더 이상 사용할 필요가 없는 경우 pvremove 명령을 사용하여 LVM 레이블을 제거할 수 있습니다. pvremove 명령을 실행하면 빈 물리 볼륨에서 LVM 메타데이터가 0입니다.

절차

1. 물리 볼륨 제거:

```
# pvremove /dev/vdb3
Labels on physical volume "/dev/vdb3" successfully wiped.
```

2. 기존 물리 볼륨을 보고 필요한 볼륨이 제거되었는지 확인합니다.

```
# pvs
  PV         VG   Fmt  Attr PSize    PFree
/dev/vdb1        lvm2       1020.00m  0
/dev/vdb2        lvm2       1020.00m  0
```

제거하려는 물리 볼륨이 현재 볼륨 그룹의 일부이면 reduce 명령을 사용하여 볼륨 그룹에서 제거해야 합니다. 자세한 내용은 볼륨 그룹에서 물리 볼륨 제거를참조하십시오.

추가 리소스

- pvremove(8) man page

3.5. 추가 리소스

- parted가 있는 디스크에 파티션 테이블 만들기
- parted(8) 도움말 페이지.

4장. LVM 볼륨 그룹 관리

볼륨 그룹(VG)은 물리 볼륨(PV) 컬렉션으로, 논리 볼륨(LV)을 할당할 수 있는 디스크 공간 풀을 생성합니다.

볼륨 그룹 내에서 할당에 사용할 수 있는 디스크 공간은 **Extent**라는 고정된 크기의 단위로 나뉩니다. 범위는 할당할 수 있는 가장 작은 공간 단위입니다. 물리 볼륨 내에서 확장 영역을 물리 확장 영역이라고 합니다.

논리 볼륨은 물리 확장 영역과 동일한 크기의 논리 확장 영역에 할당됩니다. 따라서 볼륨 그룹의 모든 논리 볼륨에 대해 확장 영역 크기는 동일합니다. 볼륨 그룹은 논리 확장 영역을 물리 확장 영역에 매핑합니다.

4.1. LVM 볼륨 그룹 생성

/dev/vdb1 및 */dev/vdb2* 물리 볼륨(PV)을 사용하여 **LVM** 볼륨 그룹(VG) **myvg**를 생성할 수 있습니다. 기본적으로 물리 볼륨을 사용하여 볼륨 그룹을 만들 때 디스크 공간은 **4MB Extent**로 나뉩니다. 이 범위 크기는 논리 볼륨을 늘리거나 줄일 수 있는 최소 양입니다. **extent** 크기는 **protect create** 명령의 **-s** 인수를 사용하여 수정할 수 있으며 다수의 **Extent**는 논리 볼륨의 **I/O** 성능에 영향을 미치지 않습니다. **Cryostat create** 명령의 **-p** 및 **-l** 인수를 사용하여 볼륨 그룹이 보유할 수 있는 물리 볼륨 또는 논리 볼륨 수에 제한을 둘 수 있습니다.

사전 요구 사항

- **lvm2** 패키지가 설치되어 있습니다.
- 하나 이상의 물리 볼륨이 생성됩니다. 물리 볼륨 생성에 대한 자세한 내용은 **LVM** 물리 볼륨 생성 을 참조하십시오.

절차

1. 다음 방법을 사용하여 ***myvg*** **VG**를 만듭니다.
   - 옵션을 지정하지 않고 다음을 수행합니다.

```
# vgcreate myvg /dev/vdb1 /dev/vdb2
 Volume group "myvg" successfully created.
```

- **-s** 인수를 사용하여 볼륨 그룹 범위 크기를 지정하면 됩니다.

```
# vgcreate -s 2 /dev/myvg /dev/vdb1 /dev/vdb2
Volume group "myvg" successfully created.
```

- **VG**에서 **-p** 및 **-l** 인수를 사용하여 물리 볼륨 또는 논리 볼륨 수를 제한하면 다음과 같습니다.

```
# vgcreate -l 1 /dev/myvg /dev/vdb1 /dev/vdb2
Volume group "myvg" successfully created.
```

2. 요구 사항에 따라 다음 명령 중 하나를 사용하여 생성된 볼륨 그룹을 확인합니다.

- **Cryo stats** 명령은 볼륨 그룹 정보를 구성 가능한 형식으로 제공하여 볼륨 그룹당 한 행을 표시합니다.

```
# vgs
  VG    #PV #LV #SN  Attr  VSize   VFree
 myvg   2    0   0   wz-n  159.99g 159.99g
```

- **display** 명령은 크기, 확장 영역, 물리 볼륨 수 및 기타 옵션과 같은 볼륨 그룹 속성을 고정된 형식으로 표시합니다. 다음 예에서는 볼륨 그룹 ***myvg*** 에 대한**display** 명령의 출력을 보여줍니다. 기존 볼륨 그룹을 모두 표시하려면 볼륨 그룹을 지정하지 마십시오.

```
# vgdisplay myvg
  --- Volume group ---
  VG Name               myvg
  System ID
  Format                lvm2
  Metadata Areas        4
  Metadata Sequence No  6
  VG Access             read/write
[..]
```

- **FlexVolume scan** 명령은 볼륨 그룹에 대해 시스템에서 지원되는 모든 **LVM** 블록 장치를 검사합니다.

```
# vgscan
  Found volume group "myvg" using metadata type lvm2
```

3. **선택 사항: 사용 가능한 물리 볼륨을 하나 이상 추가하여 볼륨 그룹의 용량을 늘립니다.**

```
# vgextend myvg /dev/vdb3
Physical volume "/dev/vdb3" successfully created.
Volume group "myvg" successfully extended
```

4. **선택 사항: 기존 볼륨 그룹의 이름을 변경합니다.**

```
# vgrename myvg myvg1
Volume group "myvg" successfully renamed to "myvg1"
```

추가 리소스

- **ECDHEcreate(8), extend(8), ,display (8) , ,ECDHEs (8) ,scan (8), lvm(8) 매뉴얼 페이지**

4.2. LVM 볼륨 그룹 결합

두 볼륨 그룹을 단일 볼륨 그룹으로 결합하려면 merge 명령을 사용합니다. 볼륨의 물리 확장 크기가 동일하고 대상 볼륨 그룹의 물리 및 논리 볼륨 합계가 대상 볼륨 그룹 제한에 적합한 경우 비활성 "소스" 볼륨을 활성 또는 비활성 "대상 "대상" 볼륨과 병합할 수 있습니다.

절차

- **비활성 볼륨 그룹 *데이터베이스* 를 활성 또는 비활성 볼륨 그룹 *myvg* 에 병합하여 자세한 런타임 정보를 제공합니다.**

```
# vgmerge -v myvg databases
```

추가 리소스

- **vgmerge(8) man page**

4.3. 볼륨 그룹에서 물리 볼륨 제거

볼륨 그룹(VG)에서 사용되지 않는 PV(물리 볼륨)를 제거하려면 Cryostatreduce 명령을 사용합니다. 이 명령은 빈 물리 볼륨을 하나 이상 제거하여 볼륨 그룹의 용량을 줄입니다. 그러면 이러한 물리 볼륨을 다른 볼륨 그룹에서 사용하거나 시스템에서 제거할 수 있습니다.

절차

1. 물리 볼륨이 아직 사용 중인 경우 동일한 볼륨 그룹에서 데이터를 다른 물리 볼륨으로 마이그레이션합니다.

```
# pvmove /dev/vdb3
  /dev/vdb3: Moved: 2.0%
 ...
  /dev/vdb3: Moved: 79.2%
 ...
  /dev/vdb3: Moved: 100.0%
```

2. 기존 볼륨 그룹의 다른 물리 볼륨에 사용 가능한 확장 영역이 충분하지 않은 경우:

   a. */dev/vdb4* 에서 새 물리 볼륨을 생성합니다.

```
# pvcreate /dev/vdb4
  Physical volume "/dev/vdb4" successfully created
```

   b. 새로 생성된 물리 볼륨을 *myvg* 볼륨 그룹에 추가합니다.

```
# vgextend myvg /dev/vdb4
  Volume group "myvg" successfully extended
```

   c. */dev/vdb3* 에서 */dev/vdb4* 로 데이터를 이동합니다.

```
# pvmove /dev/vdb3 /dev/vdb4
  /dev/vdb3: Moved: 33.33%
  /dev/vdb3: Moved: 100.00%
```

3. 볼륨 그룹에서 물리 볼륨 */dev/vdb3* 을 제거합니다.

```
# vgreduce myvg /dev/vdb3
Removed "/dev/vdb3" from volume group "myvg"
```

검증

- */dev/vdb3* 물리 볼륨이 *myvg* 볼륨 그룹에서 제거되었는지 확인합니다.

```
# pvs
```

```
PV         VG    Fmt   Attr   PSize       PFree      Used
/dev/vdb1 myvg  lvm2  a--    1020.00m    0         1020.00m
/dev/vdb2 myvg  lvm2  a--    1020.00m    0         1020.00m
/dev/vdb3       lvm2  a--    1020.00m  1008.00m    12.00m
```

추가 리소스

- ECDHEreduce(8), pvmove(8), pvs(8) 매뉴얼 페이지

4.4. LVM 볼륨 그룹 분할

물리 볼륨에 사용되지 않는 공간이 충분한 경우 새 디스크를 추가하지 않고 새 볼륨 그룹을 생성할 수 있습니다.

초기 설정에서 볼륨 그룹 *myvg* 는 */dev/vdb1*,*/dev/vdb2*, */dev/vdb3* 으로 구성됩니다. 이 절차를 완료하면 볼륨 그룹 *myvg* 가 */dev/vdb1* 및 */dev/vdb2* 로 구성되며, 두 번째 볼륨 그룹인 *yourvg* 은 */dev/vdb3* 으로 구성됩니다.

사전 요구 사항

- 볼륨 그룹에 충분한 공간이 있습니다. DestinationRule scan 명령을 사용하여 현재 볼륨 그룹에서 사용 가능한 여유 공간을 결정합니다.
- 기존 물리 볼륨의 사용 가능한 용량에 따라 pvmove 명령을 사용하여 사용된 물리 확장 영역을 모두 다른 물리 볼륨으로 이동합니다. 자세한 내용은 볼륨 그룹에서 물리 볼륨 제거를 참조하십시오.

절차

1. 기존 볼륨 그룹 *myvg* 를vg라는 새 볼륨 그룹으로 분할합니다.

```
# vgsplit myvg yourvg /dev/vdb3
  Volume group "yourvg" successfully split from "myvg"
```

참고

기존 볼륨 그룹을 사용하여 논리 볼륨을 생성한 경우 다음 명령을 사용하여 논리 볼륨을 비활성화합니다.

```
# lvchange -a n /dev/myvg/mylv
```

논리 볼륨 생성에 대한 자세한 내용은 LVM 논리 볼륨 관리를 참조하십시오.

2. 두 볼륨 그룹의 속성을 확인합니다.

```
# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  myvg     2   1   0 wz--n- 34.30G 10.80G
  yourvg   1   0   0 wz--n- 17.15G 17.15G
```

검증

- 새로 생성된 볼륨 그룹 yourvg 가 /dev/vdb3 물리 볼륨으로 구성되어 있는지 확인합니다.

```
# pvs
  PV         VG     Fmt  Attr PSize    PFree    Used
  /dev/vdb1 myvg   lvm2 a--  1020.00m 0        1020.00m
  /dev/vdb2 myvg   lvm2 a--  1020.00m 0        1020.00m
  /dev/vdb3 yourvg lvm2 a--  1020.00m 1008.00m   12.00m
```

추가 리소스

- ECDHEsplit(8) ,, , , pvs(8) 매뉴얼 페이지

4.5. 볼륨 그룹을 다른 시스템으로 이동

다음 명령을 사용하여 전체 LVM 볼륨 그룹(VG)을 다른 시스템으로 이동할 수 있습니다.

vgexport

기존 시스템에서 이 명령을 사용하여 비활성 VG가 시스템에서 액세스할 수 없도록 합니다. VG에 액세스할 수 없게 되면 PV(물리 볼륨)를 분리할 수 있습니다.

vgimport

*다른 시스템에서 이 명령을 사용하여 이전 시스템에서 비활성화된 **VG**를 새 시스템에서 액세스할 수 있도록 합니다.*

사전 요구 사항

- *이동 중인 볼륨 그룹의 활성 볼륨의 파일에 액세스 중인 사용자가 없습니다.*

절차

1. ***mylv** 논리 볼륨을 마운트 해제합니다.*

```
# umount /dev/mnt/mylv
```

2. *볼륨 그룹의 모든 논리 볼륨을 비활성화하여 볼륨 그룹의 추가 활동을 방지합니다.*

```
# vgchange -an myvg
vgchange -- volume group "myvg" successfully deactivated
```

3. *볼륨 그룹을 내보내 제거 중인 시스템에서 액세스할 수 없도록 합니다.*

```
# vgexport myvg
vgexport -- volume group "myvg" successfully exported
```

4. *내보낸 볼륨 그룹을 확인합니다.*

```
# pvscan
  PV /dev/sda1    is in exported VG myvg [17.15 GB / 7.15 GB free]
  PV /dev/sdc1    is in exported VG myvg [17.15 GB / 15.15 GB free]
  PV /dev/sdd1   is in exported VG myvg [17.15 GB / 15.15 GB free]
  ...
```

5. *시스템을 종료하고 볼륨 그룹을 구성하는 디스크를 분리하여 새 시스템에 연결합니다.*

6. *디스크를 새 시스템에 연결하고 볼륨 그룹을 가져와 새 시스템에서 액세스할 수 있도록 합니다.*

```
# vgimport myvg
```

참고

Cryostat import 명령의 **--force** 인수를 사용하여 물리 볼륨이 누락된 볼륨 그룹을 가져와서 나중에 **--removemissing** 명령을 실행할 수 있습니다.

7. 볼륨 그룹을 활성화합니다.

```
# vgchange -ay myvg
```

8. 파일 시스템을 마운트하여 사용할 수 있도록 합니다.

```
# mkdir -p /mnt/myvg/users
# mount /dev/myvg/users /mnt/myvg/users
```

추가 리소스

- **Cryostatimport(8), Cryostatexport(8), and Cryostatchange(8)** 매뉴얼 페이지

4.6. LVM 볼륨 그룹 제거

Cryostatremove 명령을 사용하여 기존 볼륨 그룹을 제거할 수 있습니다.

사전 요구 사항

- 볼륨 그룹에는 논리 볼륨이 포함되어 있지 않습니다. 볼륨 그룹에서 논리 볼륨을 제거하려면 *LVM 논리 볼륨 제거를* 참조하십시오.

절차

1. 볼륨 그룹이 클러스터형 환경에 있는 경우 다른 모든 노드에서 볼륨 그룹의 잠금 공간을 중지합니다. 제거를 수행하는 노드를 제외한 모든 노드에서 다음 명령을 사용합니다.

```
# vgchange --lockstop vg-name
```

잠금이 중지될 때까지 기다립니다.

2. *볼륨 그룹 제거:*

```
# vgremove vg-name
  Volume group "vg-name" successfully removed
```

추가 리소스

- *vgremove(8) man page*

5장. LVM 논리 볼륨 관리

논리 볼륨은 파일 시스템, 데이터베이스 또는 애플리케이션에서 사용할 수 있는 가상 블록 스토리지 장치입니다. LVM 논리 볼륨을 생성하기 위해 물리 볼륨(PV)을 볼륨 그룹(VG)으로 결합합니다. 그러면 LVM 논리 볼륨(LV)을 할당할 수 있는 디스크 공간 풀이 생성됩니다.

5.1. 논리 볼륨 개요

관리자는 표준 디스크 파티션과 달리 데이터를 삭제하지 않고 논리 볼륨을 늘리거나 줄일 수 있습니다. 볼륨 그룹의 물리 볼륨이 별도의 드라이브 또는 RAID 배열에 있는 경우 관리자는 스토리지 장치에 논리 볼륨을 분배할 수도 있습니다.

볼륨의 데이터보다 더 작은 용량으로 논리 볼륨을 축소하면 데이터가 손실될 수 있습니다. 또한 일부 파일 시스템은 축소할 수 없습니다. 유연성을 극대화하려면 현재 요구 사항을 충족하는 논리 볼륨을 생성하고 과도한 스토리지 용량을 할당되지 않은 상태로 둡니다. 필요에 따라 할당되지 않은 공간을 사용하도록 논리 볼륨을 안전하게 확장할 수 있습니다.

중요

AMD, Intel, ARM 시스템 및 IBM Power Systems 서버에서 부트 로더는 LVM 볼륨을 읽을 수 없습니다. /boot 파티션에 대해 LVM이 아닌 표준 디스크 파티션을 만들어야 합니다. IBM Z에서 zipl 부트 로더는 선형 매핑을 사용하여 LVM 논리 볼륨에서 /boot 를 지원합니다. 기본적으로 설치 프로세스는 물리 볼륨에 별도의 / boot 파티션을 사용하여 LVM 볼륨 내에 항상 / 및 스왑 파티션을 생성합니다.

다음은 다양한 유형의 논리 볼륨입니다.

선형 볼륨

선형 볼륨은 하나 이상의 물리 볼륨의 공간을 하나의 논리 볼륨으로 집계합니다. 예를 들어, 60GB 디스크가 두 개 있는 경우 120GB 논리 볼륨을 만들 수 있습니다. 물리 스토리지가 연결됩니다.

제거된 논리 볼륨

데이터를 LVM 논리 볼륨에 작성할 때 파일 시스템은 기본 물리 볼륨에 데이터를 배치합니다. 스트라이핑된 논리 볼륨을 생성하여 데이터를 물리 볼륨에 작성하는 방법을 제어할 수 있습니다. 대량의 순차적 읽기 및 쓰기의 경우 데이터 I/O의 효율성을 향상시킬 수 있습니다.

스트라이핑은 사전 결정된 물리 볼륨 수에 데이터를 라운드 로빈 방식으로 작성하여 성능을 향상시킵니다. 스트라이핑을 사용하면 I/O를 병렬로 수행할 수 있습니다. 경우에 따라 스트라이프의 추가 물리 볼륨마다 거의 인라인 성능을 얻을 수 있습니다.

RAID 논리 볼륨

LVM은 RAID 수준 0, 1, 4, 5, 6, 10을 지원합니다. RAID 논리 볼륨은 클러스터를 인식하지 않습니다. RAID 논리 볼륨을 생성할 때 LVM은 배열의 모든 데이터 또는 패리티 하위 볼륨에 대해 크기가 1인 메타데이터 하위 볼륨을 생성합니다.

씬 프로비저닝 논리 볼륨(thin volumes)

씬 프로비저닝된 논리 볼륨을 사용하여 사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다. 씬 프로비저닝된 볼륨 세트를 생성하면 요청된 전체 스토리지 양을 할당하는 대신 사용하는 항목을 할당할 수 있습니다.

스냅샷 볼륨

LVM 스냅샷 기능은 서비스가 중단되지 않고 특정 즉시 장치의 가상 이미지를 생성하는 기능을 제공합니다. 스냅샷을 만든 후 원본 장치(원본)가 변경되면 스냅샷 기능은 장치 상태를 재구성할 수 있도록 변경 전의 변경된 데이터 영역을 복사합니다.

씬 프로비저닝 스냅샷 볼륨

씬 프로비저닝 스냅샷 볼륨을 사용하여 동일한 데이터 볼륨에 저장할 가상 장치를 더 만들 수 있습니다. 씬 프로비저닝된 스냅샷은 지정된 시간에 캡처하려는 모든 데이터를 복사하지 않기 때문에 유용합니다.

볼륨 캐시

LVM은 느린 블록 장치를 위해 SSD 드라이브 또는 쓰기-스루 캐시와 같은 빠른 블록 장치를 사용할 수 있도록 지원합니다. 사용자는 캐시 논리 볼륨을 생성하여 기존 논리 볼륨의 성능을 개선하거나 크고 느린 장치가 연결된 작고 빠른 장치로 구성된 새 캐시 논리 볼륨을 생성할 수 있습니다.

5.2. LVM 논리 볼륨 생성

사전 요구 사항

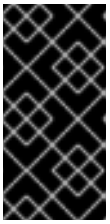
- *lvm2 패키지가 설치되어 있습니다.*
- *볼륨 그룹이 생성됩니다. 자세한 내용은 LVM 볼륨 그룹 생성 을 참조하십시오.*

절차

1. *논리 볼륨을 생성합니다.*

```
# lvcreate -n mylv -L 500M myvg
Logical volume "mylv" successfully created.
```

n 옵션을 사용하여 LV 이름을 mylv 로 설정하고 -L 옵션을 사용하여 Mb 단위로 LV 크기를 설정할 수 있지만 다른 단위를 사용할 수 있습니다. LV 유형은 기본적으로 선형이지만 --type 옵션을 사용하여 원하는 유형을 지정할 수 있습니다.



중요

VG에 요청된 크기 및 유형에 대해 사용 가능한 물리 확장 영역 수가 충분한 경우 명령이 실패합니다.

2. *요구 사항에 따라 다음 명령 중 하나를 사용하여 생성된 논리 볼륨을 확인합니다.*

   a. *lvs 명령은 논리 볼륨 정보를 구성 가능한 형식으로 제공하여 논리 볼륨당 한 줄을 표시합니다.*

```
# lvs
  LV   VG   Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
  mylv myvg -wi-ao----   500.00m
```

   b. *lvdisplay 명령은 크기, 레이아웃 및 매핑과 같은 논리 볼륨 속성을 고정된 형식으로 표시합니다.*

```
# lvdisplay -v /dev/myvg/mylv
  --- Logical volume ---
  LV Path                /dev/myvg/mylv
  LV Name                mylv
  VG Name                myvg
  LV UUID                YTnAk6-kMIT-c4pG-HBFZ-Bx7t-ePMk-7YjhaM
  LV Write Access        read/write
[..]
```

   c. *lvscan 명령은 시스템의 모든 논리 볼륨을 검색하여 나열합니다.*

```
# lvscan
  ACTIVE                 '/dev/myvg/mylv' [500.00 MiB] inherit
```

3. *논리 볼륨에 파일 시스템을 생성합니다. 다음 명령은 논리 볼륨에 xfs 파일 시스템을 생성합니다.*

```
# mkfs.xfs /dev/myvg/mylv
```

```
meta-data=/dev/myvg/mylv       isize=512    agcount=4, agsize=32000 blks
         =                     sectsz=512   attr=2, projid32bit=1
         =                     crc=1        finobt=1, sparse=1, rmapbt=0
         =                     reflink=1
data     =                       bsize=4096   blocks=128000, imaxpct=25
         =                     sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                     sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
Discarding blocks...Done.
```

4. 논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용량을 보고합니다.

```
# mount /dev/myvg/mylv /mnt

# df -h
Filesystem              1K-blocks  Used   Available Use% Mounted on

/dev/mapper/myvg-mylv   506528    29388  477140     6%   /mnt
```

추가 리소스

- lvcreate(8), lvdisplay(8), lvs(8), lvscan(8), lvm(8) 및 mkfs.xfs(8) 매뉴얼 페이지

5.3. RAID0 스트라이핑 논리 볼륨 생성

RAID0 논리 볼륨은 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 스트라이프 크기 단위로 분산합니다. 다음 절차에서는 디스크에서 데이터를 스트라이핑하는 mylv 라는 LVM RAID0 논리 볼륨을 생성합니다.

사전 요구 사항

1. 세 개 이상의 물리 볼륨을 생성했습니다. 물리 볼륨 생성에 대한 자세한 내용은 LVM 물리 볼륨 생성 을 참조하십시오.

2. 볼륨 그룹을 생성했습니다. 자세한 내용은 LVM 볼륨 그룹 생성 을 참조하십시오.

절차

1. 기존 볼륨 그룹에서 RAID0 논리 볼륨을 생성합니다. 다음 명령은 세 개의 스트라이프 크기와 스트라이프 크기가 4kB 인 볼륨 그룹 myvg 에서 RAID0 볼륨 mylv 를 생성합니다.

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv my_vg
  Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
  Logical volume "mylv" created.
```

2. *RAID0 논리 볼륨에 파일 시스템을 생성합니다. 다음 명령은 논리 볼륨에 ext4 파일 시스템을 생성합니다.*

```
# mkfs.ext4 /dev/my_vg/mylv
```

3. *논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용량을 보고합니다.*

```
# mount /dev/my_vg/mylv /mnt

# df
Filesystem            1K-blocks     Used  Available  Use% Mounted on
/dev/mapper/my_vg-mylv   2002684     6168  1875072    1%   /mnt
```

검증

- *생성된 RAID0 제거 논리 볼륨을 확인합니다.*

```
# lvs -a -o +devices,segtype my_vg
  LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert Devices Type
  mylv my_vg rwi-a-r--- 2.00g mylv_rimage_0(0),mylv_rimage_1(0),mylv_rimage_2(0) raid0
  [mylv_rimage_0] my_vg iwi-aor--- 684.00m /dev/sdf1(0) linear
  [mylv_rimage_1] my_vg iwi-aor--- 684.00m /dev/sdg1(0) linear
  [mylv_rimage_2] my_vg iwi-aor--- 684.00m /dev/sdh1(0) linear
```

5.4. LVM 논리 볼륨 이름

이 절차에서는 기존 논리 볼륨 mylv 의 이름을 mylv1 로 변경하는 방법을 설명합니다.

절차

1. *논리 볼륨이 현재 마운트된 경우 볼륨을 마운트 해제합니다.*

```
# umount /mnt
```

/mnt 를 마운트 지점으로 바꿉니다.

2. *기존 논리 볼륨의 이름을 변경합니다.*

```
# lvrename myvg mylv mylv1
Renamed "mylv" to "mylv1" in volume group "myvg"
```

장치에 대한 전체 경로를 지정하여 논리 볼륨의 이름을 변경할 수도 있습니다.

```
# lvrename /dev/myvg/mylv /dev/myvg/mylv1
```

추가 리소스

- *lvrename(8) man page*

5.5. 논리 볼륨에서 디스크 제거

이 절차에서는 디스크를 교체하거나 다른 볼륨의 일부로 디스크를 사용하는 기존 논리 볼륨에서 디스크를 제거하는 방법을 설명합니다.

디스크를 제거하려면 먼저 LVM 물리 볼륨의 Extent를 다른 디스크 또는 디스크 세트로 이동해야 합니다.

절차

1. *LV를 사용하는 경우 물리 볼륨의 사용 및 사용 가능한 공간을 확인합니다.*

```
# pvs -o+pv_used
  PV         VG    Fmt    Attr   PSize     PFree     Used
/dev/vdb1 myvg  lvm2   a--    1020.00m    0        1020.00m
/dev/vdb2 myvg  lvm2   a--    1020.00m    0        1020.00m
/dev/vdb3 myvg  lvm2   a--    1020.00m   1008.00m   12.00m
```

2. *데이터를 다른 물리 볼륨으로 이동합니다.*

   a. *기존 볼륨 그룹의 다른 물리 볼륨에 사용 가능한 확장 영역이 충분한 경우 다음 명령을 사용하여 데이터를 이동합니다.*

```
# pvmove /dev/vdb3
  /dev/vdb3: Moved: 2.0%
 ...
  /dev/vdb3: Moved: 79.2%
 ...
  /dev/vdb3: Moved: 100.0%
```

b. 기존 볼륨 그룹의 다른 물리 볼륨에 사용 가능한 확장 영역이 충분하지 않은 경우 다음 명령을 사용하여 새 물리 볼륨을 추가하고, 새로 생성된 물리 볼륨을 사용하여 볼륨 그룹을 확장한 다음 이 물리 볼륨으로 데이터를 이동합니다.

```
# pvcreate /dev/vdb4
  Physical volume "/dev/vdb4" successfully created

# vgextend myvg /dev/vdb4
  Volume group "myvg" successfully extended

# pvmove /dev/vdb3 /dev/vdb4
  /dev/vdb3: Moved: 33.33%
  /dev/vdb3: Moved: 100.00%
```

3. 물리 볼륨 제거:

```
# vgreduce myvg /dev/vdb3
Removed "/dev/vdb3" from volume group "myvg"
```

논리 볼륨에 오류가 있는 물리 볼륨이 포함된 경우 해당 논리 볼륨을 사용할 수 없습니다. 볼륨 그룹에서 누락된 물리 볼륨을 제거하려면 누락된 물리 볼륨에 할당된 논리 볼륨이 없는 경우 **reduce** 명령의 **--removemissing** 매개변수를 사용할 수 있습니다.

```
# vgreduce --removemissing myvg
```

추가 리소스

- **pvmove(8), ECDHEextend(8), vereduce(8), pvs(8)** 매뉴얼 페이지

5.6. LVM 논리 볼륨 제거

이 절차에서는 **myvg** 볼륨에서 기존 논리 볼륨 **/dev/myvg/mylv1** 을 제거하는 방법을 설명합니다.

절차

1. 논리 볼륨이 현재 마운트된 경우 볼륨을 마운트 해제합니다.

```
# umount /mnt
```

2. 논리 볼륨이 클러스터형 환경에 있는 경우 활성화되어 있는 모든 노드에서 논리 볼륨을 비활성화합니다. 이러한 각 노드에서 다음 명령을 사용합니다.

```
# lvchange --activate n vg-name/lv-name
```

3. **lvremove** 유틸리티를 사용하여 논리 볼륨을 제거합니다.

```
# lvremove /dev/myvg/mylv1

Do you really want to remove active logical volume "mylv1"? [y/n]: y
Logical volume "mylv1" successfully removed
```

참고

이 경우 논리 볼륨이 비활성화되지 않았습니다. 논리 볼륨을 제거하기 전에 명시적으로 비활성화한 경우 활성 논리 볼륨을 제거할지 여부를 확인하는 프롬프트가 표시되지 않습니다.

추가 리소스

- **lvremove(8) man page**

5.7. RHEL 시스템 역할을 사용하여 LVM 논리 볼륨 관리

스토리지 역할을 사용하여 다음 작업을 수행합니다.

- 여러 디스크로 구성된 볼륨 그룹에 **LVM** 논리 볼륨을 만듭니다.
- 논리 볼륨에 지정된 라벨을 사용하여 **ext4** 파일 시스템을 생성합니다.
- **ext4** 파일 시스템을 영구적으로 마운트합니다.

사전 요구 사항

- *스토리지 역할을 포함한 **Ansible Playbook***

5.7.1. 스토리지 RHEL 시스템 역할을 사용하여 논리 볼륨 관리

*예제 **Ansible** 플레이북은 **storage** 역할을 적용하여 볼륨 그룹에 **LVM** 논리 볼륨을 생성합니다.*

사전 요구 사항

- *컨트롤 노드 및 관리형 노드를 준비했습니다.*
- *관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.*
- *관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.*

절차

1. *다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.*

```
- hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt/dat
```

   - *myvg 볼륨 그룹은 /dev/sda,/dev/sdb, /dev/sdc 디스크로 구성됩니다.*
   - *myvg 볼륨 그룹이 이미 있는 경우 Playbook은 볼륨 그룹에 논리 볼륨을 추가합니다.*

- *myvg 볼륨 그룹이 없으면 플레이북에서 해당 그룹을 생성합니다.*

- *플레이북은 mylv 논리 볼륨에 Ext4 파일 시스템을 생성하고 /mnt 에 파일 시스템을 영구적으로 마운트합니다.*

2. *플레이북 구문을 확인합니다.*

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. *플레이북을 실행합니다.*

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- */usr/share/ansible/roles/rhel-system-roles.storage/README.md file*

- */usr/share/doc/rhel-system-roles/storage/ 디렉터리*

5.7.2. 추가 리소스

- *스토리지 역할에 대한 자세한 내용은 RHEL 시스템 역할을 사용하여 로컬 스토리지 관리를 참조하십시오.*

5.8. LVM 볼륨 그룹 제거

Cryostatremove 명령을 사용하여 기존 볼륨 그룹을 제거할 수 있습니다.

사전 요구 사항

- *볼륨 그룹에는 논리 볼륨이 포함되어 있지 않습니다. 볼륨 그룹에서 논리 볼륨을 제거하려면 LVM 논리 볼륨 제거를 참조하십시오.*

절차

1. *볼륨 그룹이 클러스터형 환경에 있는 경우 다른 모든 노드에서 볼륨 그룹의 잠금 공간을 중지합니다. 제거를 수행하는 노드를 제외한 모든 노드에서 다음 명령을 사용합니다.*

```
# vgchange --lockstop vg-name
```

   잠금이 중지될 때까지 기다립니다.

2. *볼륨 그룹 제거:*

```
# vgremove vg-name
  Volume group "vg-name" successfully removed
```

추가 리소스

- ***vgremove(8) man page***

6장. 논리 볼륨의 크기 수정

논리 볼륨을 만든 후에는 볼륨의 크기를 수정할 수 있습니다.

6.1. 논리 볼륨 및 파일 시스템 확장

lvextend 명령을 사용하여 **LV(Logical volume)**를 확장할 수 있습니다. **LV**를 확장하려는 양 또는 **LV**를 확장한 후 **LV**를 원하는 크기로 지정할 수 있습니다. **lvextend** 명령의 **-r** 옵션을 사용하여 **LV**와 함께 기본 파일 시스템을 확장합니다.

주의

lvresize 명령을 사용하여 논리 볼륨을 확장할 수도 있지만 이 명령은 우발적인 축소를 보장하지 않습니다.

사전 요구 사항

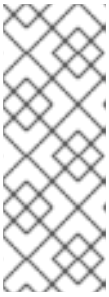
- 파일 시스템이 있는 기존 논리 볼륨(**LV**)이 있습니다. **df -Th** 명령을 사용하여 파일 시스템 유형과 크기를 결정합니다. 논리 볼륨 및 파일 시스템을 만드는 방법에 대한 자세한 내용은 *LVM 논리 볼륨 생성* 을 참조하십시오.
- 볼륨 그룹에 **LV** 및 파일 시스템을 늘리기 위한 충분한 공간이 있어야 합니다. 사용 가능한 공간을 확인하려면 **-o name,vgfree** 명령을 사용합니다. 볼륨 그룹 생성에 대한 자세한 내용은 *LVM 볼륨 그룹 생성* 을 참조하십시오.

절차

1. 선택 사항: 볼륨 그룹에 **LV**를 확장할 공간이 충분하지 않은 경우 볼륨 그룹에 새 물리 볼륨을 추가합니다.

```
# vgextend myvg /dev/vdb3
Physical volume "/dev/vdb3" successfully created.
Volume group "myvg" successfully extended.
```

2. **LV** 및 파일 시스템을 확장합니다.



참고

-r 인수 없이 **lvextend** 명령을 사용하면 **LV**만 확장합니다. 기본 **XFS** 파일 시스템을 확장하려면 **XFS** 파일 시스템의 크기 감소를 참조하십시오. **polkit2** 파일 시스템의 경우 **Growing a Cryostat2** 파일 시스템 및 **ext4** 파일 시스템의 크기를 참조하십시오. **ext4** 파일 시스템 복구를 참조하십시오.

참고

L 옵션을 사용하여 **LV**를 새 크기로 확장하고 **-l** 옵션을 사용하여 늘릴 논리 볼륨의 크기에 따라 확장 영역 수를 지정합니다.

```
# lvextend -r -L 3G /dev/myvg/mylv
fsck from util-linux 2.32.1
/dev/mapper/myvg-mylv: clean, 11/131072 files, 26156/524288 blocks
  Size of logical volume myvg/mylv changed from 2.00 GiB (512 extents) to 3.00 GiB (768 extents).
  Logical volume myvg/mylv successfully resized.
resize2fs 1.45.6 (20-Mar-2020)
Resizing the filesystem on /dev/mapper/myvg-mylv to 786432 (4k) blocks.
The filesystem on /dev/mapper/myvg-mylv is now 786432 (4k) blocks long.
```

mylv 논리 볼륨을 확장하여 **myvg** 볼륨 그룹의 할당되지 않은 모든 공간을 채울 수도 있습니다.

```
# lvextend -l +100%FREE /dev/myvg/mylv
 Size of logical volume myvg/mylv changed from 10.00 GiB (2560 extents) to 6.35 TiB (1665465 extents).
 Logical volume myvg/mylv successfully resized.
```

검증

- 파일 시스템 및 **LV**가 확장되었는지 확인합니다.

```
# df -Th
Filesystem          Type      Size  Used Avail Use% Mounted on
devtmpfs            devtmpfs  1.9G     0  1.9G   0% /dev
tmpfs               tmpfs     1.9G     0  1.9G   0% /dev/shm
tmpfs               tmpfs     1.9G  8.6M  1.9G   1% /run
tmpfs               tmpfs     1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/mapper/rhel-root xfs        45G  3.7G   42G   9% /
/dev/vda1           xfs       1014M  369M  646M  37% /boot
tmpfs               tmpfs     374M     0  374M   0% /run/user/0
/dev/mapper/myvg-mylv xfs       2.0G   47M  2.0G   3% /mnt/mnt1
```

추가 리소스

추가 리소스

- *vgextend(8), lvextend(8), and xfs_growfs(8) man pages*

6.2. 논리 볼륨 및 파일 시스템 감소

lvreduce 명령과 **resizefs** 옵션을 사용하여 논리 볼륨 및 해당 파일 시스템을 줄일 수 있습니다.

축소 중인 논리 볼륨에 파일 시스템이 포함된 경우 데이터 손실을 방지하려면 파일 시스템이 축소되는 논리 볼륨의 공간을 사용하지 않도록 해야 합니다. 따라서 논리 볼륨에 파일 시스템이 포함된 경우 **lvreduce** 명령의 **--resizefs** 옵션을 사용합니다.

--resizefs 를 사용하는 경우**lvreduce** 는 논리 볼륨을 축소하기 전에 파일 시스템을 줄임으로써 파일 시스템을 줄입니다. 파일 시스템이 가득 차거나 축소를 지원하지 않기 때문에 파일 시스템을 줄이는 경우 **lvreduce** 명령이 실패하고 논리 볼륨을 줄이지 않습니다.

주의

대부분의 경우 **lvreduce** 명령은 가능한 데이터 손실에 대해 경고하고 확인을 요청합니다. 그러나 논리 볼륨이 비활성 상태이거나 **--resizefs** 옵션이 사용되지 않는 경우와 같이 데이터 손실을 방지하기 위해 이러한 확인 프롬프트에 의존하지 않아야 합니다.

lvreduce 명령의 **--test** 옵션을 사용하면 이 옵션이 파일 시스템을 검사하거나 파일 시스템 크기 조정을 테스트하지 않기 때문에 작업이 안전한지 여부를 표시하지 않습니다.

사전 요구 사항

- 논리 볼륨의 파일 시스템은 축소를 지원합니다. **df -Th** 명령을 사용하여 파일 시스템 유형과 크기를 결정합니다.

참고

예를 들어, **XFS2** 및 **XFS** 파일 시스템은 축소를 지원하지 않습니다.

- 기본 파일 시스템은 축소되는 **LV**의 공간을 사용하지 않습니다.

절차

1. 다음 옵션 중 하나를 사용하여 **mylv** 논리 볼륨 및 해당 파일 시스템을 **myvg** 볼륨 그룹에서 축소합니다.

   - **LV**와 파일 시스템을 원하는 값으로 줄입니다.

     ```
     # lvreduce --resizefs -L 500M myvg/mylv
     File system ext4 found on myvg/mylv.
     File system size (2.00 GiB) is larger than the requested size (500.00 MiB).
     File system reduce is required using resize2fs.
     ...
     Logical volume myvg/mylv successfully resized.
     ```

   - 논리 볼륨 및 파일 시스템에서 **64MB**를 줄입니다.

     ```
     # lvreduce --resizefs -L -64M myvg/mylv
     File system ext4 found on myvg/mylv.
     File system size (500.00 MiB) is larger than the requested size (436.00 MiB).
     File system reduce is required using resize2fs.
     ...
     Logical volume myvg/mylv successfully resized
     ```

추가 리소스

- **lvreduce(8)** 매뉴얼 페이지

6.3. 스트라이핑된 논리 볼륨 확장

필요한 크기와 함께 **lvextend** 명령을 사용하여 제거된 **LV(Logical Volume)**를 확장할 수 있습니다.

사전 요구 사항

1. 스트라이프를 지원하기 위해 볼륨 그룹**(VG)**을 구성하는 기본 물리 볼륨**(PV)**에 충분한 여유 공간이 있습니다.

절차

1. *선택 사항: 볼륨 그룹을 표시합니다.*

```
# vgs
  VG      #PV #LV #SN Attr   VSize   VFree
  myvg      2   1   0 wz--n- 271.31G 271.31G
```

2. *선택 사항: 볼륨 그룹의 전체 공간을 사용하여 스트라이프를 만듭니다.*

```
# lvcreate -n stripe1 -L 271.31G -i 2 myvg
  Using default stripesize 64.00 KB
  Rounding up size to full physical extent 271.31 GiB
```

3. *선택 사항: 새 물리 볼륨을 추가하여 **myvg** 볼륨 그룹을 확장합니다.*

```
# vgextend myvg /dev/sdc1
  Volume group "myvg" successfully extended
```

스트라이프 유형 및 사용된 공간에 따라 충분한 물리 볼륨을 추가하려면 이 단계를 반복합니다. 예를 들어 전체 볼륨 그룹을 사용하는 양방향 스트라이프의 경우 두 개 이상의 물리 볼륨을 추가해야 합니다.

4. ***myvg VG**의 일부인 제거된 논리 볼륨 스트라이프**1** 을 확장합니다.*

```
# lvextend myvg/stripe1 -L 542G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 542.00 GB
  Logical volume stripe1 successfully resized
```

***stripe1** 논리 볼륨을 확장하여 **myvg** 볼륨 그룹의 할당되지 않은 모든 공간을 채울 수도 있습니다.*

```
# lvextend -l+100%FREE myvg/stripe1
  Size of logical volume myvg/stripe1 changed from 1020.00 MiB (255 extents) to <2.00 GiB (511 extents).
  Logical volume myvg/stripe1 successfully resized.
```

검증

- *확장 스트라이핑된 **LV**의 새 크기를 확인합니다.*

```
# lvs
  LV      VG     Attr     LSize   Pool   Origin Data%  Move Log Copy%  Convert
  stripe1 myvg   wi-ao---- 542.00 GB
```

7장. LVM 보고서 사용자 정의

LVM은 사용자 지정 보고서를 생성하고 보고서의 출력을 필터링하는 다양한 구성 및 명령줄 옵션을 제공합니다. 출력을 정렬하고 단위를 지정하고 선택 기준을 사용하고 lvm.conf 파일을 업데이트하여 LVM 보고서를 사용자 지정할 수 있습니다.

7.1. LVM 디스플레이의 제어 형식

pvs,lvs 또는 Cryostats를 사용할지 여부에 관계없이 이러한 명령은 표시된 기본 필드 세트와 정렬 순서를 결정합니다. 다음 명령을 실행하여 이러한 명령의 출력을 제어할 수 있습니다.

절차

- -o 옵션을 사용하여 LVM 디스플레이의 기본 필드를 변경합니다.

```
# pvs -o pv_name,pv_size,pv_free
  PV         PSize  PFree
  /dev/vdb1  17.14G 17.14G
  /dev/vdb2  17.14G 17.09G
  /dev/vdb3  17.14G 17.14G
```

- -O 옵션을 사용하여 LVM 표시를 정렬합니다.

```
# pvs -o pv_name,pv_size,pv_free -O pv_free
  PV         PSize  PFree
  /dev/vdb2 17.14G 17.09G
  /dev/vdb1  17.14G 17.14G
  /dev/vdb3  17.14G 17.14G
```

- - 문자와 함께 -O 인수를 사용하여 역방향 정렬을 표시합니다.

```
# pvs -o pv_name,pv_size,pv_free -O -pv_free
  PV         PSize  PFree
  /dev/vdb1  17.14G 17.14G
  /dev/vdb3  17.14G 17.14G
  /dev/vdb2  17.14G 17.09G
```

추가 리소스

- lvmreport(7), lvs(8), Cryostats(8), pvs(8) 도움말 페이지

- *LVM 보고서 표시 단위 지정*
- *LVM 구성 파일 사용자 정의*

7.2. LVM 보고서 디스플레이의 단위 지정

report 명령의 --units 인수를 지정하여 기본 2 또는 기본 10 단위로 LVM 장치의 크기를 볼 수 있습니다.

기본 2 단위

기본 단위는 1024의 배수인 2의 powers로 표시됩니다. 사람이 읽을 수 있는(r) < 및 > 반올림 표시기, 바이트(b), 섹터(s), 킬로바이트(s), 메가바이트(m), 기가바이트(g), 테라바이트(p) 및 사람이 읽을 수 있는 (h)를 사용하여 지정할 수 있습니다.

기본 디스플레이는 r 입니다. --units 가 지정되지 않은 경우입니다. /etc/lvm/lvm.conf 파일의 global 섹션에서 units 매개변수를 설정하여 기본값을 덮어쓸 수 있습니다.

기본 10 단위

단위 사양(R,B,S,K,M,G,T,E,H)을 대문자로 표시할 단위를 1000으로 지정할 수 있습니다.

절차

- 기본 2GB 단위의 LVM 단위를 지정합니다.

```
# pvs --units g /dev/vdb
  PV         VG    Fmt  Attr PSize   PFree
  /dev/vdb  myvg  lvm2 a--  931.00g 930.00g

# vgs --units g myvg
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 931.00g 931.00g

# lvs --units g myvg
  LV    VG   Attr      LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg  wi-a---- 1.OOg
```

- 출력에서 < 또는 > 접두사와 함께 r 옵션을 사용하여 LVM의 실제 크기를 지정합니다.

```
# vgs --units g myvg
```

```
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 931.00g 930.00g

# vgs --units r myvg
  VG   #PV #LV #SN Attr VSize    VFree
  myvg   1   1   0 wz-n <931.00g <930.00

# vgs myvg
  VG   #PV #LV #SN Attr VSize    VFree
  myvg   1   1   0 wz-n <931.00g <930.00g
```

r 단위는 h (human-readable)와 유사하게 작동하지만 보고된 값은 < 또는 > 접두사를 가져와서 실제 크기가 표시된 크기보다 약간 많거나 적다는 것을 나타냅니다. LVM은 10진수 값을 반올림하여 존재하지 않는 크기를 보고합니다.

또한 --units g 또는 기타 --units 가 항상 올바른 크기를 표시하지 않는 방법을 보여줍니다. 또한 표시된 크기가 정확하지 않음을 나타내는 <인 r 의 기본 용도를 보여줍니다. 이 예에서는 VG 크기가 정확히 기가바이트의 배수가 아니며 .01도 분수를 정확하게 표현하지 않기 때문에 값이 정확하지 않습니다.

- *기본 10GB 단위의 LVM 단위를 지정합니다.*

```
# pvs --units G /dev/vdb
  PV         VG   Fmt  Attr  PSize   PFree
  /dev/vdb  myvg lvm2 a--   999.65G 998.58G

# vgs --units G myvg
  VG   #PV #LV #SN Attr VSize   VFree
  myvg   1   1   0 wz-n 999.65G 998.58G

# lvs --units G myvg
  LV    VG   Attr      LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg wi-a---- 1.07G
```

- *512바이트 또는 사용자 지정 단위로 정의된 섹터(s)를 지정합니다. 다음 예제는 pvs 명령의 출력을 여러 섹터로 표시합니다.*

```
# pvs --units s
  PV         VG     Fmt  Attr PSize       PFree
  /dev/vdb   myvg   lvm2 a--  1952440320S 1950343168S
```

- *메가바이트(m)를 지정합니다. 다음 예제는 PVC 명령 의 출력을 4MB 단위로 표시합니다.*

```
# pvs --units 4m
  PV         VG     Fmt  Attr PSize      PFree
  /dev/vdb   myvg   lvm2 a--  238335.00U 238079.00U
```

7.3. LVM 구성 파일 사용자 정의

lvm.conf 파일을 편집하여 특정 스토리지 및 시스템 요구 사항에 따라 **LVM**을 사용자 지정할 수 있습니다. 예를 들어 **lvm.conf** 를 사용하여 필터 설정을 수정하거나, 볼륨 그룹 자동 활성화를 구성하거나, **thin** 풀을 관리하거나, 스냅샷을 자동으로 확장할 수 있습니다.

절차:

1. 기본 **lvm.conf** 파일을 표시합니다.

```
# lvmconfig --typeconfig default --withcomments
```

기본적으로 **lvm.conf** 파일에는 가능한 설정을 표시하는 주석만 포함되어 있습니다.

2. **lvm.conf** 의 설정 주석을 제거하여 요구 사항에 따라 **lvm.conf** 파일을 사용자 지정합니다. 다음 설정은 특정 명령의 기본 표시를 변경하는 데 중점을 둡니다.

- **lvm.conf** 파일에서 지정된 필드만 출력하도록 **lvs_cols** 매개변수를 조정합니다.

```
{
  ...
  lvs_cols="lv_name,vg_name,lv_attr"
  ...
}
```

lvs -o lv_name,vg_name,lv_attr 명령 대신 이 옵션을 사용하여 **-o** 옵션을 자주 사용하지 않도록 합니다.

- **lvm.conf** 파일에서 **compact_output=1** 설정을 사용하여 **pvs, Cryostats , lvs** 명령의 빈 필드를 출력하지 않습니다.

```
{
  ...
  compact_output = 1
  ...
}
```

3. *lvm.conf 파일을 수정한 후 기본값을 확인합니다.*

```
# lvmconfig --typeconfig diff
```

추가 리소스

- *lvm.conf(5) man page*

7.4. LVM 선택 기준 정의

선택 기준은 < field> <operator> <value > 형식으로 된 문 세트이며 비교 연산자를 사용하여 특정 필드의 값을 정의합니다. 그런 다음 선택 기준과 일치하는 오브젝트가 처리되거나 표시됩니다. 설명은 논리 및 그룹화 연산자로 결합됩니다. 선택 기준을 정의하려면 -S 또는 --select 옵션 다음에 하나 이상의 문을 사용합니다.

일부 LVM 명령은 -S 옵션을 지원하여 특정 속성을 기반으로 처리할 오브젝트를 선택합니다. 이러한 오브젝트는 물리 볼륨(PV), 볼륨 그룹(VG) 또는 논리 볼륨(LV)일 수 있습니다.

S 옵션은 각 오브젝트의 이름을 지정하는 대신 처리할 오브젝트를 설명하는 방식으로 작동합니다. 이는 많은 오브젝트를 처리할 때 도움이 되며 각 개체를 별도로 찾아서 이름을 지정하거나 복잡한 특성을 가진 오브젝트를 검색할 때 유용합니다. 선택 옵션은 여러 이름을 입력하지 않도록 바로 가기로 사용할 수도 있습니다.

lvs -S help 명령을 사용하여 전체 필드 및 가능한 Operator 세트를 확인합니다. 해당 명령의 세부 정보를 확인하려면 lvs 를 보고 또는 처리 명령으로 바꿉니다.

LVM 보고 및 처리 명령과 함께 선택 기준을 사용하여 선택한 기준을 충족하는 오브젝트만 표시하거나 처리합니다.

- *보고 명령에는 pvs, Cryostats ,lvs,pvdisplay, Cryostatdisplay ,lvdisplay, dmsetup info -c 가 포함됩니다.*
- *처리 명령에는 pvchange, Cryostatchange,lvchange, Cryostatimport, Cryostatexport, Cryostatremove, lvremove 가 포함됩니다.*

절차

- ***pvs** 명령을 사용한 선택 기준의 예:*

```
# pvs
  PV          VG   Fmt  Attr PSize    PFree
  /dev/nvme2n1       lvm2 ---     1.00g   1.00g
  /dev/vdb1     myvg lvm2 a--  1020.00m 396.00m
  /dev/vdb2     myvg lvm2 a--  1020.00m 896.00m
```

```
# pvs -S name=~nvme
  PV          Fmt  Attr PSize PFree
  /dev/nvme2n1 lvm2 ---  1.00g 1.00g
```

```
# pvs -S vg_name=myvg
  PV         VG   Fmt  Attr PSize    PFree
  /dev/vdb1  myvg lvm2 a--  1020.00m 396.00m
  /dev/vdb2  myvg lvm2 a--  1020.00m 896.00m
```

- ***lvs** 명령을 사용한 선택 기준의 예:*

```
# lvs
  LV    VG   Attr       LSize   Cpy%Sync
  mylv  myvg -wi-a----- 200.00m
  lvol0 myvg -wi-a----- 100.00m
  lvol1 myvg -wi-a----- 100.00m
  lvol2 myvg -wi------- 100.00m
  rr    myvg rwi-a-r--- 120.00m 100.00
```

```
# lvs -S 'size > 100m && size < 200m'
  LV   VG   Attr       LSize   Cpy%Sync
  rr   myvg rwi-a-r--- 120.00m 100.00
```

```
# lvs -S name=~lvol[02]
  LV    VG   Attr       LSize
  lvol0 myvg -wi-a----- 100.00m
  lvol2 myvg -wi------- 100.00m
```

```
# lvs -S segtype=raid1
  LV   VG   Attr       LSize   Cpy%Sync
  rr   myvg rwi-a-r--- 120.00m 100.00
```

- *고급 예:*

```
# lvchange --addtag mytag -S active=1
  Logical volume myvg/mylv changed.
  Logical volume myvg/lvol0 changed.
  Logical volume myvg/lvol1 changed.
  Logical volume myvg/rr changed.
```

```
# lvs -a -o lv_name,vg_name,attr,size,pool_lv,origin,role -S 'name!~_pmspare'
  LV          VG      Attr       LSize Pool Origin Role
  thin1       example Vwi-a-tz-- 2.00g tp          public,origin,thinorigin
  thin1s      example Vwi---tz-- 2.00g tp   thin1  public,snapshot,thinsnapshot
  thin2       example Vwi-a-tz-- 3.00g tp          public
  tp          example twi-aotz-- 1.00g             private
  [tp_tdata] example Twi-ao---- 1.00g              private,thin,pool,data
  [tp_tmeta] example ewi-ao---- 4.00m              private,thin,pool,metadata
```

```
# lvchange --setactivationskip n -S 'role=thinsnapshot && origin=thin1'
  Logical volume myvg/thin1s changed.
```

```
# lvs -a -S 'name=~_tmeta && role=metadata && size <= 4m'
  LV          VG      Attr       LSize
  [tp_tmeta] myvg   ewi-ao---- 4.00m
```

추가 리소스

- ***lvmreport(7)** 도움말 페이지*

8장. 공유 스토리지에서 LVM 구성

공유 스토리지는 동시에 여러 노드에서 액세스할 수 있는 스토리지입니다. LVM을 사용하여 공유 스토리지를 관리할 수 있습니다. 공유 스토리지는 일반적으로 클러스터 및 고가용성 설정에 사용되며 시스템에 공유 스토리지가 표시되는 방법에 대한 두 가지 일반적인 시나리오가 있습니다.

- LVM 장치는 호스트에 연결되어 사용할 게스트 VM에 전달됩니다. 이 경우 게스트 VM에서만 해당 장치를 호스트에서 사용하지 않습니다.
- 시스템은 SAN(Storage Area Network)에 연결되어 있으며, 예를 들어 파이버 채널을 사용하고 SAN LUN은 여러 시스템에 표시됩니다.

8.1. VM 디스크에 대한 LVM 구성

VM 스토리지가 호스트에 노출되지 않도록 LVM 장치 액세스 및 LVM 시스템 ID 를 구성할 수 있습니다. 호스트에서 해당 장치를 제외하여 호스트의 LVM이 표시되지 않거나 게스트 VM에 전달된 장치를 사용하여 이 작업을 수행할 수 있습니다. VG의 LVM 시스템 ID 를 게스트 VM과 일치하도록 설정하여 호스트에서 VM의 VG를 실수로 사용하지 않도록 보호할 수 있습니다.

절차

1. lvm.conf 파일에서 장치를 제외하도록 경로를 필터링합니다.

```
filter = [ "r|^path_to_device$|" ]
```

2. 선택 사항: LVM 장치를 추가로 보호할 수 있습니다.

   a. lvm.conf 파일의 호스트와 VM 모두에서 LVM 시스템 ID 기능을 설정합니다.

   ```
   system_id_source = "uname"
   ```

   b. VM 시스템 ID 와 일치하도록 VG의 시스템 ID 를 설정합니다. 이렇게 하면 게스트 VM만 VG를 활성화할 수 있습니다.

   ```
   $ vgchange --systemid <VM_system_id> <VM_vg_name>
   ```

8.2. 한 시스템에서 SAN 디스크를 사용하도록 LVM 구성

잘못된 시스템에서 **SAN LUN**을 사용하지 않도록 하려면 **lvm.conf** 필터에서 해당 디스크를 사용할 수 있는 시스템 하나를 제외한 모든 시스템의 해당 디스크를 제외합니다.

모든 시스템에서 시스템 **ID**를 구성하고 해당 시스템을 사용하는 시스템과 일치하도록 **VG**의 시스템 **ID**를 설정하여 잘못된 시스템에서 **VG**를 사용하지 않도록 보호할 수도 있습니다.

절차

1. **lvm.conf** 파일에서 제외할 장치의 경로를 필터링합니다.

```
filter = [ "r|^path_to_device$|" ]
```

2. **lvm.conf** 파일에서 **LVM** 시스템 **ID** 기능을 설정합니다.

```
system_id_source = "uname"
```

3. 이 **VG**를 사용하여 시스템의 시스템 **ID** 와 일치하도록 **VG**의 시스템 **ID** 를 설정합니다.

```
$ vgchange --systemid <system_id> <vg_name>
```

8.3. 장애 조치에 SAN 디스크를 사용하도록 LVM 구성

장애 조치(**failover**)를 위해 시스템 간에 이동하도록 **LUN**을 구성할 수 있습니다. **lvm.conf** 필터를 구성하고 각 시스템에서 **LVM** 시스템 **ID**를 구성하여 사용할 수 있는 모든 시스템에 **LUN**을 포함하도록 **LVM**을 설정할 수 있습니다.

다음 절차에서는 페일오버를 위해 **LVM** 설정을 완료하고 **VG**를 시스템 간에 이동하려면 **VG**를 사용할 수 있는 시스템의 시스템 **ID**와 일치하도록 **VG**의 시스템 **ID**를 자동으로 수정하는 **pacemaker** 및 **LVM** 활성화 리소스 에이전트를 구성해야 합니다. 자세한 내용은 고가용성 클러스터 구성 및 관리를 참조하십시오.

절차

1. **lvm.conf** 파일에서 장치를 제외하도록 경로를 필터링합니다.

```
filter = [ "a|^path_to_device$|" ]
```

2. *lvm.conf* 파일의 모든 시스템에서 *LVM* 시스템 *ID* 기능을 설정합니다.

```
system_id_source = "uname"
```

8.4. 여러 시스템에서 SAN 디스크를 공유하도록 LVM 구성

lvmlockd 데몬과 *dlm* 또는 *sanlock* 과 같은 잠금 관리자를 사용하면 여러 시스템에서 *SAN* 디스크의 공유 *VG*에 액세스할 수 있습니다. 특정 명령은 사용되는 잠금 관리자 및 운영 체제에 따라 다를 수 있습니다. 다음 절차에서는 여러 시스템에서 *SAN* 디스크를 공유하도록 *LVM*을 구성하는 데 필요한 단계의 개요를 설명합니다.

주의

pacemaker 를 사용하는 경우 *고가용성 클러스터 구성 및 관리에* 표시된 *pacemaker* 단계를 사용하여 시스템을 구성하고 시작해야 합니다.

절차

1. *해당 항목을 사용할 모든 시스템의 LUN을 포함하도록 lvm.conf 필터를 구성합니다.*

```
filter = ["a|^path_to_device$|" ]
```

2. *모든 시스템에서 lvmlockd 데몬을 사용하도록 lvm.conf 파일을 구성합니다.*

```
use_lvmlockd=1
```

3. *모든 시스템에서 lvmlockd 데몬 파일을 시작합니다.*

4. *모든 시스템에서 dlm 또는 sanlock 과 같이 lvmlockd 와 함께 사용하려면 잠금 관리자를 시작합니다.*

5. *protect create --shared 명령을 사용하여 새 공유 VG를 만듭니다.*

6. 모든 시스템에서 **Cryostat change --lockstart** 및 **Cryostat change --lock stop** 명령을 사용하여 기존 공유 **VG**에 대한 액세스를 시작하고 중지합니다.

추가 리소스

- **lvmlockd(8)** 매뉴얼 페이지

8.5. 스토리지 RHEL 시스템 역할을 사용하여 공유 LVM 장치 생성

여러 시스템이 동일한 스토리지에 동시에 액세스하도록 하려면 스토리지 **RHEL** 시스템 역할을 사용하여 공유 **LVM** 장치를 생성할 수 있습니다.

이는 다음과 같은 주요 이점을 가져올 수 있습니다.

- 리소스 공유
- 스토리지 리소스 관리의 유연성
- 스토리지 관리 작업 간소화

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.
- **lvmlockd** 가 구성되어 있습니다. 자세한 내용은 여러 머신 간에 **SAN** 디스크를 공유하도록 **LVM** 구성 을 참조하십시오.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: ~/playbook.yml)을 생성합니다.

```
---
- name: Create shared LVM device
  hosts: managed-node-01.example.com
  become: true
  tasks:
    - name: Create shared LVM device
      ansible.builtin.include_role:
        name: rhel-system-roles.storage
      vars:
        storage_pools:
          - name: vg1
            disks: /dev/vdb
            type: lvm
            shared: true
            state: present
            volumes:
              - name: lv1
                size: 4g
                mount_point: /opt/test1
        storage_safe_mode: false
        storage_use_partitions: true
```

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- /usr/share/ansible/roles/rhel-system-roles.storage/README.md file
- /usr/share/doc/rhel-system-roles/storage/ 디렉터리

9장. RAID 논리 볼륨 구성

LVM(Logical Volume Manager)을 사용하여 RAID(Redundant Array of Independent Disks) 볼륨을 생성하고 관리할 수 있습니다.

9.1. RAID 논리 볼륨

LVM(Logical volume manager)은 0, 1, 4, 5, 6, 10의 RAID(Red-zero Array of Disk)를 지원합니다. LVM RAID 볼륨에는 다음과 같은 특징이 있습니다.

- LVM에서는 MD(Multiple Devices) 커널 드라이버를 활용하는 RAID 논리 볼륨을 생성하고 관리합니다.
- RAID1 이미지를 배열에서 임시로 분할하고 나중에 다시 배열에 병합할 수 있습니다.
- LVM RAID 볼륨은 스냅샷을 지원합니다.

기타 특성은 다음과 같습니다.

클러스터

RAID 논리 볼륨은 클러스터를 인식하지 않습니다.

하나의 시스템에서만 RAID 논리 볼륨을 생성하고 활성화할 수 있지만 두 개 이상의 시스템에서 동시에 활성화할 수 없습니다.

subvolumes

RAID 논리 볼륨(LV)을 생성할 때 LVM은 배열의 모든 데이터 또는 패리티 하위 볼륨마다 크기가 한 범위인 메타데이터 하위 볼륨을 생성합니다.

예를 들어 2방향 RAID1 배열을 생성하면 두 개의 메타데이터 하위 볼륨(lv_rmeta_0 및 lv_rmeta_1)과 두 개의 데이터 하위 볼륨(lv_rimage_0 및 lv_rimage_1)이 생성됩니다. 마찬가지로 3방향 스트라이프 및 하나의 암시적 패리티 장치를 생성하면 RAID4는 4개의 메타데이터 하위 볼륨(lv_rmeta_0,lv_rmeta_1,lv_rmeta_2, lv_rmeta_3)과 데이터 하위 볼륨(lv_rimage_ 0)을 생성합니다. lv_rimage_1,lv_rimage_2, lv_rimage_3).

무결성

RAID 장치가 실패하거나 소프트 손상이 발생할 때 데이터를 손실할 수 있습니다. 데이터 스토리지의 소프트 손상은 스토리지 장치에서 검색한 데이터가 해당 장치에 기록된 데이터와 다르다는 것을 의미합니다. RAID LV에 무결성을 추가하면 소프트 손상이 감소하거나 방지할 수 있습니다. 자세한 내용은 DM 무결성을 사용하여 RAID LV 생성을 참조하십시오.

9.2. RAID 수준 및 선형 지원

다음은 레벨 0, 1, 4, 5, 6, 10 및 선형을 포함하여 RAID에서 지원하는 구성입니다.

수준 0

RAID 레벨 0은 종종 스트라이핑이라고 하며 성능 지향 스트라이핑 데이터 매핑 기술입니다. 즉, 배열에 기록되는 데이터는 스트라이프로 분류되고 배열의 멤버 디스크에 기록되어 낮은 고유 비용으로 높은 I/O 성능을 허용하지만 중복은 제공하지 않습니다.

RAID 레벨 0 구현은 배열에서 가장 작은 장치의 크기까지 멤버 장치에서만 데이터를 스트라이핑합니다. 즉, 크기가 약간 다른 여러 장치가 있는 경우 각 장치는 가장 작은 드라이브와 동일한 크기인 것처럼 처리됩니다. 따라서 수준 0 배열의 일반적인 스토리지 용량이 모든 디스크의 총 용량입니다. 멤버 디스크에 다른 크기가 있는 경우 RAID0은 사용 가능한 영역을 사용하여 해당 디스크의 모든 공간을 사용합니다.

수준 1

RAID 수준 1 또는 미러링은 배열의 각 멤버 디스크에 동일한 데이터를 작성하여 각 디스크에 미러링된 복사본을 남겨 두어 중복성을 제공합니다. 미러링은 단순성과 높은 수준의 데이터 가용성으로 인해 널리 사용됩니다. 수준 1은 두 개 이상의 디스크로 작동하며 매우 우수한 데이터 신뢰성을 제공하고 읽기 집약적인 애플리케이션의 성능을 개선하지만 상대적으로 높은 비용으로 작동합니다.

RAID 수준 1은 데이터 신뢰성을 제공하는 어레이의 모든 디스크에 동일한 정보를 쓰므로 비용이 많이 발생하지만 레벨 5와 같은 패리티 기반 RAID 수준보다 훨씬 공간 효율적입니다. 그러나 이 공간의 비효율성은 성능 이점을 제공하며, 이는 패리티를 생성하기 위해 더 많은 CPU 성능을 소비하는 패리티 기반 RAID 레벨인 반면 RAID 레벨 1은 CPU 오버헤드가 거의 없는 여러 RAID 멤버에 한 번 이상 동일한 데이터를 씁니다. 따라서 RAID 레벨 1은 소프트웨어 RAID가 채용되고 시스템의 CPU 리소스가 RAID 활동 이외의 작업과 함께 지속적으로 과세되는 머신에서 패리티 기반 RAID 수준을 줄일 수 있습니다.

수준 1 배열의 스토리지 용량은 하드웨어 RAID에서 가장 작은 미러링된 하드 디스크 용량 또는 소프트웨어 RAID에서 가장 작은 미러링된 파티션의 용량과 동일합니다. 레벨 1 중복성은 모든 RAID 유형 중에서 가능한 가장 높으며, 배열은 단일 디스크에서만 작동할 수 있습니다.

수준 4

레벨 4는 데이터를 보호하기 위해 단일 디스크 드라이브에 집중된 패리티를 사용합니다. 패리티 정보는 배열의 나머지 멤버 디스크의 내용에 따라 계산됩니다. 그러면 배열의 디스크가 실패할 때 이

정보를 사용하여 데이터를 재구성할 수 있습니다. 그런 다음 재구축된 데이터를 교체하기 전에 실패한 디스크에 대한 I/O 요청을 충족하고 교체된 후 실패한 디스크를 다시 리포지토리하는 데 사용할 수 있습니다.

전용 패리티 디스크는 RAID 배열에 대한 모든 쓰기 트랜잭션의 고유 성능 장애를 나타내기 때문에 레벨 4는 나중 쓰기 캐싱과 같은 기술을 함께 사용하지 않고 거의 사용되지 않습니다. 또는 시스템 관리자가 데이터가 채워지면 쓰기 트랜잭션이 거의 없는 배열과 같이 이러한 병목 현상이 있는 소프트웨어 RAID 장치를 의도적으로 설계하는 특정 상황에서 사용됩니다. RAID 수준 4는 너무 드물게 사용되므로 Anaconda에서 옵션으로 사용할 수 없습니다. 그러나 필요한 경우 사용자가 수동으로 생성할 수 있습니다.

하드웨어 RAID 수준 4의 스토리지 용량은 가장 작은 멤버 파티션의 용량과 같으며 파티션 수와 1을 곱한 값입니다. RAID 수준 4 배열의 성능은 항상 DestinationRule이며, 이는 outperform 쓰기를 의미합니다. 이는 쓰기 작업에서 패리티를 생성할 때 추가 CPU 리소스와 주요 메모리 대역폭을 소비하기 때문에 데이터를 작성할 뿐만 아니라 패리티를 작성할 때 실제 데이터를 디스크에 쓸 때 추가 버스 대역폭을 사용하기 때문입니다. 읽기 작업은 배열이 성능 저하된 상태에 있지 않는 한 패리티가 아닌 데이터를 읽기만 하면 됩니다.**Read operations need only read the data and not the parity unless the array is in a degraded state.** 결과적으로 읽기 작업은 정상적인 작동 상태에서 동일한 양의 데이터 전송을 위해 드라이브 및 컴퓨터 전체에 대한 트래픽을 줄일 수 있습니다.

수준 5

RAID의 가장 일반적인 유형입니다. RAID 수준 5는 배열의 모든 멤버 디스크 드라이브에 패리티를 배포함으로써 레벨 4에 포함된 쓰기 병목 현상을 제거합니다. 유일한 성능 장애는 패리티 계산 프로세스 자체입니다. 최신 CPU는 패리티를 매우 빠르게 계산할 수 있습니다. 그러나 RAID 5 어레이에 많은 디스크가 있으므로 모든 장치에서 집계 데이터 전송 속도가 충분히 높으면 패리티 계산이 병목 현상이 발생할 수 있습니다.

레벨 5는 성능이 뛰어나고 실제로 성능이 저하되는 쓰기를 읽습니다. RAID 레벨 5의 스토리지 용량은 레벨 4와 동일한 방식으로 계산됩니다.

수준 6

이는 데이터 중복성과 보존이 아닌 경우 일반적인 RAID 수준입니다. 그러나 수준 1의 공간 비효율성이 허용되지 않는 가장 중요한 문제입니다. 레벨 6은 복잡한 패리티 체계를 사용하여 배열의 두 드라이브의 손실에서 복구할 수 있습니다. 이러한 복잡한 패리티 체계는 소프트웨어 RAID 장치에 상당한 CPU 부담을 발생시키고 쓰기 트랜잭션 중에 부담을 증가시킵니다. 따라서 레벨 6은 레벨 4 및 5보다 성능이 크게 향상됩니다.

RAID 레벨 6 배열의 총 용량은 추가 패리티 스토리지 공간에 대해 장치 수에서 1개 대신 두 개 대신 두 개의 장치를 뺀 경우를 제외하고 RAID 수준 5 및 4와 유사하게 계산됩니다.

수준 10

이 RAID 수준은 수준 0의 성능 이점과 수준 1의 중복성을 결합하려고 시도합니다. 또한 두 개 이상의 장치가 있는 수준 1 배열에서 부족한 공간 중 일부를 줄입니다. 예를 들어 레벨 10을 사용하면 각 데

이터의 두 사본만 저장하도록 구성된 3-드 드라이브 어레이를 생성할 수 있으며, 이를 통해 전체 배열 크기가 3 장치 수준 1 어레이와 유사한 소형 장치보다 1.5배나 작은 장치의 크기를 1.5배로 설정할 수 있습니다. 이렇게 하면 RAID 수준 6과 유사한 패리티를 계산하기 위해 CPU 프로세스 사용량이 발생하지 않지만 공간 효율적입니다.

RAID 수준 10 생성은 설치 중에 지원되지 않습니다. 설치 후 수동으로 만들 수 있습니다.

선형 RAID

선형 RAID는 더 큰 가상 드라이브를 생성하는 드라이브 그룹입니다.

선형 RAID에서 청크는 하나의 멤버 드라이브에서 순차적으로 할당되며 첫 번째 드라이브가 완전히 채워지는 경우에만 다음 드라이브로 이동합니다. 이 그룹화는 멤버 드라이브 간에 I/O 작업 분할이 발생할 가능성은 드물기 때문에 성능상의 이점을 제공하지 않습니다. Linear RAID는 또한 중복성을 제공하지 않으며 신뢰성을 저하시킵니다. 하나의 멤버 드라이브가 실패하면 전체 배열을 사용할 수 없으며 데이터가 손실될 수 있습니다. 용량은 모든 멤버 디스크의 합계입니다.

9.3. LVM RAID 세그먼트 유형

RAID 논리 볼륨을 생성하려면 lvcreate 명령의 --type 인수를 사용하여 RAID 유형을 지정할 수 있습니다. 대부분의 사용자에 대해 raid1,raid4,raid5,raid6 및 raid10 인 5가지 기본 유형 중 하나를 지정하면 충분합니다.

다음 표에서는 가능한 RAID 세그먼트 유형을 설명합니다.

표 9.1. LVM RAID 세그먼트 유형

세그먼트 유형	설명
raid1	RAID1 미러링. 스트라이핑을 지정하지 않고 **-m** 인수를 지정할 때 **lvcreate** 명령의 **--type** 인수의 기본값입니다.
raid4	RAID4 전용 패리티 디스크.
raid5_la	• RAID5를 남겨 둡니다. • 데이터 연속을 사용하여 패리티 0 회전.
raid5_ra	• RAID5 오른쪽metric. • 데이터 연속으로 회전 패리티 N.

세그먼트 유형	설명
raid5_ls	• RAID5에서 대칭을 남겨 둡니다. • 이것은 **raid5** 와 동일합니다. • 데이터를 다시 시작하여 패리티 0을 순환합니다.
raid5_rs	• RAID5 올바른 대칭입니다. • 데이터를 다시 시작하여 패리티 N을 회전합니다.
raid6_zr	• RAID6 다시 시작 • 이것은 **raid6** 과 동일합니다. • 데이터를 다시 시작하여 제로(left-to-right) 회전.
raid6_nr	• RAID6 N 재시작. • 데이터를 다시 시작하여 분할 패리티 N(left-to-right)입니다.
raid6_nc	• RAID6 N이 계속됩니다. • 데이터 연속으로 회전 패리티 N (left-to-right)
raid10	• 스트라이핑된 미러입니다. 1보다 큰 스트라이프 수와 함께 **-m** 인수를 지정하는 경우 **lvcreate** 명령의 **--type** 인수의 기본값입니다. • 미러 세트의 제거
raid0/raid0_meta	스트라이핑. RAID0은 스트라이프 크기 단위로 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 분산합니다. 이는 성능을 향상시키는 데 사용됩니다. 하위 볼륨 중 하나라도 실패하면 논리 볼륨 데이터가 손실됩니다.

9.4. RAID 논리 볼륨 생성

***-m** 인수에 대해 지정한 값에 따라 여러 복사본을 사용하여 **RAID1** 배열을 생성할 수 있습니다. 마찬가지로 **-i** 인수를 사용하여 **RAID 0, 4, 5, 6, 10**개의 스트라이프 수를 지정할 수 있습니다. **-I** 인수를 사용하여 스트라이프 크기를 지정할 수도 있습니다. 다음 절차에서는 다양한 유형의 **RAID** 논리 볼륨을 생성하는 다양한 방법을 설명합니다.*

절차

- *2방향 RAID를 생성합니다. 다음 명령은 볼륨 그룹 my_vg, 즉 크기가 1G 인 my_lv 라는 2방향 RAID1 어레이를 생성합니다.*

```
# lvcreate --type raid1 -m 1 -L 1G -n my_lv my_vg
Logical volume "my_lv" created.
```

- *스트라이프를 사용하여 RAID5 어레이를 생성합니다. 다음 명령은 볼륨 그룹 my_vg, 크기가 1G 인 세 개의 스트라이프와 my_lv 라는 하나의 암시적 패리티 드라이브와 함께 RAID5 배열을 생성합니다. LVM 스트라이프 볼륨과 유사한 스트라이프 수를 지정할 수 있습니다. 올바른 패리티 드라이브 수가 자동으로 추가됩니다.*

```
# lvcreate --type raid5 -i 3 -L 1G -n my_lv my_vg
```

- *스트라이프를 사용하여 RAID6 어레이를 생성합니다. 다음 명령은 볼륨 그룹 my_ vg 에서 3 개의 스트라이프와 두 개의 암시적 패리티 드라이브(예: 1G 1 gigabyte)인 RAID6 어레이를 생성합니다.*

```
# lvcreate --type raid6 -i 3 -L 1G -n my_lv my_vg
```

검증

- *2방향 RAID1 어레이인 LVM 장치 my_vg/my_lv 를 표시합니다.*

```
# lvs -a -o name,copy_percent,devices _my_vg_
  LV              Copy%  Devices
  my_lv             6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]        /dev/sde1(0)
  [my_lv_rimage_1]        /dev/sdf1(1)
  [my_lv_rmeta_0]         /dev/sde1(256)
  [my_lv_rmeta_1]         /dev/sdf1(0)
```

추가 리소스

- *lvcreate(8) 및 lvmraidECDHE 도움말 페이지*

9.5. RAID0 스트라이핑 논리 볼륨 생성

RAID0 논리 볼륨은 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 스트라이프 크기 단위로 분산합니다. 다음 절차에서는 디스크에서 데이터를 스트라이핑하는 mylv 라는 LVM RAID0 논리 볼륨을 생성합니다.

사전 요구 사항

1. *세 개 이상의 물리 볼륨을 생성했습니다. 물리 볼륨 생성에 대한 자세한 내용은 **LVM 물리 볼륨 생성** 을 참조하십시오.*

2. *볼륨 그룹을 생성했습니다. 자세한 내용은 **LVM 볼륨 그룹 생성** 을 참조하십시오.*

절차

1. *기존 볼륨 그룹에서 **RAID0** 논리 볼륨을 생성합니다. 다음 명령은 세 개의 스트라이프 크기와 스트라이프 크기가 **4kB** 인 볼륨 그룹 **myvg** 에서 **RAID0** 볼륨 **mylv** 를 생성합니다.*

```
# lvcreate --type raid0 -L 2G --stripes 3 --stripesize 4 -n mylv my_vg
  Rounding size 2.00 GiB (512 extents) up to stripe boundary size 2.00 GiB(513 extents).
  Logical volume "mylv" created.
```

2. *RAID0 논리 볼륨에 파일 시스템을 생성합니다. 다음 명령은 논리 볼륨에 **ext4** 파일 시스템을 생성합니다.*

```
# mkfs.ext4 /dev/my_vg/mylv
```

3. *논리 볼륨을 마운트하고 파일 시스템 디스크 공간 사용량을 보고합니다.*

```
# mount /dev/my_vg/mylv /mnt

# df
Filesystem            1K-blocks     Used  Available  Use% Mounted on
/dev/mapper/my_vg-mylv   2002684     6168  1875072    1%   /mnt
```

검증

- *생성된 **RAID0** 제거 논리 볼륨을 확인합니다.*

```
# lvs -a -o +devices,segtype my_vg
  LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert Devices Type
  mylv my_vg rwi-a-r--- 2.00g mylv_rimage_0(0),mylv_rimage_1(0),mylv_rimage_2(0) raid0
  [mylv_rimage_0] my_vg iwi-aor--- 684.00m /dev/sdf1(0) linear
  [mylv_rimage_1] my_vg iwi-aor--- 684.00m /dev/sdg1(0) linear
  [mylv_rimage_2] my_vg iwi-aor--- 684.00m /dev/sdh1(0) linear
```

9.6. 스토리지 RHEL 시스템 역할을 사용하여 RAID LVM 볼륨의 스트라이프 크기 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform을 사용하여 RHEL에서 RAID LVM 볼륨의 스트라이프 크기를 구성할 수 있습니다. 사용 가능한 매개 변수로 Ansible 플레이북을 설정하여 RAID로 LVM 풀을 구성할 수 있습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 sudo 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: ~/playbook.yml)을 생성합니다.

```
---
- name: Configure stripe size for RAID LVM volumes
  hosts: managed-node-01.example.com
  roles:
    - rhel-system-roles.storage
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        volumes:
          - name: my_volume
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            raid_level: raid1
            raid_stripe_size: "256 KiB"
            state: present
```

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. *플레이북을 실행합니다.*

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- */usr/share/ansible/roles/rhel-system-roles.storage/README.md file*
- */usr/share/doc/rhel-system-roles/storage/ 디렉터리*
- *RAID 관리*

9.7. RAID0을 생성하는 매개변수

lvcreate --type raid0[meta] --stripes_stripes --stripes --stripes --stripes VolumeGroup [PhysicalVolumePath] 명령을 사용하여 RAID0 스트리핑 논리 볼륨을 생성할 수 있습니다.

다음 표에서는 RAID0 스트라이프 논리 볼륨을 생성하는 동안 사용할 수 있는 다양한 매개변수를 설명합니다.

표 9.2. RAID0 스트라이핑 논리 볼륨을 생성하는 매개변수

매개변수	설명
--type raid0[_meta]	**raid0** 을 지정하면 메타데이터 볼륨 없이 RAID0 볼륨이 생성됩니다. **raid0_meta** 를 지정하면 메타데이터 볼륨이 포함된 RAID0 볼륨이 생성됩니다. RAID0은 복제되지 않으므로 미러링된 데이터 블록을 RAID1/10으로 저장하거나 패리티 블록을 RAID4/5/6으로 계산 및 저장하지 않습니다. 따라서 미러링되거나 패리티 블록의 동기화 진행 상태를 유지하기 위해 메타데이터 볼륨이 필요하지 않습니다. RAID0에서 RAID4/5/6/10으로 변환하는 경우 메타데이터 볼륨이 필수입니다. **raid0_meta** 를 지정하면 해당 메타데이터 볼륨이 사전 할당 실패가 발생하지 않습니다.
--strip *es*	논리 볼륨을 분배할 장치 수를 지정합니다.
--stripesize *StripeSize*	각 스트라이프의 크기를 킬로바이트로 지정합니다. 이는 다음 장치로 이동하기 전에 한 장치에 기록된 데이터의 양입니다.

매개변수	설명
VolumeGroup	사용할 볼륨 그룹을 지정합니다.
PhysicalVolumePath	사용할 장치를 지정합니다. 이 값을 지정하지 않으면 LVM에서 *Stripes* 옵션에 지정된 장치 수를 선택합니다.

9.8. 소프트 데이터 손상

데이터 스토리지의 소프트 손상은 스토리지 장치에서 검색한 데이터가 해당 장치에 기록된 데이터와 다르다는 것을 의미합니다. 손상된 데이터는 저장 장치에 무한정 존재할 수 있습니다. 이 데이터를 검색하고 사용할 때까지 이러한 손상된 데이터를 검색하지 못할 수 있습니다.

구성 유형에 따라 장치 장애가 발생할 때 RID(Red 배터리 디스크) 논리 볼륨(LV)이 손실되지 않습니다. RAID 어레이로 구성된 장치가 실패하면 해당 RAID LV의 일부인 다른 장치에서 데이터를 복구할 수 있습니다. 그러나 RAID 구성은 데이터 자체의 무결성을 보장하지 않습니다. 소프트 손상, 자동 손상, 소프트 오류 및 자동 오류는 시스템 설계 및 소프트웨어가 예상대로 작동하는 경우에도 손상된 데이터를 설명하는 용어입니다.

DM(Device mapper) 무결성은 RAID 수준 1, 4, 5, 6, 10과 함께 사용되어 소프트 손상으로 인한 데이터 손실을 완화하거나 방지합니다. RAID 계층을 사용하면 데이터의 손상되지 않은 사본이 소프트 손상 오류를 수정할 수 있습니다. 무결성 계층은 각 RAID 이미지 위에 위치하며 추가 하위 LV는 각 RAID 이미지에 대한 무결성 메타데이터 또는 데이터 체크섬을 저장합니다. 무결성이 있는 RAID LV에서 데이터를 검색하는 경우 무결성 데이터 체크섬은 손상에 대한 데이터를 분석합니다. 손상이 감지되면 무결성 계층에서 오류 메시지를 반환하고 RAID 계층에서 다른 RAID 이미지에서 데이터의 손상되지 않은 사본을 검색합니다. RAID 계층은 손상된 데이터에 대해 변경되지 않은 데이터를 자동으로 다시 작성하여 소프트 손상을 복구합니다.

DM 무결성을 사용하여 새 RAID LV를 생성하거나 기존 RAID LV에 무결성을 추가하는 경우 다음 사항을 고려하십시오.

- *무결성 메타데이터에는 추가 스토리지 공간이 필요합니다. 각 RAID 이미지에 대해 500MB의 데이터에는 데이터에 추가되는 체크섬으로 인해 4MB의 추가 스토리지 공간이 필요합니다.*
- *일부 RAID 구성은 다른 것보다 더 영향을 미치지만 DM 무결성을 추가하면 데이터에 액세스할 때 대기 시간으로 인해 성능에 영향을 미칩니다. RAID1 구성은 일반적으로 RAID5 또는 해당 변형보다 더 나은 성능을 제공합니다.*
- *RAID 무결성 블록 크기는 성능에도 영향을 미칩니다. 더 큰 RAID 무결성 블록 크기를 구성하면 성능이 향상됩니다. 그러나 더 작은 RAID 무결성 블록 크기는 더 큰 이전 버전과의 호환성을*

제공합니다.

- 사용 가능한 무결성 모드는 두 가지, 즉, **journal** 또는 **journal** 입니다. 일반적으로 무결성 모드는 저널 모드보다 더 나은 성능을 제공합니다.

작은 정보

성능 문제가 발생하면 **RAID1**을 무결성으로 사용하거나 특정 **RAID** 구성의 성능을 테스트하여 요구 사항을 충족하는지 확인합니다.

9.9. DM 무결성으로 RAID LV 생성

DM(Device mapper) 무결성을 사용하여 **RAID LV**를 생성하거나 기존 **RAID LV**에 무결성을 추가하면 소프트 손상으로 인해 데이터가 손실될 위험이 완화됩니다. **LV**를 사용하기 전에 무결성 동기화 및 **RAID** 메타데이터가 완료될 때까지 기다립니다. 그러지 않으면 백그라운드 초기화가 **LV**의 성능에 영향을 미칠 수 있습니다.

절차

1. **DM** 무결성을 사용하여 **RAID LV**를 생성합니다. 다음 예제에서는 사용 가능한 크기가 **256M** 및 **RAID** 수준 **1** 인 **test-lv** 볼륨 그룹에 **test-lv**라는 이름의 새 **RAID LV**를 생성합니다.

```
# lvcreate --type raid1 --raidintegrity y -L 256M -n test-lv my_vg
Creating integrity metadata LV test-lv_rimage_0_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_0_imeta" created.
Creating integrity metadata LV test-lv_rimage_1_imeta with size 8.00 MiB.
Logical volume "test-lv_rimage_1_imeta" created.
Logical volume "test-lv" created.
```

참고

기존 **RAID LV**에 **DM** 무결성을 추가하려면 다음 명령을 사용합니다.

```
# lvconvert --raidintegrity y my_vg/test-lv
```

RAID LV에 무결성을 추가하면 해당 **RAID LV**에서 수행할 수 있는 작업 수가 제한됩니다.

2. 선택 사항: 특정 작업을 수행하기 전에 무결성을 제거합니다.

```
# lvconvert --raidintegrity n my_vg/test-lv
Logical volume my_vg/test-lv has removed integrity.
```

검증

- 추가된 DM 무결성에 대한 정보를 봅니다.
  - my_vg 볼륨 그룹에 생성된 test-lv RAID LV에 대한 정보를 확인합니다.

```
# lvs -a my_vg
  LV                  VG      Attr       LSize   Origin              Cpy%Sync
  test-lv             my_vg rwi-a-r--- 256.00m                      2.10
  [test-lv_rimage_0]      my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 93.75
  [test-lv_rimage_0_imeta]  my_vg ewi-ao----   8.00m
  [test-lv_rimage_0_iorig]  my_vg -wi-ao---- 256.00m
  [test-lv_rimage_1]      my_vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 85.94
  [...]
```

다음은 이 출력과 다른 옵션을 설명합니다.

G 특성

Attr 열의 속성 목록에는 RAID 이미지가 무결성을 사용하고 있음을 나타냅니다. 무결성은 체크섬을 _imeta RAID LV에 저장합니다.

CPY%Sync 열

최상위 RAID LV와 각 RAID 이미지에 대한 동기화 진행 상황을 나타냅니다.

RAID 이미지

LV 열에 raid_image_N 으로 표시됩니다.

LV 열

동기화 진행 상황으로 최상위 RAID LV와 각 RAID 이미지에 대해 100%가 표시됩니다.

  - 각 RAID LV 유형을 표시합니다.

```
# lvs -a my-vg -o+segtype
  LV                  VG      Attr       LSize   Origin              Cpy%Sync Type
  test-lv             my_vg rwi-a-r--- 256.00m                      87.96    raid1
  [test-lv_rimage_0]      my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 100.00
```

```
integrity
 [test-lv_rimage_0_imeta] my_vg ewi-ao----   8.00m                                  linear
 [test-lv_rimage_0_iorig] my_vg -wi-ao---- 256.00m                                  linear
 [test-lv_rimage_1]       my_vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 100.00
integrity
 [...]
```

- 각 RAID 이미지에서 탐지된 불일치 수를 계산하는 증분 카운터가 있습니다. my_vg/test-lv 에서 rimage_0 에서 무결성에 의해 감지된 데이터 불일치를 확인합니다.

```
# lvs -o+integritymismatches my_vg/test-lv_rimage_0
 LV              VG     Attr       LSize   Origin                 Cpy%Sync IntegMismatches
 [test-lv_rimage_0] my_vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig]    100.00
0
```

이 예에서 무결성은 데이터 불일치를 감지하지 못하므로 IntegMismatches 카운터는 0 (0)을 표시합니다.

- 다음 예와 같이 /var/log/ECDHE 로그 파일에서 데이터 무결성 정보를 확인합니다.

예 9.1. 커널 메시지 로그에서 dm-integrity 불일치의 예

```
device-mapper: integrity: dm-12: 체크섬 0x24e7 섹터에서 실패했습니다.
```

예 9.2. 커널 메시지 로그의 dm-integrity 데이터 수정 예

```
MD/raid1:mdX: 읽기 오류가 수정되었습니다 (dm-16)의 9448에서 수정
```

추가 리소스

- lvcreate(8) 및 lvmraidECDHE 도움말 페이지

9.10. 최소 및 최대 I/O 속도 옵션

RAID 논리 볼륨을 생성할 때 동기화 작업으로 논리 볼륨을 초기화하는 데 필요한 백그라운드 I/O 작업은 볼륨 그룹 메타데이터 업데이트 등 LVM 장치에 다른 I/O 작업을 사용할 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다.

복구 제한을 구현하여 **RAID** 논리 볼륨이 초기화되는 속도를 제어할 수 있습니다. 동기화 작업이 수행되는 속도를 제어하려면 **lvcreate** 명령의 **--minrecoveryrate** 및**-maxrecoveryrate** 옵션을 사용하여 해당 작업에 대한 최소 및 최대 **I/O** 속도를 설정합니다.

다음 옵션을 다음과 같이 지정할 수 있습니다.

--maxrecoveryrate Rate[bBsSkKmMgG]

nominal I/O 작업을 사용하지 않도록 **RAID** 논리 볼륨의 최대 복구 속도를 설정합니다. 백분율을 배열의 각 장치에 대한 초당 양으로 지정합니다. 접미사를 지정하지 않으면 **kiB/sec/device**로 가정합니다. 복구 비율을 **0**으로 설정하면 바인딩 해제됩니다.

--minrecoveryrate Rate[bBsSkKmMgG]

nominal I/O가 많은 경우에도 동기화 작업에 필요한 **I/O**가 최소 처리량을 실현하도록 **RAID** 논리 볼륨의 최소 복구 속도를 설정합니다. 백분율을 배열의 각 장치에 대한 초당 양으로 지정합니다. 접미사를 지정하지 않으면 **kiB/sec/device**로 가정합니다.

예를 들어 **lvcreate --type raid10 -i 2 -m 1 -L 10G --maxrecoveryrate 128 -n my_lv my_vg** 명령을 사용하여 볼륨 그룹에 있는 **2**방향 **RAID10** 어레이 **my_lv** 를 생성합니다. 크기가 **10G** 이고 최대 복구률인 **my_vg** 는 **128 kiB/sec/device**입니다. **RAID** 스크럽 작업에 대해 최소 및 최대 복구 속도를 지정할 수도 있습니다.

9.11. 선형 장치를 RAID 논리 볼륨으로 변환

기존 선형 논리 볼륨을 **RAID** 논리 볼륨으로 변환할 수 있습니다. 이 작업을 수행하려면 **lvconvert** 명령의 **--type** 인수를 사용합니다.

RAID 논리 볼륨은 메타데이터 및 데이터 하위 볼륨으로 구성됩니다. 선형 장치를 **RAID1** 어레이로 변환할 때 새 메타데이터 하위 볼륨을 생성하여 선형 볼륨이 있는 동일한 물리 볼륨 중 하나에 원본 논리 볼륨을 연결합니다. 추가 이미지는 **metadata/data** 하위 볼륨 쌍에 추가됩니다. 원래 논리 볼륨과 쌍을 이루는 메타데이터 이미지가 동일한 물리 볼륨에 배치할 수 없는 경우 **lvconvert** 가 실패합니다.

절차

1. 변환해야 하는 논리 볼륨 장치를 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV     Copy%  Devices
  my_lv         /dev/sde1(0)
```

2.

선형 논리 볼륨을 RAID 장치로 변환합니다. 다음 명령은 볼륨 그룹 __my_vg에서 선형 논리 볼륨 my_lv 를 2 방향 RAID1 배열로 변환합니다.

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
  Are you sure you want to convert linear LV my_vg/my_lv to raid1 with 2 images
enhancing resilience? [y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

검증

- 논리 볼륨이 RAID 장치로 변환되는지 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(256)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

추가 리소스

- lvconvert(8) 매뉴얼 페이지

9.12. LVM RAID1 논리 볼륨을 LVM 선형 논리 볼륨으로 변환

기존 RAID1 LVM 논리 볼륨을 LVM 선형 논리 볼륨으로 변환할 수 있습니다. 이 작업을 수행하려면 lvconvert 명령을 사용하고 -m0 인수를 지정합니다. 이렇게 하면 RAID 어레이를 구성하는 모든 RAID 데이터 하위 볼륨과 RAID 어레이를 구성하는 모든 RAID 메타데이터 하위 볼륨이 제거되고 최상위 수준 RAID1 이미지를 선형 논리 볼륨으로 남겨 둡니다.

절차

1. 기존 LVM RAID1 논리 볼륨을 표시합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

2.

기존 RAID1 LVM 논리 볼륨을 LVM 선형 논리 볼륨으로 변환합니다. 다음 명령은 LVM RAID1 논리 볼륨 my_vg/my_lv 를 LVM 선형 장치로 변환합니다.

```
# lvconvert -m0 my_vg/my_lv
  Are you sure you want to convert raid1 LV my_vg/my_lv to type linear losing all resilience? [y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

LVM RAID1 논리 볼륨을 LVM 선형 볼륨으로 변환하는 경우 제거할 물리 볼륨도 지정할 수 있습니다. 다음 예제에서 lvconvert 명령은 /dev/sde1 을 제거하고 선형 장치를 구성하는 물리 볼륨으로 /dev/sdf1 을 유지하도록 지정합니다.

```
# lvconvert -m0 my_vg/my_lv /dev/sde1
```

검증

- RAID1 논리 볼륨이 LVM 선형 장치로 변환되었는지 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV    Copy%  Devices
  my_lv       /dev/sdf1(1)
```

추가 리소스

- lvconvert(8) 매뉴얼 페이지

9.13. 미러링된 LVM 장치를 RAID1 논리 볼륨으로 변환

세그먼트 유형 미러를 사용하여 기존 미러링된 LVM 장치를 RAID1 LVM 장치로 변환할 수 있습니다. 이 작업을 수행하려면 --type raid1 인수와 함께 lvconvert 명령을 사용합니다. 이렇게 하면 mimage 라는 미러 하위 볼륨이 rimage 라는 RAID 하위 볼륨으로 변경됩니다.

또한 미러 로그를 제거하고 해당 데이터 하위 볼륨과 동일한 물리 볼륨의 data 하위 볼륨에 대해 rmeta 라는 메타데이터 하위 볼륨을 생성합니다.

절차

1. 미러링된 논리 볼륨 my_vg/my_lv:의 레이아웃을 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
```

```
LV              Copy%  Devices
my_lv           15.20 my_lv_mimage_0(0),my_lv_mimage_1(0)
[my_lv_mimage_0]       /dev/sde1(0)
[my_lv_mimage_1]       /dev/sdf1(0)
[my_lv_mlog]           /dev/sdd1(0)
```

2. 미러링된 논리 볼륨 **my_vg/my_lv** 를 **RAID1** 논리 볼륨으로 변환합니다.

```
# lvconvert --type raid1 my_vg/my_lv
Are you sure you want to convert mirror LV my_vg/my_lv to raid1 type? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

검증

- 미러링된 논리 볼륨이 **RAID1** 논리 볼륨으로 변환되는지 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
 LV              Copy%  Devices
 my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
 [my_lv_rimage_0]       /dev/sde1(0)
 [my_lv_rimage_1]       /dev/sdf1(0)
 [my_lv_rmeta_0]        /dev/sde1(125)
 [my_lv_rmeta_1]        /dev/sdf1(125)
```

추가 리소스

- **lvconvert(8)** 매뉴얼 페이지

9.14. RAID 논리 볼륨의 크기를 조정하는 명령

다음과 같은 방법으로 **RAID** 논리 볼륨의 크기를 조정할 수 있습니다.

- **lvresize** 또는 **lvextend** 명령을 사용하여 모든 유형의 **RAID** 논리 볼륨의 크기를 늘릴 수 있습니다. **RAID** 이미지 수는 변경되지 않습니다. 스트라이프 **RAID** 논리 볼륨의 경우 스트라이핑된 **RAID** 논리 볼륨을 생성할 때 동일한 스트라이프 반올림 제약 조건이 적용됩니다.

- **lvresize** 또는 **lvreduce** 명령을 사용하여 모든 유형의 **RAID** 논리 볼륨의 크기를 줄일 수 있습니다. **RAID** 이미지 수는 변경되지 않습니다. **lvextend** 명령과 마찬가지로 스트라이프 라운드링 제약 조건이 스트라이프 **RAID** 논리 볼륨을 생성할 때 적용됩니다.

- **lvconvert** 명령의 **--stripes N** 매개 변수를 사용하여 **RAID4, RAID5, RAID6, RAID10**과 같은

스트라이프 RAID 논리 볼륨에서 스트라이프 수를 변경할 수 있습니다. 이렇게 하면 추가 또는 제거된 스트라이프의 용량에 따라 RAID 논리 볼륨의 크기를 늘리거나 줄입니다. raid10 볼륨은 스트라이프만 추가할 수 있습니다. 이 기능은 RAID 재형태의 일부이며 이 기능을 사용하면 동일한 RAID 수준을 유지하면서 RAID 논리 볼륨의 속성을 변경할 수 있습니다.

9.15. 기존 RAID1 장치의 이미지 수 변경

LVM 미러링 구현에서 이미지 수를 변경할 수 있는 방식과 유사하게 기존 RAID1 어레이의 이미지 수를 변경할 수 있습니다.

lvconvert 명령을 사용하여 RAID1 논리 볼륨에 이미지를 추가하는 경우 다음 작업을 수행할 수 있습니다.

- 결과 장치의 총 이미지 수를 지정합니다.
- 장치에 추가할 이미지 수입니다.
- 선택적으로 새 메타데이터/데이터 이미지 쌍이 상주하는 물리 볼륨을 지정할 수 있습니다.

절차

1. 2방향 RAID1 어레이인 LVM 장치 my_vg/my_lv 를 표시합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           6.25   my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(256)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

rmeta 라는 메타데이터 하위 볼륨은 항상 데이터 subvolumes rimage 와 동일한 물리적 장치에 존재합니다. metadata/data 하위 볼륨 쌍은 --alloc 을 지정하지 않는 한 RAID 배열의 다른 metadata/data 하위 볼륨 쌍과 동일한 물리 볼륨에 생성되지 않습니다.

2. 2 방향 RAID1 논리 볼륨 my_vg/my_lv 를 3방향 RAID1 논리 볼륨으로 변환합니다.

```
# lvconvert -m 2 my_vg/my_lv
```

```
Are you sure you want to convert raid1 LV my_vg/my_lv to 3 images enhancing
resilience? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

다음은 기존 **RAID1** 장치에서 이미지 수를 변경하는 몇 가지 예입니다.

- **RAID**에 이미지를 추가하는 동안 사용할 물리 볼륨을 지정할 수도 있습니다. 다음 명령은 배열에 사용할 물리 볼륨 **/dev/sdd1** 을 지정하여 **2-way RAID1** 논리 볼륨 **my_vg/my_lv** 를 **3-way RAID1** 논리 볼륨으로 변환합니다.

```
# lvconvert -m 2 my_vg/my_lv /dev/sdd1
```

- **3**방향 **RAID1** 논리 볼륨을 **2**방향 **RAID1** 논리 볼륨으로 변환합니다.

```
# lvconvert -m1 my_vg/my_lv
Are you sure you want to convert raid1 LV my_vg/my_lv to 2 images reducing
resilience? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

- 제거할 이미지가 포함된 물리 볼륨 **/dev/sde1** 을 지정하여 **3**방향 **RAID1** 논리 볼륨을 **2**방향 **RAID1** 논리 볼륨으로 변환합니다.

```
# lvconvert -m1 my_vg/my_lv /dev/sde1
```

또한 이미지 및 관련 메타데이터의 하위 볼륨을 제거하면 숫자가 높은 이미지가 슬롯을 채우기 위해 아래로 전환됩니다. **lv_rimage_0, lv_rimage_1, lv_rimage_1**로 구성된 **3-way RAID1** 어레이에서 **lv_rimage _1** 을 제거하면 **lv_rimage_ 0** 및 **lv_rimage _1** 로 구성된 **RAID1** 배열이 생성됩니다. 하위 볼륨의 **lv_rimage_2** 는 이름이 되고 빈 슬롯을 초과하여 **lv_rimage_1** 이 됩니다.

검증

- 기존 **RAID1** 장치에서 이미지 수를 변경한 후 **RAID1** 장치를 표시합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
 LV Cpy%Sync Devices
 my_lv 100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0] /dev/sdd1(1)
 [my_lv_rimage_1] /dev/sde1(1)
 [my_lv_rimage_2] /dev/sdf1(1)
 [my_lv_rmeta_0] /dev/sdd1(0)
 [my_lv_rmeta_1] /dev/sde1(0)
 [my_lv_rmeta_2] /dev/sdf1(0)
```

추가 리소스

- *lvconvert(8)* 매뉴얼 페이지

9.16. RAID 이미지를 별도의 논리 볼륨으로 분할

RAID 논리 볼륨의 이미지를 분할하여 새 논리 볼륨을 구성할 수 있습니다. 기존 RAID1 논리 볼륨에서 RAID 이미지를 제거하거나 장치 중간에서 RAID 데이터 하위 볼륨 및 연결된 메타데이터 하위 볼륨을 제거하는 경우 번호가 많은 이미지가 슬롯을 채우기 위해 아래로 이동합니다. 따라서 RAID 배열을 구성하는 논리 볼륨의 인덱스 번호는 손상된 정수 시퀀스입니다.

참고

RAID1 배열이 아직 동기화되지 않은 경우 RAID 이미지를 분리할 수 없습니다.

절차

1. 2방향 RAID1 어레이인 LVM 장치 my_vg/my_lv 를 표시합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           12.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
```

2. RAID 이미지를 별도의 논리 볼륨으로 분할합니다.

- 다음 예제에서는 2-way RAID1 논리 볼륨인 my_lv 를 두 개의 선형 논리 볼륨 my_lv 및 new 로 분할합니다.

```
# lvconvert --splitmirror 1 -n new my_vg/my_lv
Are you sure you want to split raid1 LV my_vg/my_lv losing all resilience? [y/n]: y
```

- 다음 예제에서는 3방향 RAID1 논리 볼륨인 my_lv 를 2방향 RAID1 논리 볼륨, my_lv 및 선형 논리 볼륨 new 로 분할합니다.

```
# lvconvert --splitmirror 1 -n new my_vg/my_lv
```

-

검증

- RAID 논리 볼륨의 이미지를 분리한 후 논리 볼륨을 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
 LV      Copy%  Devices
 my_lv          /dev/sde1(1)
 new            /dev/sdf1(1)
```

추가 리소스

- lvconvert(8) 매뉴얼 페이지

9.17. RAID 이미지 분할 및 병합

lvconvert 명령의 --splitmirrors 인수와 함께 --trackchanges 인수를 사용하여 변경 사항을 추적하면서 읽기 전용으로 RAID1 배열의 이미지를 일시적으로 나눌 수 있습니다. 이 기능을 사용하면 이미지가 분할된 이후 변경된 배열의 해당 부분만 다시 동기화하면서 나중에 이미지를 배열에 병합할 수 있습니다.

RAID 이미지를 --trackchanges 인수로 분할하면 분할할 이미지를 지정할 수 있지만 분할되는 볼륨의 이름은 변경할 수 없습니다. 또한 결과 볼륨에는 다음과 같은 제약 조건이 있습니다.

- 생성하는 새 볼륨은 읽기 전용입니다.
- 새 볼륨의 크기를 조정할 수 없습니다.
- 나머지 배열의 이름을 변경할 수 없습니다.
- 나머지 배열의 크기를 조정할 수 없습니다.
- 새 볼륨 및 나머지 배열을 독립적으로 활성화할 수 있습니다.

분할된 이미지를 병합할 수 있습니다. 이미지를 병합할 때 이미지가 분할된 이후 변경된 배열의 부분만 다시 동기화됩니다.

절차

1. *RAID 논리 볼륨을 생성합니다.*

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. *선택 사항: 생성된 RAID 논리 볼륨을 확인합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

3. *생성된 RAID 논리 볼륨에서 이미지를 분할하고 나머지 배열에 대한 변경 사항을 추적합니다.*

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
  my_lv_rimage_2 split from my_lv for read-only purposes.
  Use 'lvconvert --merge my_vg/my_lv_rimage_2' to merge back into my_lv
```

4. *선택 사항: 이미지를 분할한 후 논리 볼륨을 확인합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sdc1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

5. *볼륨을 배열에 다시 병합합니다.*

```
# lvconvert --merge my_vg/my_lv_rimage_1
  my_vg/my_lv_rimage_1 successfully merged back into my_vg/my_lv
```

검증

- 병합된 논리 볼륨을 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sdc1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdc1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

추가 리소스

- lvconvert(8) 매뉴얼 페이지

9.18. RAID 오류 정책 설정

/etc/lvm/lvm.conf 파일의 raid_fault_policy 필드 기본 설정에 따라 LVM RAID가 장치 오류를 자동으로 처리합니다. 요구 사항에 따라 raid_fault_policy 필드를 다음 매개변수 중 하나로 설정할 수 있습니다.

warn

이 매개변수는 실패한 장치를 수동으로 복구하고 시스템 로그를 사용하여 경고를 표시할 수 있습니다.

기본적으로 raid_fault_policy 필드의 값은 lvm.conf 에서 warn 입니다. 장치가 충분한 경우 RAID 논리 볼륨이 계속 작동합니다.

allocate

이 매개변수를 사용하여 실패한 장치를 자동으로 교체할 수 있습니다.

9.18.1. 할당할 RAID 오류 정책 설정

raid_fault_policy 필드를 /etc/lvm/lvm.conf 파일의 allocate 매개변수로 설정할 수 있습니다. 이 기본 설정을 사용하면 시스템에서 실패한 장치를 볼륨 그룹의 예비 장치로 교체하려고 합니다. 예비 장치가 없으면 시스템 로그에 이 정보가 포함됩니다.

절차

1. RAID 논리 볼륨을 확인합니다.

```
# lvs -a -o name,copy_percent,devices my_vg

 LV              Copy%  Devices
 my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0]       /dev/sdb1(1)
 [my_lv_rimage_1]       /dev/sdc1(1)
 [my_lv_rimage_2]       /dev/sdd1(1)
 [my_lv_rmeta_0]        /dev/sdb1(0)
 [my_lv_rmeta_1]        /dev/sdc1(0)
 [my_lv_rmeta_2]        /dev/sdd1(0)
```

2. ***/dev/sdb** 장치가 실패하면 **RAID** 논리 볼륨을 확인합니다.*

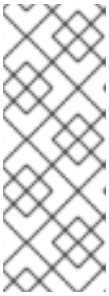
```
# lvs --all --options name,copy_percent,devices my_vg

 /dev/sdb: open failed: No such device or address
 Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
 WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and
assumed devices.
 WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and
assumed devices.
 LV              Copy%  Devices
 my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0]       [unknown](1)
 [my_lv_rimage_1]       /dev/sdc1(1)
 [...]
```

***/dev/sdb** 장치가 실패하는 경우 오류 메시지에 대한 시스템 로그를 볼 수도 있습니다.*

3. ***lvm.conf** 파일에 할당 할 **raid_fault_policy** 필드를 설정합니다.*

```
# vi /etc/lvm/lvm.conf
raid_fault_policy = "allocate"
```



참고

***raid_fault_policy** 를 할당 하도록 설정했지만 예비 장치가 없는 경우 할당이 실패하고 논리 볼륨을 그대로 둡니다. 할당에 실패하면 **lvconvert --repair** 명령을 사용하여 실패한 장치를 수정하고 교체할 수 있습니다. 자세한 내용은 논리 볼륨에서 실패한 **RAID** 장치 교체를 참조하십시오.*

검증

- *실패한 장치가 볼륨 그룹의 새 장치로 교체되었는지 확인합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  Couldn't find device with uuid 3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
  LV            Copy%  Devices
  lv            100.00 lv_rimage_0(0),lv_rimage_1(0),lv_rimage_2(0)
  [lv_rimage_0]        /dev/sdh1(1)
  [lv_rimage_1]        /dev/sdc1(1)
  [lv_rimage_2]        /dev/sdd1(1)
  [lv_rmeta_0]         /dev/sdh1(0)
  [lv_rmeta_1]         /dev/sdc1(0)
  [lv_rmeta_2]         /dev/sdd1(0)
```

참고

*이제 실패한 장치가 교체되었지만 장치가 볼륨 그룹에서 아직 제거되지 않았기 때문에 **LVM**에서 실패한 장치를 찾을 수 없다는 표시가 계속 표시됩니다. **my_vg**명령을 실행하여 볼륨 그룹에서 실패한 장치를 제거할 수 있습니다.*

추가 리소스

- ***lvm.conf(5) man page***

9.18.2. 경고로 RAID 오류 정책 설정

***raid_fault_policy** 필드를 **lvm.conf** 파일의 **warn** 매개변수로 설정할 수 있습니다. 이 기본 설정을 사용하면 시스템에서 실패한 장치를 나타내는 경고를 시스템 로그에 추가합니다. 경고에 따라 추가 단계를 확인할 수 있습니다.*

절차

1. ***RAID** 논리 볼륨을 확인합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

2. ***lvm.conf** 파일에서 **warn**로 **raid_fault_policy** 필드를 설정합니다.*

```
# vi /etc/lvm/lvm.conf
 # This configuration option has an automatic default value.
 raid_fault_policy = "warn"
```

3. **/dev/sdb** 장치가 실패하는 경우 오류 메시지를 표시하도록 시스템 로그를 확인합니다.

```
# grep lvm /var/log/messages

Apr 14 18:48:59 virt-506 kernel: sd 25:0:0:0: rejecting I/O to offline device
Apr 14 18:48:59 virt-506 kernel: I/O error, dev sdb, sector 8200 op 0x1:(WRITE) flags
0x20800 phys_seg 0 prio class 2
[...]
Apr 14 18:48:59 virt-506 dmeventd[91060]: WARNING: VG my_vg is missing PV 9R2TVV-
bwfn-Bdyj-Gucu-1p4F-qJ2Q-82kCAF (last written to /dev/sdb).
Apr 14 18:48:59 virt-506 dmeventd[91060]: WARNING: Couldn't find device with uuid
9R2TVV-bwfn-Bdyj-Gucu-1p4F-qJ2Q-82kCAF.
Apr 14 18:48:59 virt-506 dmeventd[91060]: Use 'lvconvert --repair my_vg/ly_lv' to replace
failed device.
```

/dev/sdb 장치가 실패하면 시스템 로그에 오류 메시지가 표시됩니다. 그러나 이 경우 **LVM**은 이미지 중 하나를 교체하여 **RAID** 장치를 자동으로 복구하지 않습니다. 대신 장치가 실패한 경우 장치를 **lvconvert** 명령의 **--repair** 인수로 교체할 수 있습니다. 자세한 내용은 논리 볼륨에서 실패한 **RAID** 장치 교체를 참조하십시오.

추가 리소스

- **lvm.conf(5) man page**

9.19. 논리 볼륨에서 RAID 장치 교체

다음 시나리오에 따라 논리 볼륨에서 **RAID** 장치를 교체할 수 있습니다.

- 작동 중인 **RAID** 장치 교체.
- 논리 볼륨에서 실패한 **RAID** 장치 교체.

9.19.1. 작동 중인 RAID 장치 교체

lvconvert 명령의 **--replace** 인수를 사용하여 논리 볼륨에서 작동 중인 **RAID** 장치를 교체할 수 있습니다.

주의

***RAID** 장치 오류가 발생하는 경우 다음 명령이 작동하지 않습니다.*

사전 요구 사항

- ***RAID** 장치가 실패하지 않았습니다.*

절차

1. ***RAID1** 배열을 생성합니다.*

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg
  Logical volume "my_lv" created
```

2. *생성된 **RAID1** 배열을 검사합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv          100.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdb2(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdb2(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
```

3. *요구 사항에 따라 **RAID** 장치를 다음 방법으로 교체합니다.*

   a. *교체할 물리 볼륨을 지정하여 **RAID1** 장치를 교체합니다.*

```
# lvconvert --replace /dev/sdb2 my_vg/my_lv
```

   b. *교체에 사용할 물리 볼륨을 지정하여 **RAID1** 장치를 교체합니다.*

```
# lvconvert --replace /dev/sdb1 my_vg/my_lv /dev/sdd1
```

c.

*여러 교체 인수를 지정하여 한 번에 여러 **RAID** 장치를 교체합니다.*

```
# lvconvert --replace /dev/sdb1 --replace /dev/sdc1 my_vg/my_lv
```

검증

1.

*교체할 물리 볼륨을 지정한 후 **RAID1** 배열을 검사합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           37.50 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sdb1(1)
  [my_lv_rimage_1]       /dev/sdc2(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sdb1(0)
  [my_lv_rmeta_1]        /dev/sdc2(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
```

2.

*교체에 사용할 물리 볼륨을 지정한 후 **RAID1** 배열을 검사합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           28.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
```

3.

*한 번에 여러 **RAID** 장치를 교체한 후 **RAID1** 배열을 검사합니다.*

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           60.00 my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdd1(1)
  [my_lv_rimage_2]       /dev/sde1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdd1(0)
  [my_lv_rmeta_2]        /dev/sde1(0)
```

추가 리소스

- ***lvconvert(8) man page***

9.19.2. 논리 볼륨에서 실패한 RAID 장치 교체

RAID는 기존 LVM 미러링과 동일하지 않습니다. LVM 미러링의 경우 실패한 장치를 제거합니다. 그렇지 않으면 RAID 배열이 실패한 장치에서 계속 실행되는 동안 미러링된 논리 볼륨이 중단됩니다. RAID1 이외의 RAID 수준의 경우 장치를 제거하면 RAID6에서 RAID5로 또는 RAID4 또는 RAID0으로의 낮은 RAID 수준으로의 변환을 의미합니다.

실패한 장치를 제거하고 교체를 LVM으로 할당하는 대신 lvconvert 명령의 --repair 인수를 사용하여 RAID 논리 볼륨에서 물리 볼륨으로 사용되는 실패한 장치를 교체할 수 있습니다.

사전 요구 사항

- 볼륨 그룹에는 실패한 장치를 교체할 수 있는 충분한 여유 용량을 제공하는 물리 볼륨이 포함되어 있습니다.

  볼륨 그룹에서 사용 가능한 확장 영역이 충분한 물리 볼륨이 없는 경우 Cryostatextend 유틸리티를 사용하여 충분히 큰 새 물리 볼륨을 추가합니다.

절차

1. RAID 논리 볼륨을 확인합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
  LV              Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdc1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdc1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

2. /dev/sdc 장치가 실패한 후 RAID 논리 볼륨을 확인합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vIzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking
used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking
used and assumed devices.
  LV              Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
```

```
[my_lv_rimage_1]        [unknown](1)
[my_lv_rimage_2]        /dev/sdd1(1)
[my_lv_rmeta_0]         /dev/sde1(0)
[my_lv_rmeta_1]         [unknown](0)
[my_lv_rmeta_2]         /dev/sdd1(0)
```

3. 실패한 장치를 교체합니다.

```
# lvconvert --repair my_vg/my_lv
 /dev/sdc: open failed: No such device or address
 Couldn't find device with uuid A4kRl2-vlzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
 WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking used and assumed devices.
 WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking used and assumed devices.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
Faulty devices in my_vg/my_lv successfully replaced.
```

4. 선택 사항: 실패한 장치를 대체하는 물리 볼륨을 수동으로 지정합니다.

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 교체를 사용하여 논리 볼륨을 검사합니다.

```
# lvs --all --options name,copy_percent,devices my_vg

 /dev/sdc: open failed: No such device or address
 /dev/sdc1: open failed: No such device or address
 Couldn't find device with uuid A4kRl2-vlzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
 LV              Cpy%Sync Devices
 my_lv             43.79  my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
 [my_lv_rimage_0]         /dev/sde1(1)
 [my_lv_rimage_1]         /dev/sdb1(1)
 [my_lv_rimage_2]         /dev/sdd1(1)
 [my_lv_rmeta_0]          /dev/sde1(0)
 [my_lv_rmeta_1]          /dev/sdb1(0)
 [my_lv_rmeta_2]          /dev/sdd1(0)
```

볼륨 그룹에서 실패한 장치를 제거할 때까지 **LVM** 유틸리티에서 오류가 발생한 장치를 찾을 수 없음을 계속 표시합니다.

6. 볼륨 그룹에서 실패한 장치를 제거합니다.

```
# vgreduce --removemissing my_vg
```

검증

1. 실패한 장치를 제거한 후 사용 가능한 물리 볼륨을 확인합니다.

```
# pvscan
PV /dev/sde1 VG rhel_virt-506 lvm2 [<7.00 GiB / 0 free]
PV /dev/sdb1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
```

2. 실패한 장치를 교체한 후 논리 볼륨을 검사합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]        /dev/sde1(1)
  [my_lv_rimage_1]        /dev/sdb1(1)
  [my_lv_rimage_2]        /dev/sdd1(1)
  [my_lv_rmeta_0]         /dev/sde1(0)
  [my_lv_rmeta_1]         /dev/sdb1(0)
  [my_lv_rmeta_2]         /dev/sdd1(0)
```

추가 리소스

- **lvconvert(8)** 및 **Cryo statreduce(8)** 도움말 페이지

9.20. RAID 논리 볼륨에서 데이터 일관성 확인

LVM은 **RAID** 논리 볼륨에 대한 스크럽 지원을 제공합니다. **RAID** 스크럽은 배열의 모든 데이터 및 패리티 블록을 읽고 일치 여부를 확인하는 프로세스입니다. **lvchange --syncaction repair** 명령은 배열에서 백그라운드 동기화 작업을 시작합니다. 다음 속성은 데이터 일관성에 대한 세부 정보를 제공합니다.

- **raid_sync_action** 필드에는 **RAID** 논리 볼륨이 수행하는 현재 동기화 작업이 표시됩니다. 다음 값 중 하나일 수 있습니다.

  idle

  모든 동기화 작업을 완료했습니다(없음).

  resync

  정리되지 않은 시스템 종료 후 배열을 초기화하거나 재동기화합니다.

  recover

배열에서 장치를 교체합니다.

Check

배열 불일치를 찾습니다.

복구

불일치를 찾고 복구합니다.

- **raid_mismatch_count** 필드에는 검사 작업 중에 발견된 불일치 수가 표시됩니다.
- **Cpy%Sync** 필드는 동기화 작업의 진행 상황을 표시합니다.
- **lv_attr** 필드는 추가 지표를 제공합니다. 이 필드의 비트 **9**는 논리 볼륨의 상태를 표시하고 다음 지표를 지원합니다.

m 또는 **mismatches**

RAID 논리 볼륨에 불일치가 있음을 나타냅니다. 스크러블링 작업이 **RAID**의 일부를 감지한 후 이 문자를 볼 수 있습니다. 이 문자는 일관성이 없습니다.

R 또는 **refresh**

LVM에서 장치 레이블을 읽고 장치가 작동하는 것으로 간주하더라도 **RAID** 배열에서 실패한 장치를 나타냅니다. 논리 볼륨을 새로 고쳐 장치를 사용할 수 있음을 커널에 알리거나 실패한 것으로 의심되는 경우 장치를 교체합니다.

절차

1. 선택 사항: 스크럽 프로세스에서 사용하는 **I/O** 대역폭을 제한합니다. **RAID** 스크러블링 작업을 수행할 때 동기화 작업에 필요한 백그라운드 **I/O**는 볼륨 그룹 메타데이터 업데이트 등 **LVM** 장치에 대한 다른 **I/O**의 충돌을 줄일 수 있습니다. 이로 인해 다른 **LVM** 작업이 느려질 수 있습니다.

복구 제한을 구현하여 스크럽 작업의 비율을 제어할 수 있습니다. **lvchange --syncaction** 명령과 함께 **--maxrecoveryrate Rate[bBsSkKmMgG]** 또는 **--minrecoveryrate Rate[bBsSkKmMgG]** 를 사용하여 복구 속도를 설정할 수 있습니다. 자세한 내용은 *최소 및 최대 I/O 속도 옵션을* 참조하십시오.

Rate 값을 배열의 각 장치에 대한 초당 양으로 지정합니다. 접미사를 제공하지 않으면 옵션은 장치당 초당 **kiB**를 가정합니다.

2. 복구하지 않고 배열의 불일치를 표시합니다.

```
# lvchange --syncaction check my_vg/my_lv
```

이 명령은 배열에서 백그라운드 동기화 작업을 시작합니다.

3. 선택 사항: 커널 메시지의 **var/log/syslog** 파일을 확인합니다.

4. 배열의 불일치를 수정하십시오.

```
# lvchange --syncaction repair my_vg/my_lv
```

이 명령은 **RAID** 논리 볼륨에서 실패한 장치를 복구하거나 교체합니다. 이 명령을 실행한 후 커널 메시지의 **var/log/syslog** 파일을 볼 수 있습니다.

검증

1. 스크럽 작업에 대한 정보를 표시합니다.

```
# lvs -o +raid_sync_action,raid_mismatch_count my_vg/my_lv
LV    VG    Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
SyncAction Mismatches
my_lv my_vg rwi-a-r--- 500.00m                    100.00           idle        0
```

추가 리소스

- **lvchange(8)** 및 **lvmraid(7)** 도움말 페이지
- 최소 및 최대 I/O 속도 옵션

9.21. RAID 논리 볼륨을 다른 RAID 수준으로 변환

LVM은 **RAID** 논리 볼륨을 하나의 **RAID** 수준에서 다른 **RAID** 수준으로 변환하는 것을 의미합니다(예: **RAID 5**에서 **RAID 6**로 변환). **RAID** 수준을 변경하여 장치 오류에 대한 복원력을 늘리거나 줄일 수 있습니

다.

절차

1. ***RAID** 논리 볼륨을 생성합니다.*

```
# lvcreate --type raid5 -i 3 -L 500M -n my_lv my_vg
Using default stripesize 64.00 KiB.
Rounding size 500.00 MiB (125 extents) up to stripe boundary size 504.00 MiB (126 extents).
Logical volume "my_lv" created.
```

2. ***RAID** 논리 볼륨을 확인합니다.*

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices                                                     Type
  my_lv           my_vg       rwi-a-r--- 504.00m                                    100.00           my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0) raid5
  [my_lv_rimage_0] my_vg      iwi-aor--- 168.00m                                                     /dev/sda(1)                                                    linear
```

3. ***RAID** 논리 볼륨을 다른 **RAID** 수준으로 변환합니다.*

```
# lvconvert --type raid6 my_vg/my_lv
Using default stripesize 64.00 KiB.
Replaced LV type raid6 (same as raid6_zr) with possible type raid6_ls_6.
Repeat this command to convert to raid6 after an interim conversion has finished.
Are you sure you want to convert raid5 LV my_vg/my_lv to raid6_ls_6 type? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

4. *선택 사항: 이 명령이 변환을 반복하도록 요청하는 메시지를 표시하는 경우 다음을 실행합니다.*

```
# lvconvert --type raid6 my_vg/my_lv
```

검증

1. *변환된 **RAID** 수준으로 **RAID** 논리 볼륨을 확인합니다.*

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices                                                     Type
```

```
  my_lv          my_vg        rwi-a-r--- 504.00m                                    100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0),my_lv_rimage_3(0),my_lv_ri
mage_4(0) raid6
  [my_lv_rimage_0] my_vg        iwi-aor--- 172.00m
/dev/sda(1)                                                    linear
```

추가 리소스

- lvconvert(8) 및 lvmraid(8) 도움말 페이지

9.22. RAID1 논리 볼륨에서 I/O 작업

lvchange 명령의 --writemost 및 --write behind 매개변수를 사용하여 RAID1 논리 볼륨에서 장치의 I/O 작업을 제어할 수 있습니다. 다음은 이러한 매개변수를 사용하는 형식입니다.

--[RAID]writemostly physicalVolume[:{t|y|n}]

RAID1 논리 볼륨의 장치를 대부분 쓰기 로 표시하고 필요한 경우가 아니면 이러한 드라이브에 대한 모든 읽기 작업을 방지합니다. 이 매개변수를 설정하면 I/O 작업 수를 최소로 유지합니다. lvchange --writemostly /dev/sdb my_vg/ly_lv 명령을 사용하여 이 매개변수를 설정합니다.

다음과 같은 방법으로 writemostly 속성을 설정할 수 있습니다.

:y

기본적으로 writemostly 속성 값은 논리 볼륨에서 지정된 물리 볼륨에 대해 yes입니다.

:n

writemostly 플래그를 제거하려면 :n 을 물리 볼륨에 추가합니다.

:t

writemostly 속성의 값을 토글하려면 -- writemostly 인수를 지정합니다. 단일 명령에서 이 인수를 두 번 이상 사용하여 논리 볼륨의 모든 물리 볼륨에 대한 쓰기 속성을 한 번에 전환할 수 있습니다.

--[RAID]writebehind IOCount

writemostly 로 표시된 보류 중인 쓰기의 최대 수를 지정합니다. RAID1 논리 볼륨의 장치에 적용할 수 있는 쓰기 작업 수입니다. 이 매개 변수의 값을 초과한 후 RAID 배열에서 모든 쓰기 작업이 완료되도록 하기 전에 구성 장치에 대한 모든 쓰기 작업이 동기적으로 완료됩니다.

lvchange --writebehind 100 my_vg/ly_lv 명령을 사용하여 이 매개변수를 설정할 수 있습니다.

writemostly 특성의 값을 **0**으로 설정하면 기본 설정이 지워집니다. 이 설정을 사용하면 시스템은 임의로 값을 선택합니다.

9.23. RAID 볼륨 교체

RAID 교체는 **RAID** 수준을 변경하지 않고 **RAID** 논리 볼륨의 속성을 변경하는 것을 의미합니다. 변경할 수 있는 일부 속성에는 **RAID** 레이아웃, 스트라이프 크기 및 스트라이프 수가 포함됩니다.

절차

1. **RAID** 논리 볼륨을 생성합니다.

```
# lvcreate --type raid5 -i 2 -L 500M -n my_lv my_vg

Using default stripesize 64.00 KiB.
Rounding size 500.00 MiB (125 extents) up to stripe boundary size 504.00 MiB (126 extents).
Logical volume "my_lv" created.
```

2. **RAID** 논리 볼륨을 확인합니다.

```
# lvs -a -o +devices

LV              VG    Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
my_lv           my_vg rwi-a-r--- 504.00m                                    100.00
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
[my_lv_rimage_0] my_vg iwi-aor--- 252.00m                                                 /dev/sda(1)
[my_lv_rimage_1] my_vg iwi-aor--- 252.00m                                                 /dev/sdb(1)
[my_lv_rimage_2] my_vg iwi-aor--- 252.00m                                                 /dev/sdc(1)
[my_lv_rmeta_0]  my_vg ewi-aor---   4.00m                                                 /dev/sda(0)
[my_lv_rmeta_1]  my_vg ewi-aor---   4.00m                                                 /dev/sdb(0)
[my_lv_rmeta_2]  my_vg ewi-aor---   4.00m                                                 /dev/sdc(0)
```

3. 선택 사항: **RAID** 논리 볼륨의 스트라이프 이미지 및 스트라이프 크기를 확인합니다.

```
# lvs -o stripes my_vg/my_lv
  #Str
    3
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  64.00k
```

4. *요구 사항에 따라 다음 방법을 사용하여 **RAID** 논리 볼륨의 속성을 수정합니다.*

a. ***RAID** 논리 볼륨의 스트라이프 이미지를 수정합니다.*

```
# lvconvert --stripes 3 my_vg/my_lv
Using default stripesize 64.00 KiB.
WARNING: Adding stripes to active logical volume my_vg/my_lv will grow it from 126 to
189 extents!
Run "lvresize -l126 my_vg/my_lv" to shrink it or use the additional capacity.
Are you sure you want to add 1 images to raid5 LV my_vg/my_lv? [y/n]: y
Logical volume my_vg/my_lv successfully converted.
```

b. ***RAID** 논리 볼륨의 스트라이프 크기를 수정합니다.*

```
# lvconvert --stripesize 128k my_vg/my_lv
  Converting stripesize 64.00 KiB of raid5 LV my_vg/my_lv to 128.00 KiB.
Are you sure you want to convert raid5 LV my_vg/my_lv? [y/n]: y
  Logical volume my_vg/my_lv successfully converted.
```

c. ***maxrecoveryrate** 및 **minrecoveryrate** 속성을 수정합니다.*

```
# lvchange --maxrecoveryrate 4M my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

```
# lvchange --minrecoveryrate 1M my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

d. ***syncaction** 속성을 수정합니다.*

```
# lvchange --syncaction check my_vg/my_lv
```

e. ***writemostly** 및 **writebehind** 속성을 수정합니다.*

```
# lvchange --writemostly /dev/sdb my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

```
# lvchange --writebehind 100 my_vg/my_lv
  Logical volume my_vg/my_lv changed.
```

검증

1. **RAID 논리 볼륨의 스트라이프 이미지 및 스트라이프 크기를 확인합니다.**

```
# lvs -o stripes my_vg/my_lv
  #Str
     4
```

```
# lvs -o stripesize my_vg/my_lv
  Stripe
  128.00k
```

2. **maxrecoveryrate 특성을 수정한 후 RAID 논리 볼륨을 확인합니다.**

```
# lvs -a -o +raid_max_recovery_rate
  LV              VG      Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert MaxSync
  my_lv           my_vg   rwi-a-r---  10.00g                                    100.00          4096
  [my_lv_rimage_0] my_vg   iwi-aor---  10.00g
[...]
```

3. **minrecoveryrate 특성을 수정한 후 RAID 논리 볼륨을 확인합니다.**

```
# lvs -a -o +raid_min_recovery_rate
  LV              VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert MinSync
  my_lv           my_vg  rwi-a-r---  10.00g                                    100.00          1024
  [my_lv_rimage_0] my_vg  iwi-aor---  10.00g
  [...]
```

4. **syncaction 속성을 수정한 후 RAID 논리 볼륨을 확인합니다.**

```
# lvs -a
  LV              VG      Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
  my_lv           my_vg   rwi-a-r---  10.00g                                    2.66
  [my_lv_rimage_0] my_vg   iwi-aor---  10.00g
  [...]
```

추가 리소스

- **lvconvert(8) 및 lvmraid(8) 도움말 페이지**

9.24. RAID 논리 볼륨에서 영역 크기 변경

RAID 논리 볼륨을 생성할 때 **/etc/lvm/lvm.conf** 파일의 **raid_region_size** 매개변수는 **RAID** 논리 볼륨의 리전 크기를 나타냅니다. **RAID** 논리 볼륨을 생성한 후 볼륨의 영역 크기를 변경할 수 있습니다. 이 매개 변수는 더티 상태 또는 정리 상태를 추적하는 세분성을 정의합니다. 비트맵의 더티 비트는 **RAID** 볼륨의 더티 종료 후 동기화할 작업 세트를 정의합니다(예: 시스템 오류).

raid_region_size 를 더 높은 값으로 설정하면 비트맵 크기와 혼잡이 줄어듭니다. 그러나 **RAID**에 대한 쓰기 는 영역을 동기화할 때까지 지연되기 때문에 영역을 재동기화하는 동안 쓰기 작업에 영향을 미칩니다.

절차

1. **RAID** 논리 볼륨을 생성합니다.

```
# lvcreate --type raid1 -m 1 -L 10G test
  Logical volume "lvol0" created.
```

2. **RAID** 논리 볼륨을 확인합니다.

```
# lvs -a -o +devices,region_size

LV              VG     Attr     LSize Pool Origin Data% Meta% Move Log   Cpy%Sync
Convert Devices                         Region
lvol0           test rwi-a-r--- 10.00g                             100.00
lvol0_rimage_0(0),lvol0_rimage_1(0)  2.00m
[lvol0_rimage_0]  test iwi-aor--- 10.00g                                    /dev/sde1(1)
0
[lvol0_rimage_1]  test iwi-aor--- 10.00g                                    /dev/sdf1(1)
0
[lvol0_rmeta_0]   test ewi-aor---  4.00m                                    /dev/sde1(0)
0
[lvol0_rmeta_1]   test ewi-aor---  4.00m
```

Region 열은 **raid_region_size** 매개변수의 값을 나타냅니다.

3. 선택 사항: **raid_region_size** 매개변수의 값을 확인합니다.

```
# cat /etc/lvm/lvm.conf | grep raid_region_size

# Configuration option activation/raid_region_size.
 # raid_region_size = 2048
```

4. **RAID 논리 볼륨의 영역 크기를 변경합니다.**

```
# lvconvert -R 4096K my_vg/my_lv

Do you really want to change the region_size 512.00 KiB of LV my_vg/my_lv to 4.00 MiB?
[y/n]: y
  Changed region size on RAID LV my_vg/my_lv to 4.00 MiB.
```

5. **RAID 논리 볼륨을 다시 동기화합니다.**

```
# lvchange --resync my_vg/my_lv

Do you really want to deactivate logical volume my_vg/my_lv to resync it? [y/n]: y
```

검증

1. **RAID 논리 볼륨을 확인합니다.**

```
# lvs -a -o +devices,region_size

LV              VG   Attr       LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices                          Region
lvol0           test rwi-a-r--- 10.00g                                    6.25
lvol0_rimage_0(0),lvol0_rimage_1(0)  4.00m
[lvol0_rimage_0] test iwi-aor--- 10.00g                                           /dev/sde1(1)
0
[lvol0_rimage_1] test iwi-aor--- 10.00g                                           /dev/sdf1(1)
0
[lvol0_rmeta_0]  test ewi-aor---  4.00m                                           /dev/sde1(0)
0
```

Region 열은 raid_region_size 매개변수의 변경된 값을 나타냅니다.

2. **lvm.conf 파일에서 raid_region_size 매개변수의 값을 확인합니다.**

```
# cat /etc/lvm/lvm.conf | grep raid_region_size

# Configuration option activation/raid_region_size.
 # raid_region_size = 4096
```

추가 리소스

- ***lvconvert(8) man page***

10장. 논리 볼륨의 스냅샷

LVM 스냅샷 기능을 사용하면 서비스 중단 없이 특정 즉시 볼륨(예: /dev/sda)의 가상 이미지를 생성할 수 있습니다.

10.1. 스냅샷 볼륨 개요

스냅샷을 만든 후 원래 볼륨(원본)을 수정할 때 스냅샷 기능은 변경 전의 수정된 데이터 영역 복사본을 만들어 볼륨 상태를 재구성할 수 있도록 합니다. 스냅샷을 만들 때 원본에 대한 전체 읽기 및 쓰기 액세스는 가능합니다.

스냅샷은 스냅샷을 생성한 후 변경되는 데이터 영역만 복사하므로 스냅샷 기능에 최소한의 스토리지 양이 필요합니다. 예를 들어 원본의 용량이 거의 업데이트된 원본의 경우 스냅샷을 유지하기에 충분한 용량의 3~5~5~4%이면 됩니다. 백업 프로시저를 대체하지 않습니다. 스냅샷 복사본은 가상 복사본이며 실제 미디어 백업이 아닙니다.

스냅샷의 크기는 원본 볼륨에 변경 사항을 저장하기 위해 별도로 설정된 공간을 제어합니다. 예를 들어 스냅샷을 만든 다음 원본을 완전히 덮어쓰는 경우 변경 사항을 유지하기 위한 원본 볼륨보다 최소한 큰 스냅샷이 있어야 합니다. 스냅샷 크기를 정기적으로 모니터링해야 합니다. 예를 들어 /usr 과 같은 읽기 볼륨의 수명이 짧은 스냅샷은 /home 과 같은 쓰기 횟수가 많기 때문에 볼륨의 수명이 짧은 스냅샷보다 적은 공간이 필요합니다.

스냅샷이 가득 차면 원본 볼륨에서 변경 사항을 더 이상 추적할 수 없기 때문에 스냅샷이 유효하지 않습니다. 그러나 스냅샷이 유효하지 않도록 사용량이 snapshot_autoextend_threshold 값을 초과할 때마다 스냅샷을 자동으로 확장하도록 LVM을 구성할 수 있습니다. 스냅샷은 완전히 조정 가능하며 다음 작업을 수행할 수 있습니다.

- 스토리지 용량이 있는 경우 스냅샷 볼륨의 크기를 늘려 삭제하지 않도록 할 수 있습니다.
- 스냅샷 볼륨이 필요한 것보다 큰 경우 볼륨 크기를 줄여 다른 논리 볼륨에 필요한 공간을 확보할 수 있습니다.

스냅샷 볼륨은 다음과 같은 이점을 제공합니다.

- 일반적으로 데이터를 지속적으로 업데이트하는 라이브 시스템을 중지하지 않고 논리 볼륨에서 백업을 수행해야 할 때 스냅샷을 만듭니다.

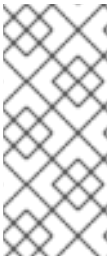
- 스냅샷 파일 시스템에서 **fsck** 명령을 실행하여 파일 시스템의 무결성을 확인하고 원래 파일 시스템에 파일 시스템 복구가 필요한지 확인할 수 있습니다.

- 스냅샷은 읽기/쓰기이므로 실제 데이터를 사용하지 않고 스냅샷에 대해 스냅샷을 작성하고 테스트를 실행하여 프로덕션 데이터에 대해 애플리케이션을 테스트할 수 있습니다.

- **Red Hat Virtualization**에서 사용할 **LVM** 볼륨을 생성할 수 있습니다. **LVM** 스냅샷을 사용하여 가상 게스트 이미지의 스냅샷을 생성할 수 있습니다. 이러한 스냅샷은 기존 게스트를 수정하거나 최소한의 추가 스토리지로 새 게스트를 생성하는 편리한 방법을 제공할 수 있습니다.

10.2. 원래 볼륨의 스냅샷 생성

lvcreate 명령을 사용하여 원래 볼륨(원본)의 스냅샷을 생성합니다. 볼륨 스냅샷에 쓸 수 있습니다. 기본적으로 스냅샷 볼륨은 씬 프로비저닝된 스냅샷과 비교하여 일반 활성화 명령 중에 원본으로 활성화됩니다. **LVM**에서는 원본 볼륨 크기 및 볼륨에 필요한 메타데이터 크기보다 큰 스냅샷 볼륨 생성을 지원하지 않습니다. 이보다 큰 스냅샷 볼륨을 지정하면 **LVM**에서 원본 크기에 필요한 스냅샷 볼륨을 생성합니다.



참고

클러스터의 노드는 **LVM** 스냅샷을 지원하지 않습니다. 공유 볼륨 그룹에서 스냅샷 볼륨을 생성할 수 없습니다. 그러나 공유 논리 볼륨에서 일관된 데이터 백업을 생성해야 하는 경우 볼륨을 독점적으로 활성화한 다음 스냅샷을 생성할 수 있습니다.

다음 절차에서는 **origin** 이라는 원본 논리 볼륨과 **snap** 이라는 이 원래 볼륨의 스냅샷 볼륨을 생성합니다.

사전 요구 사항

- 볼륨 그룹 **001**을 생성했습니다. 자세한 내용은 *LVM 볼륨 그룹 생성* 을 참조하십시오.

절차

1. 볼륨 그룹**ECDHE 001** 에서 **origin** 이라는 논리 볼륨을 생성합니다.

```
# lvcreate -L 1G -n origin vg001
Logical volume "origin" created.
```

2.

100MB 크기의 /dev/vg001/origin 이라는 스냅샷 논리 볼륨을 생성합니다.

```
# lvcreate --size 100M --name snap --snapshot /dev/vg001/origin
  Logical volume "snap" created.
```

--snapshot 을 사용하여 스냅샷을 생성하는 대신 --size,-n 을 사용하는 대신 - L 인수를 사용할 수도 있습니다.

원래 논리 볼륨에 파일 시스템이 포함된 경우 의 임의의 디렉터리에 스냅샷 논리 볼륨을 마운트하여 원본 파일 시스템이 계속 업데이트되는 동안 파일 시스템의 콘텐츠에 액세스하여 백업을 실행할 수 있습니다.

3.

원본 볼륨과 사용 중인 스냅샷 볼륨의 현재 백분율을 표시합니다.

```
# lvs -a -o +devices
  LV     VG    Attr       LSize  Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices
 origin vg001  owi-a-s---  1.00g                                      /dev/sde1(0)
  snap vg001  swi-a-s--- 100.00m     origin 0.00                        /dev/sde1(256)
```

lvdisplay /dev/vg001/origin 명령을 사용하여 모든 스냅샷 논리 볼륨 및 비활성과 같은 논리 볼륨 /dev/vg001/origin상태를 표시할 수도 있습니다.

주의

스냅샷 LV의 공간은 원본 LV가 작성되면 사용됩니다. lvs 명령은 Data% data_percent 필드 값의 현재 스냅샷 공간 사용량을 보고합니다. 스냅샷 공간이 100%에 도달하면 스냅샷이 유효하지 않고 사용할 수 없게 됩니다.

유효하지 않은 스냅샷은 Attr 열의 5번째 위치에 있거나 lvs 의 lv_snapshot_invalid 보고 필드와 함께 보고됩니다. lvremove 명령을 사용하여 유효하지 않은 스냅샷을 제거할 수 있습니다.

4.

선택 사항: 스냅샷이 100% 가득 차기 전에 스냅샷을 확장하고 다음 옵션 중 하나를 사용하여 유효하지 않게 됩니다.

- */etc/lvm.conf* 파일에서 다음 매개 변수를 사용하여 스냅샷을 자동으로 확장하도록 **LVM**을 구성합니다.

snapshot_autoextend_threshold

이 매개 변수에 설정된 값을 초과하면 스냅샷을 확장합니다. 기본적으로 자동 확장을 비활성화하는 **100**으로 설정됩니다. 이 매개변수의 최소 값은 **50**입니다.

snapshot_autoextend_percent

현재 크기의 백분율인 스냅샷에 공간을 추가합니다. 기본적으로 **20**으로 설정됩니다.

다음 예에서 다음 매개 변수를 설정한 후 **1G** 스냅샷은 사용량이**700M**을 초과하면 **1.2G**로 확장됩니다.

예 10.1. 스냅샷 자동 확장

```
# vi /etc/lvm.conf
snapshot_autoextend_threshold = 70
snapshot_autoextend_percent = 20
```

참고

이 기능을 사용하려면 볼륨 그룹에 할당되지 않은 공간이 필요합니다. 스냅샷 자동 확장은 스냅샷에 필요한 최대 계산된 크기 이상으로 스냅샷 볼륨의 크기를 늘리지 않습니다. 스냅샷이 원본을 처리할 수 있을 만큼 크게 확장되면 더 이상 자동 확장을 모니터링하지 않습니다.

- **lvextend** 명령을 사용하여 이 스냅샷을 수동으로 확장합니다.

```
# lvextend -L+100M /dev/vg001/snap
```

추가 리소스

- **lvcreate(8), lvextend(8), lvs(8)** 도움말 페이지
- **/etc/lvm/lvm.conf file**

10.3. 스냅샷을 원래 볼륨에 병합

lvconvert 명령을 --merge 옵션과 함께 사용하여 스냅샷을 원본(원본) 볼륨에 병합합니다. 데이터 또는 파일이 손실되었거나 시스템을 이전 상태로 복원해야 하는 경우 시스템 롤백을 수행할 수 있습니다. 스냅샷 볼륨을 병합한 후 결과 논리 볼륨에 원본 볼륨의 이름, 마이너 번호, UUID가 있습니다. 병합이 진행 중인 동안 병합되는 스냅샷에 지시된 것처럼 원본을 읽거나 씁니다. 병합이 완료되면 병합된 스냅샷이 제거됩니다.

origin 및 snapshot 볼륨이 모두 열려 있고 활성 상태가 아닌 경우 병합이 즉시 시작됩니다. 그렇지 않으면 원본 또는 스냅샷이 활성화된 후 병합이 시작되고 둘 다 닫힙니다. 원본 볼륨이 활성화된 후 종료할 수 없는 원본(예: 루트 파일 시스템)에 스냅샷을 병합할 수 있습니다.

절차

1. 스냅샷 볼륨을 병합합니다. 다음 명령은 스냅샷 볼륨 001/snap 을 원본 으로 병합합니다.

```
# lvconvert --merge vg001/snap
Merging of volume vg001/snap started.
  vg001/origin: Merged: 100.00%
```

2. origin 볼륨을 확인합니다.

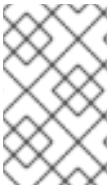
```
# lvs -a -o +devices
  LV     VG    Attr       LSize  Pool Origin Data% Meta% Move Log Cpy%Sync Convert
Devices
  origin vg001 owi-a-s---  1.00g                                      /dev/sde1(0)
```

추가 리소스

- lvconvert(8) man page

10.4. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 생성

새 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷을 생성할 수 있습니다. 또한 생성된 스냅샷에 충분한 공간이 있는지 확인하고 snapshot_lvm_action 매개변수를 확인하여 이름과 충돌하지 않습니다. 생성된 스냅샷을 마운트하려면 snapshot_lvm_action 을 mount 로 설정합니다.



참고

현재 스냅샷 RHEL 시스템 역할은 thin LVM 스냅샷을 지원하지 않습니다.

다음 예제에서는 nouuid 옵션이 설정되며 XFS 파일 시스템으로 작업하는 경우에만 필요합니다. XFS 에서는 동일한 UUID로 여러 파일 시스템을 동시에 마운트하는 것은 지원되지 않습니다.

사전 요구 사항

- 컨트롤 노드 및 관리형 노드를 준비했습니다.
- 관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.
- 관리 노드에 연결하는 데 사용하는 계정에는 sudo 권한이 있습니다.

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: ~/playbook.yml)을 생성합니다.

```
---
- name: Run the snapshot system role
  hosts: managed-node-01.example.com
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
          percent_space_required: 25
          mountpoint: /data1_snapshot
          options: nouuid
          mountpoint_create: true
        - name: data2 snapshot
          vg: data_vg
          lv: data2
          percent_space_required: 25
          mountpoint: /data2_snapshot
          options: nouuid
          mountpoint_create: true

  tasks:
    - name: Create a snapshot set
```

```
    vars:
      snapshot_lvm_action: snapshot
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"

  - name: Verify the set of snapshots for the LVs
    vars:
      snapshot_lvm_action: check
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"
      snapshot_lvm_verify_only: true

  - name: Mount the snapshot set
    vars:
      snapshot_lvm_action: mount
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"

roles:
  - redhat.rhel_system_roles.snapshot
```

여기에서 **snapshot_lvm_set** 매개변수는 동일한 볼륨 그룹(VG)의 특정 논리 볼륨(LV)을 설명합니다. 이 매개 변수를 설정하는 동안 다른 **VG**에서 **LV**를 지정할 수도 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** 파일
- **/usr/share/doc/rhel-system-roles/snapshot/** 디렉터리

10.5. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 마운트 해제

snapshot_lvm_action 매개변수를 **umount** 로 설정하여 특정 스냅샷 또는 모든 스냅샷을 마운트 해제할 수 있습니다.

사전 요구 사항

- *컨트롤 노드 및 관리형 노드를 준비했습니다.*
- *관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.*
- *관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.*

절차

1. *다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.*

   - *특정 **LVM** 스냅샷을 마운트 해제합니다.*

```
---
- name: Unmount a snapshot
  hosts: all
  tasks:
    - name: Unmount data_vg/data2
      vars:
        snapshot_lvm_snapset_name: snapset1
        snapshot_lvm_action: umount
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data2
        snapshot_lvm_mountpoint: /data2_snapshot

  roles:
    - redhat.rhel_system_roles.snapshot
```

   *여기에서 **snapshot_lvm_lv** 매개변수는 특정 논리 볼륨(**LV**)을 설명하고 **snapshot_lvm_vg** 매개변수는 특정 볼륨 그룹(**VG**)을 설명합니다.*

   - *LVM 스냅샷 세트를 마운트 해제합니다.*

```
---
- name: Unmount a set of snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
```

```
      vg: data_vg
      lv: data1
    - name: data2 snapshot
      vg: data_vg
      lv: data2

tasks:
  - name: Unmount a snapshot set
    vars:
      snapshot_lvm_action: umount
      snapshot_lvm_set: "{{ snapshot_lvm_set }}"

roles:
  - redhat.rhel_system_roles.snapshot
```

여기에서 **snapshot_lvm_set** 매개 변수는 동일한 **VG**의 특정 **LV**를 설명합니다. 이 매개 변수를 설정하는 동안 다른 **VG**에서 **LV**를 지정할 수도 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** 파일
- **/usr/share/doc/rhel-system-roles/snapshot/** 디렉터리

10.6. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 확장

새 스냅샷 **RHEL** 시스템 역할을 사용하면 **snapshot _lvm_action** 매개변수를 확장하도록 설정하여 **LVM** 스냅샷을 확장할 수 있습니다. **snapshot_lvm_percent_space_required** 매개변수를 확장 후 스냅샷에 할당해야 하는 필수 공간으로 설정할 수 있습니다.

사전 요구 사항

- *컨트롤 노드 및 관리형 노드를 준비했습니다.*
- *관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.*
- *관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.*

절차

1. *다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.*

   - ***percent_space_required** 매개변수 값을 지정하여 모든 **LVM** 스냅샷을 확장합니다.*

```
---
- name: Extend all snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapshot
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Extend the snapshot set
      vars:
        snapshot_lvm_percent_space_required: 40
        snapshot_lvm_all_vgs: true
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"
        snapshot_lvm_action: extend

  roles:
    - redhat.rhel_system_roles.snapshot
```

   *여기에서 **snapshot_lvm_all_vgs** 매개변수는 모든 볼륨 그룹(VG)의 모든 논리 볼륨(LV)을 설명합니다. **snapshot_lvm_set** 매개 변수는 동일한 VG의 특정 LV를 설명합니다.*

   - *각 볼륨 그룹 및 논리 볼륨 쌍의 다른 값으로 **percent_space_required** 를 설정하여 설*

정된 LVM 스냅샷을 확장합니다.

```
---
- name: Extend the snapshot
  hosts: all
  tasks:
    - name: Extend data1 LV by 30%
      vars:
        snapshot_lvm_snapset_name: snapset1
        percent_space_required: 30
        snapshot_lvm_action: extend
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data1

    - name: Extend data2 LV by 40%
      vars:
        snapshot_lvm_snapset_name: snapset2
        percent_space_required: 40
        snapshot_lvm_action: extend
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data2

  roles:
    - redhat.rhel_system_roles.snapshot
```

여기에서 **snapshot_lvm_lv** 매개변수는 특정 **LV**를 설명하고 **snapshot_lvm_vg** 매개변수는 특정 **VG**를 설명합니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. 플레이북을 실행합니다.

```
$ ansible-playbook ~/playbook.yml
```

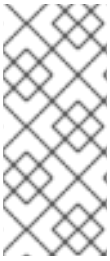
추가 리소스

- **/usr/share/ansible/roles/rhel-system-roles.snapshot/README.md** 파일

- */usr/share/doc/rhel-system-roles/snapshot/* 디렉터리

10.7. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 복원

*새 스냅샷 RHEL 시스템 역할을 사용하면 **snapshot _lvm_action** 매개변수를 되돌리도록 설정하여 LVM 스냅샷 을 원래 볼륨으로 되돌릴 수 있습니다.*



참고

논리 볼륨과 스냅샷 볼륨이 모두 열려 있지 않고 활성 상태가 아니면 되돌리기 작업이 즉시 시작됩니다. 그렇지 않으면 원본 또는 스냅샷이 활성화된 후 시작하고 둘 다 닫힙니다.

사전 요구 사항

- *컨트롤 노드 및 관리형 노드를 준비했습니다.*
- *관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.*
- *관리 노드에 연결하는 데 사용하는 계정에는 **sudo** 권한이 있습니다.*

절차

1. *다음 콘텐츠를 사용하여 플레이북 파일(예: **~/playbook.yml**)을 생성합니다.*

   - *특정 LVM 스냅샷을 원래 볼륨으로 되돌립니다.*

```
---
- name: Revert a snapshot to its original volume
  hosts: all
  tasks:
    - name: Revert data_vg/data2
      vars:
        snapshot_lvm_snapset_name: snapset1
        snapshot_lvm_action: revert
        snapshot_lvm_vg: data_vg
        snapshot_lvm_lv: data2
```

```
  roles:
    - redhat.rhel_system_roles.snapshot
```

여기에서 **snapshot_lvm_lv** 매개변수는 특정 논리 볼륨(LV)을 설명하고 **snapshot_lvm_vg** 매개변수는 특정 볼륨 그룹(VG)을 설명합니다.

- LVM 스냅샷 세트를 원래 볼륨으로 되돌립니다.

```
---
- name: Revert a set of snapshot
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Revert the snapshot set
      vars:
        snapshot_lvm_action: revert
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"

  roles:
    - redhat.rhel_system_roles.snapshot
```

여기에서 **snapshot_lvm_set** 매개 변수는 동일한 **VG**의 특정 **LV**를 설명합니다. 이 매개 변수를 설정하는 동안 다른 **VG**에서 **LV**를 지정할 수도 있습니다.

참고

되돌리기 작업을 완료하는 데 시간이 다소 걸릴 수 있습니다.

2. 플레이북 구문을 확인합니다.

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. *플레이북을 실행합니다.*

```
$ ansible-playbook ~/playbook.yml
```

4. *호스트를 재부팅하거나 다음 단계를 사용하여 논리 볼륨을 비활성화 및 다시 활성화합니다.*

```
$ umount /data1; umount /data2

$ lvchange -an data_vg/data1 data_vg/data2

$ lvchange -ay data_vg/data1 data_vg/data2

$ mount /data1; mount /data2
```

추가 리소스

- */usr/share/ansible/roles/rhel-system-roles.snapshot/README.md* 파일
- */usr/share/doc/rhel-system-roles/snapshot/* 디렉터리

10.8. 스냅샷 RHEL 시스템 역할을 사용하여 LVM 스냅샷 제거

새 스냅샷 RHEL 시스템 역할을 사용하면 스냅샷의 접두사 또는 패턴을 지정하고 snapshot_lvm_action 매개변수를 제거하여 모든 LVM 스냅샷을 제거할 수 있습니다.

사전 요구 사항

- *컨트롤 노드 및 관리형 노드를 준비했습니다.*
- *관리 노드에서 플레이북을 실행할 수 있는 사용자로 제어 노드에 로그인되어 있습니다.*
- *관리 노드에 연결하는 데 사용하는 계정에는 sudo 권한이 있습니다.*

절차

1. 다음 콘텐츠를 사용하여 플레이북 파일(예: ~/playbook.yml)을 생성합니다.

- 특정 LVM 스냅샷을 제거합니다.

```
---
- name: Remove a snapshot
  hosts: all
  tasks:
    - name: Remove snapshots in data_vg
      vars:
        snapshot_lvm_snapset_name: snapset1
        snapshot_lvm_action: remove
        snapshot_lvm_vg: data_vg

  roles:
    - redhat.rhel_system_roles.snapshot
```

여기에서 snapshot_lvm_vg 매개변수는 볼륨 그룹(VG)의 특정 논리 볼륨(LV)을 설명합니다.

- LVM 스냅샷 세트를 제거합니다.

```
---
- name: Remove a set of snapshots
  hosts: all
  vars:
    snapshot_lvm_set:
      name: snapset1
      volumes:
        - name: data1 snapshot
          vg: data_vg
          lv: data1
        - name: data2 snapshot
          vg: data_vg
          lv: data2

  tasks:
    - name: Remove a snapshot set
      vars:
        snapshot_lvm_action: remove
        snapshot_lvm_set: "{{ snapshot_lvm_set }}"

  roles:
    - redhat.rhel_system_roles.snapshot
```

여기에서 snapshot_lvm_set 매개 변수는 동일한 VG의 특정 LV를 설명합니다. 이 매개 변수를 설정하는 동안 다른 VG에서 LV를 지정할 수도 있습니다.

2. *플레이북 구문을 확인합니다.*

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

이 명령은 구문만 검증하고 잘못되었지만 유효한 구성으로부터 보호하지 않습니다.

3. *플레이북을 실행합니다.*

```
$ ansible-playbook ~/playbook.yml
```

추가 리소스

- */usr/share/ansible/roles/rhel-system-roles.snapshot/README.md* 파일
- */usr/share/doc/rhel-system-roles/snapshot/* 디렉터리

11장. 씬 프로비저닝 볼륨 생성 및 관리(THIN VOLUMES)

Red Hat Enterprise Linux는 씬 프로비저닝된 스냅샷 볼륨 및 논리 볼륨을 지원합니다.

논리 볼륨 및 스냅샷 볼륨은 씬 프로비저닝할 수 있습니다.

- 씬 프로비저닝된 논리 볼륨을 사용하여 사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다.
- 씬 프로비저닝 스냅샷 볼륨을 사용하여 동일한 데이터 볼륨에 더 많은 가상 장치를 저장할 수 있습니다.

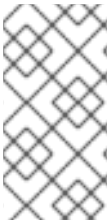
11.1. 씬 프로비저닝 개요

이제 많은 최신 스토리지 스택에서 프로비저닝과 씬 프로비저닝 중에서 선택할 수 있는 기능을 제공합니다.

- 프로비저닝은 실제 사용과 관계없이 블록이 할당된 블록 스토리지의 기존 동작을 제공합니다.
- 씬 프로비저닝을 사용하면 데이터를 저장하는 물리적 장치보다 크기가 클 수 있는 블록 스토리지 풀을 더 많이 프로비저닝하여 과도하게 프로비저닝할 수 있습니다. 개별 블록이 실제로 사용될 때까지 할당되지 않기 때문에 초과 프로비저닝이 가능합니다. 동일한 풀을 공유하는 씬 프로비저닝 장치가 여러 개인 경우 이러한 장치를 과도하게 프로비저닝할 수 있습니다.

씬 프로비저닝을 사용하면 물리 스토리지를 과도하게 커밋할 수 있으며, 대신 씬 풀이라는 여유 공간 풀을 관리할 수 있습니다. 애플리케이션에 필요할 때 이 씬 풀을 임의의 수의 장치에 할당할 수 있습니다. 스토리지 공간을 비용 효율적으로 할당하기 위해 필요한 경우 **thin** 풀을 동적으로 확장할 수 있습니다.

예를 들어, **10**명의 사용자가 애플리케이션에 대해 **100GB** 파일 시스템을 요청하면 각 사용자에 대해 **100GB** 파일 시스템으로 표시되는 항목을 생성할 수 있지만 필요한 경우 사용되는 실제 스토리지가 거의 지원되지 않습니다.



참고

씬 프로비저닝을 사용할 때는 스토리지 풀을 모니터링하고 사용 가능한 물리 공간이 부족할 때 더 많은 용량을 추가해야 합니다.

다음은 씬 프로비저닝 장치를 사용할 때의 몇 가지 이점입니다.

- *사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다.*
- *동일한 데이터 볼륨에 저장할 가상 장치를 더 추가할 수 있습니다.*
- *데이터 요구 사항을 지원하기 위해 논리적으로 자동으로 증가할 수 있는 파일 시스템을 생성할 수 있으며, 사용되지 않은 블록은 풀의 모든 파일 시스템에서 사용할 수 있습니다.*

다음은 씬 프로비저닝 장치를 사용하는 잠재적인 단점입니다.

- *씬 프로비저닝 볼륨은 사용 가능한 물리적 스토리지가 부족할 위험이 있습니다. 기본 스토리지를 과도하게 프로비저닝한 경우 사용 가능한 물리 스토리지 부족으로 인해 중단될 수 있습니다. 예를 들어 백업을 위한 **1T** 물리적 스토리지로 씬 프로비저닝 스토리지 **10T**를 생성하는 경우 **1T**가 고갈되면 볼륨을 사용할 수 없게 됩니다.*
- *볼륨이 씬 프로비저닝 장치 후 계층으로 삭제되지 않으면 사용량에 대한 회계가 정확하지 않습니다. 예를 들어 **-o discard** 마운트 옵션 없이 파일 시스템을 배치하고 씬 프로비저닝된 장치 위에서 **fstrim** 을 주기적으로 실행하지 않으면 이전에 사용된 스토리지를 할당 해제하지 않습니다. 이러한 경우 실제로 사용하지 않더라도 시간이 지남에 따라 전체 프로비저닝 된 양을 사용합니다.*
- *사용 가능한 물리적 공간이 부족하려면 논리 및 물리적 사용을 모니터링해야 합니다.*
- *스냅샷이 있는 파일 시스템에서 **Write (CoW)** 작업 속도가 느려질 수 있습니다.*
- *데이터 블록은 여러 파일 시스템 간에 상호 혼합될 수 있으므로 최종 사용자에게 표시되지 않는 경우에도 기본 스토리지에 대한 임의의 액세스 제한으로 이어질 수 있습니다.*

11.2. 씬 프로비저닝된 논리 볼륨 생성

씬 프로비저닝된 논리 볼륨을 사용하여 사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다. 씬 프로비저닝된 볼륨 세트를 생성하면 요청된 스토리지의 전체 양을 할당하는 대신 사용하는 항목을 할당할 수 있습니다.

lvcreate 명령의 **-T** 또는 **--thin** 옵션을 사용하여 **thin** 풀 또는 **thin** 볼륨을 만들 수 있습니다. **lvcreate** 명령의 **-T** 옵션을 사용하여 단일 명령으로 **thin** 풀과 **thin** 볼륨을 동시에 생성할 수도 있습니다. 이 절차에서는 씬 프로비저닝된 논리 볼륨을 생성하고 확장하는 방법을 설명합니다.

사전 요구 사항

- 볼륨 그룹을 생성했습니다. 자세한 내용은 *LVM 볼륨 그룹 생성* 을 참조하십시오.

절차

1. **thin** 풀을 생성합니다.

```
# lvcreate -L 100M -T vg001/mythinpool
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "mythinpool" created.
```

물리 공간 풀을 생성하므로 풀 크기를 지정해야 합니다. **lvcreate** 명령의 **-T** 옵션은 인수를 사용하지 않습니다. 명령으로 추가된 다른 옵션에서 생성할 장치 유형을 결정합니다. 다음 예와 같이 추가 매개변수를 사용하여 **thin** 풀을 생성할 수도 있습니다.

- **lvcreate** 명령의 **--thinpool** 매개변수를 사용하여 **thin** 풀을 생성할 수도 있습니다. **-T** 옵션과 달리 **--thinpool** 매개변수는 생성 중인 씬 풀 논리 볼륨의 이름을 지정해야 합니다. 다음 예제에서는 **--thinpool** 매개변수를 사용하여 볼륨 그룹인 **mythinpool** 에 크기가 **100M** 인 **thin** 풀을 생성합니다.

```
# lvcreate -L 100M --thinpool mythinpool vg001
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of
data.
  Logical volume "mythinpool" created.
```

- 풀 생성에 대해 스트라이핑이 지원되므로 **-i** 및 **-I** 옵션을 사용하여 스트라이프를 생성할 수 있습니다. 다음 명령은 두 개의 **64 kB** 스트라이프와 **256 kB** 의 청크 크기를 사용하여 볼륨 그룹 **10.0.0.1**에서 **thinpool** 로 이름이 지정된 **100M** 씬 풀을 생성합니다. 또한 이름이 **ECDHE 001/thinvolume**인 **1T thin volume** 을 생성합니다.

참고

볼륨 그룹에 충분한 여유 공간이 있는 두 개의 물리 볼륨이 있는지 또는 thin 풀을 만들 수 없는지 확인합니다.

```
# lvcreate -i 2 -I 64 -c 256 -L 100M -T vg001/thinpool -V 1T --name thinvolume
```

2.

thin 볼륨을 생성합니다.

```
# lvcreate -V 1G -T vg001/mythinpool -n thinvolume
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool
vg001/mythinpool (100.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of
space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.
  Logical volume "thinvolume" created.
```

이 경우 볼륨을 포함하는 풀보다 큰 볼륨의 가상 크기를 지정합니다. 다음 예와 같이 추가 매개변수를 사용하여 씬 볼륨을 생성할 수도 있습니다.

- thin 볼륨과 thin 풀을 모두 생성하려면 lvcreate 명령의 -T 옵션을 사용하고 size 및 virtual size 인수를 둘 다 지정합니다.

```
# lvcreate -L 100M -T vg001/mythinpool -V 1G -n thinvolume
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of
data.
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool
vg001/mythinpool (100.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of
space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.
  Logical volume "thinvolume" created.
```

- 나머지 여유 공간을 사용하여 thin volume 및 thin 풀을 생성하려면 100%FREE 옵션을 사용합니다.

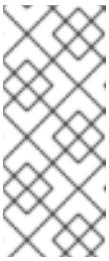
```
# lvcreate -V 1G -l 100%FREE -T vg001/mythinpool -n thinvolume
Thin pool volume with chunk size 64.00 KiB can address at most <15.88 TiB of
data.
  Logical volume "thinvolume" created.
```

-

*기존 논리 볼륨을 씬 풀 볼륨으로 변환하려면 **lvconvert** 명령의 **--thinpool** 매개 변수를 사용합니다. 또한 **--thinpool** 매개변수와 함께 **--poolmetadata** 매개변수를 사용하여 기존 논리 볼륨을 씬 풀 볼륨의 메타데이터 볼륨으로 변환해야 합니다.*

*다음 예제에서는 볼륨 그룹 **lv1** 의 기존 논리 볼륨 **lv1**을 씬 풀 볼륨으로 변환하고 볼륨 그룹**ECDHE 001** 의 기존 논리 볼륨 **lv2** 를 해당 씬 풀 볼륨의 메타데이터 볼륨으로 변환합니다.*

```
# lvconvert --thinpool vg001/lv1 --poolmetadata vg001/lv2
  Converted vg001/lv1 to thin pool.
```



참고

*논리 볼륨을 씬 풀 볼륨 또는 씬 풀 메타데이터 볼륨으로 변환하면 **lvconvert** 가 장치의 콘텐츠를 보존하지 않고 콘텐츠를 덮어쓰므로 논리 볼륨의 콘텐츠가 제거됩니다.*

- *기본적으로 **lvcreate** 명령은 다음 공식을 사용하여 **thin pool** 메타데이터 논리 볼륨의 크기를 설정합니다.*

```
Pool_LV_size / Pool_LV_chunk_size * 64
```

*다수의 스냅샷이 있거나 씬 풀에 대해 작은 청크 크기가 있어 나중에 씬 풀 크기의 크기가 크게 증가하는 경우 **lvcreate** 명령의 **--poolmetadatasize** 매개변수를 사용하여 **thin pool** 의 **metadata** 볼륨의 기본값을 늘려야 할 수 있습니다. 씬 풀의 메타데이터 논리 볼륨에 지원되는 값은 **2MiB**에서 **16GiB** 사이의 범위입니다.*

*다음 예제에서는 **thin pool**의 메타데이터 볼륨의 기본값을 늘리는 방법을 보여줍니다.*

```
# lvcreate -V 1G -l 100%FREE -T vg001/mythinpool --poolmetadatasize 16M -n
thinvolume
Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "thinvolume" created.
```

3. *생성된 **thin** 풀 및 **thin** 볼륨을 확인합니다.*

```
# lvs -a -o +devices
  LV              VG    Attr       LSize   Pool       Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  [lvol0_pmspare]    vg001 ewi-------   4.00m
/dev/sda(0)
  mythinpool         vg001 twi-aotz-- 100.00m                 0.00   10.94
```

```
  mythinpool_tdata(0)
   [mythinpool_tdata] vg001 Twi-ao---- 100.00m
/dev/sda(1)
   [mythinpool_tmeta] vg001 ewi-ao----   4.00m
/dev/sda(26)
   thinvolume        vg001 Vwi-a-tz--   1.00g mythinpool       0.00
```

4. *선택 사항: lvextend 명령을 사용하여 씬 풀의 크기를 확장합니다. 그러나 씬 풀의 크기를 줄일 수는 없습니다.*

참고

thin pool 및 thin volume을 생성하는 동안 -l 100%FREE 인수를 사용하면 이 명령이 실패합니다.

다음 명령은 다른 100M 을 확장하여 크기가 100M 인 기존 씬 풀의 크기를 조정합니다.

```
# lvextend -L+100M vg001/mythinpool
  Size of logical volume vg001/mythinpool_tdata changed from 100.00 MiB (25 extents)
to 200.00 MiB (50 extents).
  WARNING: Sum of all thin volume sizes (1.00 GiB) exceeds the size of thin pool
vg001/mythinpool (200.00 MiB).
  WARNING: You have not turned on protection against thin pools running out of
space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger
automatic extension of thin pools before they get full.

  Logical volume vg001/mythinpool successfully resized
```

```
# lvs -a -o +devices
  LV              VG    Attr      LSize   Pool       Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  [lvol0_pmspare]    vg001 ewi-------   4.00m
/dev/sda(0)
  mythinpool         vg001 twi-aotz-- 200.00m                 0.00   10.94
mythinpool_tdata(0)
  [mythinpool_tdata] vg001 Twi-ao---- 200.00m
/dev/sda(1)
  [mythinpool_tdata] vg001 Twi-ao---- 200.00m
/dev/sda(27)
  [mythinpool_tmeta] vg001 ewi-ao----   4.00m
/dev/sda(26)
  thinvolume         vg001 Vwi-a-tz--   1.00g mythinpool       0.00
```

5. *선택 사항: thin pool 및 thin 볼륨의 이름을 바꾸려면 다음 명령을 사용합니다.*

```
# lvrename vg001/mythinpool vg001/mythinpool1
  Renamed "mythinpool" to "mythinpool1" in volume group "vg001"

# lvrename vg001/thinvolume vg001/thinvolume1
  Renamed "thinvolume" to "thinvolume1" in volume group "vg001"
```

이름을 변경한 후 thin 풀 및 thin 볼륨을 확인합니다.

```
# lvs
  LV          VG      Attr     LSize   Pool       Origin Data%  Move Log Copy%  Convert
mythinpool1 vg001   twi-a-tz 100.00m                    0.00
thinvolume1 vg001   Vwi-a-tz   1.00g mythinpool1        0.00
```

6. *선택 사항: thin 풀을 제거하려면 다음 명령을 사용합니다.*

```
# lvremove -f vg001/mythinpool1
  Logical volume "thinvolume1" successfully removed.
  Logical volume "mythinpool1" successfully removed.
```

추가 리소스

- *lvcreate(8), lvrename(8), lvs(8), and lvconvert(8) man pages*

11.3. 청크 크기 개요

청크는 스냅샷 스토리지 전용 물리 디스크의 가장 큰 단위입니다.

청크 크기 사용에 대해 다음 기준을 사용합니다.

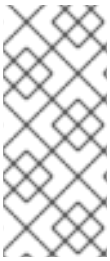
- *청크 크기가 작아지면 메타데이터가 더 많이 필요하며 성능이 저하되지만 스냅샷에 더 나은 공간 사용률을 제공합니다.*
- *더 큰 청크 크기는 메타데이터 조작을 덜 필요로 하지만 스냅샷을 공간을 덜 효율적으로 만듭니다.*

기본값인 lvm2 는 64KiB 청크 크기로 시작하고 이러한 청크 크기에 대해 좋은 메타데이터 크기를 추정합니다. lvm2 의 최소 메타데이터 크기는 2MiB입니다. 메타데이터 크기가 128MiB보다 커야 하는 경우 청크 크기를 늘리기 시작하여 메타데이터 크기가 컴팩트하게 유지됩니다. 그러나 이로 인해 스냅샷 사용에

효율적인 공간이 더 적은 일부 청크 크기 값이 발생할 수 있습니다. 이러한 경우 작은 청크 크기와 더 큰 메타데이터 크기가 더 나은 옵션입니다.

*요구 사항에 따라 청크 크기를 지정하려면 **-c** 또는 **--chunksize** 매개변수를 사용하여 **lvm2** 예상 청크 크기를 덮어씁니다. **thinpool**이 생성된 후에는 청크 크기를 변경할 수 없습니다.*

*볼륨 데이터 크기가 **TiB** 범위에 있는 경우 최대 지원되는 크기인 **~15.8GiB**를 메타데이터 크기로 사용하고 요구 사항에 따라 청크 크기를 설정합니다. 그러나 볼륨의 데이터 크기를 확장해야 하며 청크 크기가 작은 경우 메타데이터 크기를 늘릴 수 없습니다.*



참고

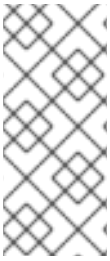
청크 크기와 메타데이터 크기의 부적절한 조합을 사용하면 사용자가 메타데이터 에서 공간이 부족하거나 주소 지정 가능한 씬 풀 데이터 크기로 인해 씬 풀 크기를 추가로 늘리지 못할 수 있습니다.

추가 리소스

- ***lvmthin>-& lt; man** 페이지*

11.4. 씬 프로비저닝된 스냅샷 볼륨

***Red Hat Enterprise Linux**는 씬 프로비저닝된 스냅샷 볼륨을 지원합니다. **thin** 논리 볼륨의 스냅샷도 씬 논리 볼륨(**LV**)을 생성합니다. 씬 스냅샷 볼륨은 다른 씬 볼륨과 동일한 특성을 갖습니다. 볼륨을 독립적으로 활성화하고, 볼륨을 확장하고, 볼륨 이름을 바꾸며, 볼륨을 제거하고, 볼륨 스냅샷을 만들 수도 있습니다.*



참고

*모든 **LVM** 스냅샷 볼륨 및 모든 씬 볼륨과 유사하게 **thin** 스냅샷 볼륨은 클러스터의 노드에서 지원되지 않습니다. 스냅샷 볼륨은 하나의 클러스터 노드에서만 활성화해야 합니다.*

*기존 스냅샷은 생성된 각 스냅샷에 새 공간을 할당해야 합니다. 여기서 데이터는 원본이 변경될 때 보존됩니다. 그러나 씬 프로비저닝 스냅샷은 원본과 동일한 공간을 공유합니다. **thin LVs**의 스냅샷은 **thin LV** 및 해당 스냅샷에 공통된 데이터 블록이 공유되므로 효율적입니다. **thin LVs**의 스냅샷 또는 기타 씬 스냅샷을 생성할 수 있습니다. 제귀 스냅샷에 공통된 블록은 **thin** 풀에서도 공유됩니다.*

씬 스냅샷 볼륨은 다음과 같은 이점을 제공합니다.

- 원본의 스냅샷 수를 늘리면 성능에 미치는 영향은 무시할 수 있습니다.
- 새 데이터만 작성되고 각 스냅샷에 복사되지 않으므로 씬 스냅샷 볼륨은 디스크 사용량을 줄일 수 있습니다.
- 기존 스냅샷의 요구 사항인 원본과 함께 **thin** 스냅샷 볼륨을 동시에 활성화할 필요가 없습니다.
- 스냅샷에서 원본을 복원할 때 **thin** 스냅샷을 병합할 필요가 없습니다. 원본을 제거하고 대신 스냅샷을 사용할 수 있습니다. 기존 스냅샷에는 변경 사항을 저장하는 별도의 볼륨이 있으며, 다시 복사해야 합니다. 즉, 재설정을 위해 원본과 병합됩니다.
- 기존 스냅샷과 비교하여 허용된 스냅샷 수에 대한 제한이 크게 높아집니다.

thin 스냅샷 볼륨을 사용하는 데는 많은 이점이 있지만 기존 **LVM** 스냅샷 볼륨 기능이 필요에 더 적합할 수 있는 몇 가지 사용 사례가 있습니다. 모든 유형의 볼륨에서 기존 스냅샷을 사용할 수 있습니다. 그러나 **thin-snapshot**을 사용하려면 **thin-provisioning**을 사용해야 합니다.

참고

씬 스냅샷 볼륨의 크기를 제한할 수 없습니다. 스냅샷은 필요한 경우 씬 풀의 모든 공간을 사용합니다. 일반적으로 사용할 스냅샷 형식을 결정할 때 사이트의 특정 요구 사항을 고려해야 합니다.

기본적으로 씬 스냅샷 볼륨은 일반 활성화 명령 중에 건너뜁니다.

11.5. 씬 프로비저닝된 스냅샷 볼륨 생성

씬 프로비저닝 스냅샷 볼륨을 사용하여 동일한 데이터 볼륨에 더 많은 가상 장치를 저장할 수 있습니다.

중요

씬 스냅샷 볼륨을 생성할 때 볼륨의 크기를 지정하지 마십시오. **size** 매개변수를 지정하면 생성되는 스냅샷이 씬 스냅샷 볼륨이 아니며 데이터를 저장하는 데 **thin pool**을 사용하지 않습니다. 예를 들어, 원래 볼륨이 씬 볼륨인 경우에도 **lvcreate -svirtualization/thinvolume -L10M** 은 씬 스냅샷을 생성하지 않습니다.

씬 프로비저닝된 원본 볼륨 또는 씬 프로비저닝된 원본 볼륨에 대해 씬 스냅샷을 생성할 수 있습니다. 다음 절차에서는 씬 프로비저닝된 스냅샷 볼륨을 생성하는 다양한 방법을 설명합니다.

사전 요구 사항

- 씬 프로비저닝된 논리 볼륨을 생성했습니다. 자세한 내용은 *씬 프로비저닝 개요를* 참조하십시오.

절차

- 씬 프로비저닝된 스냅샷 볼륨을 생성합니다. 다음 명령은 씬 프로비저닝된 논리 볼륨**ECDHE 001/thinvolume**의 **mysnapshot1** 이라는 씬 프로비저닝된 스냅샷 볼륨을 생성합니다.

```
# lvcreate -s --name mysnapshot1 vg001/thinvolume
  Logical volume "mysnapshot1" created
```

```
# lvs
  LV          VG      Attr     LSize   Pool       Origin     Data%  Move Log Copy%  Convert
  mysnapshot1 vg001   Vwi-a-tz   1.00g mythinpool thinvolume   0.00
  mythinpool  vg001   twi-a-tz 100.00m                        0.00
  thinvolume  vg001   Vwi-a-tz   1.00g mythinpool             0.00
```

참고

씬 프로비저닝을 사용할 때는 스토리지 관리자가 스토리지 풀을 모니터링하고 가득 차기 시작할 때 용량을 더 추가하는 것이 중요합니다. **thin** 볼륨의 크기를 확장하는 방법에 대한 자세한 내용은 *씬 프로비저닝된 논리 볼륨 생성* 을 참조하십시오.

- 씬 프로비저닝된 논리 볼륨의 스냅샷을 생성할 수도 있습니다. 프로비저닝되지 않은 논리 볼륨은 씬 풀에 포함되어 있지 않으므로 외부 원본이라고 합니다. 외부 원본 볼륨은 다른 씬 풀의 경우에도 여러 씬 프로비저닝 스냅샷 볼륨에서 사용하고 공유할 수 있습니다. 씬 프로비저닝된 스냅샷이 생성될 때 외부 원본은 비활성 상태이고 읽기 전용이어야 합니다.

다음 예제에서는 **origin_volume** 이라는 읽기 전용 비활성 논리 볼륨의 씬 스냅샷 볼륨을 생성합니다. **thin** 스냅샷 볼륨의 이름은 **mythinsnap** 입니다. 그런 다음 논리 볼륨 **origin_volume** 은 기존 **thin poolECDHE 001 /pool** 을 사용하는 볼륨 그룹 **mythinsnap** 의 씬 스냅샷 볼륨 **mythinsnap**의 씬 외부 원본이 됩니다. 원본 볼륨은 스냅샷 볼륨과 동일한 볼륨 그룹에 있어야 합니다. **origin** 논리 볼륨을 지정할 때 볼륨 그룹을 지정하지 마십시오.

```
# lvcreate -s --thinpool vg001/pool origin_volume --name mythinsnap
```

- 다음 명령을 실행하여 첫 번째 스냅샷 볼륨의 두 번째 씬 프로비저닝 스냅샷 볼륨을 생성할 수 있습니다.

```
# lvcreate -s vg001/mysnapshot1 --name mysnapshot2
Logical volume "mysnapshot2" created.
```

세 번째 씬 프로비저닝된 스냅샷 볼륨을 생성하려면 다음 명령을 사용합니다.

```
# lvcreate -s vg001/mysnapshot2 --name mysnapshot3
Logical volume "mysnapshot3" created.
```

검증

- **thin snapshot** 논리 볼륨의 모든 항목 및 하위 항목 목록을 표시합니다.

```
$ lvs -o name,lv_ancestors,lv_descendants vg001
 LV          Ancestors                    Descendants
 mysnapshot2 mysnapshot1,thinvolume       mysnapshot3
 mysnapshot1 thinvolume                   mysnapshot2,mysnapshot3
 mysnapshot3 mysnapshot2,mysnapshot1,thinvolume
 mythinpool
 thinvolume                               mysnapshot1,mysnapshot2,mysnapshot3
```

여기로,

- **thinvolume** 은 볼륨 그룹**ECDHE 001** 의 원본 볼륨입니다.
- **mysnapshot1** 은 **thinvolume**의 스냅샷입니다.
- **mysnapshot2** 는 **mysnapshot1**의 스냅샷입니다.

- ***mysnapshot3 은 mysnapshot2의 스냅샷입니다.***

참고

lv_ancestors 및 lv_descendants 필드에 기존 종속 항목이 표시됩니다. 그러나 제거된 항목은 체인의 중간에서 제거된 경우 종속성 체인을 손상시킬 수 있습니다.

추가 리소스

- ***lvcreate(8) 매뉴얼 페이지***

12장. 캐싱을 활성화하여 논리 볼륨 성능 개선

캐싱을 LVM 논리 볼륨에 추가하여 성능을 향상시킬 수 있습니다. LVM은 SSD와 같은 빠른 장치를 사용하여 I/O 작업을 논리 볼륨에 캐시합니다.

다음 절차에서는 빠른 장치에서 특수 LV를 생성하고 성능을 개선하기 위해 이 특수 LV를 원래 LV에 첨부합니다.

12.1. LVM의 캐싱 방법

LVM은 다음과 같은 종류의 캐싱을 제공합니다. 각 논리 볼륨의 다양한 종류의 I/O 패턴에 적합합니다.

dm-cache

이 방법을 사용하면 더 빠른 볼륨에서 자주 사용하는 데이터에 액세스할 수 있습니다. 이 메서드는 읽기 및 쓰기 작업을 모두 캐시합니다.

dm-cache 메서드는 캐시 의 논리 볼륨을 생성합니다.

dm-writecache

이 메서드는 쓰기 작업만 캐시합니다. 더 빠른 볼륨은 쓰기 작업을 저장한 다음 백그라운드에서 느린 디스크로 마이그레이션합니다. 더 빠른 볼륨은 일반적으로 SSD 또는 영구 메모리(PMEM) 디스크입니다.

dm-writecache 메서드는 writecache 유형의 논리 볼륨을 생성합니다.

추가 리소스

- lvmcache(7) man page

12.2. LVM 캐싱 구성 요소

LVM에서는 LVM 논리 볼륨에 캐시를 추가할 수 있습니다. LVM 캐싱은 다음 LVM 논리 볼륨 유형을 사용합니다.

Main LV

크기가 크고 느리거나 원래 볼륨입니다.

캐시 풀 LV

기본 LV에서 데이터를 캐싱하는 데 사용할 수 있는 복합 LV입니다. 캐시 데이터를 관리하기 위한 두 개의 하위 LV(캐시 데이터 보관용 데이터 및 메타데이터)가 있습니다. 데이터 및 메타데이터에 대한 특정 디스크를 구성할 수 있습니다. dm-cache 와 함께만 캐시 풀을 사용할 수 있습니다.

Cachevol LV

기본 LV에서 데이터를 캐싱하는 데 사용할 수 있는 선형 LV입니다. 데이터 및 메타데이터에 대해 별도의 디스크를 구성할 수 없습니다. cachevol 은 dm-cache 또는 dm-writecache 에서만 사용할 수 있습니다.

연결된 모든 LV가 동일한 볼륨 그룹에 있어야 합니다.

기본 논리 볼륨(LV)을 캐시된 데이터를 보유하는 더 빠르고 일반적으로 작은 LV와 결합할 수 있습니다. fast LV는 SSD 드라이브와 같은 빠른 블록 장치에서 생성됩니다. 논리 볼륨에 대한 캐싱을 활성화하면 LVM의 이름을 변경하고 원래 볼륨을 숨기고 원래 논리 볼륨으로 구성된 새 논리 볼륨을 제공합니다. 새 논리 볼륨의 구성은 캐싱 방법과 cachevol 또는 cachepool 옵션을 사용 중인지에 따라 달라집니다.

cachevol 및 cachepool 옵션은 캐싱 구성 요소의 배치에 대해 다른 수준의 제어를 노출합니다.

- cachevol 옵션을 사용하면 빠른 장치는 캐시된 데이터 블록 복사본과 캐시 관리를 위한 메타데이터 모두를 저장합니다.
- cachepool 옵션을 사용하면 별도의 장치에서 캐시된 데이터 블록 복사본과 캐시 관리를 위한 메타데이터를 저장할 수 있습니다.

  dm-writecache 방법은 cachepool 과 호환되지 않습니다.

모든 구성에서 LVM은 모든 캐싱 구성 요소를 함께 그룹화하는 단일 결과 장치를 노출합니다. 결과 장치의 이름은 느린 원래 논리 볼륨과 동일합니다.

추가 리소스

- lvmcache(7) man page

- 씬 프로비저닝 볼륨 생성 및 관리(thin volumes)

12.3. 논리 볼륨에 대한 DM-CACHE 캐싱 활성화

이 절차에서는 **dm-cache** 메서드를 사용하여 논리 볼륨에서 일반적으로 사용되는 데이터를 캐싱할 수 있습니다.

사전 요구 사항

- **dm-cache** 를 사용하여 속도를 높이려는 느린 논리 볼륨이 시스템에 있습니다.
- 느린 논리 볼륨이 포함된 볼륨 그룹에는 **fast** 블록 장치에서 사용되지 않은 물리 볼륨도 포함되어 있습니다.

절차

1. 빠른 장치에 **cachevol** 볼륨을 생성합니다.

```
# lvcreate --size cachevol-size --name <fastvol> <vg> </dev/fast-pv>
```

다음 값을 바꿉니다.

cachevol-size

cachevol 볼륨의 크기 (예: **5G**)

fastvol

cachevol 볼륨의 이름

vg

볼륨 그룹 이름

/dev/fast-pv

빠른 블록 장치의 경로(예: **/dev/sdf**)

예 **12.1. cachevol** 볼륨 생성

```
# lvcreate --size 5G --name fastvol vg /dev/sdf
Logical volume "fastvol" created.
```

2. **cachevol** 볼륨을 기본 논리 볼륨에 연결하여 캐싱을 시작합니다.

```
# lvconvert --type cache --cachevol <fastvol> <vg/main-lv>
```

다음 값을 바꿉니다.

fastvol

cachevol 볼륨의 이름

vg

볼륨 그룹 이름

main-lv

느린 논리 볼륨의 이름입니다.

예 **12.2.** 기본 **LV**에 **cachevol** 볼륨 연결

```
# lvconvert --type cache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]: y
Logical volume vg/main-lv is now cached.
```

검증 단계

- 새로 생성된 논리 볼륨에 **dm-cache** 가 활성화되어 있는지 확인합니다.

```
# lvs --all --options +devices <vg>

LV              Pool          Type   Devices
main-lv         [fastvol_cvol] cache  main-lv_corig(0)
[fastvol_cvol]                linear /dev/fast-pv
[main-lv_corig]               linear /dev/slow-pv
```

추가 리소스

- ***lvmcache(7) man page***

12.4. 논리 볼륨의 CACHEPOOL을 사용하여 DM-CACHE 캐싱 활성화

이 프로세스를 사용하면 캐시 데이터 및 캐시 메타데이터 논리 볼륨을 개별적으로 생성한 다음 볼륨을 캐시 풀로 결합할 수 있습니다.

사전 요구 사항

- ***dm-cache** 를 사용하여 속도를 높이려는 느린 논리 볼륨이 시스템에 있습니다.*
- *느린 논리 볼륨이 포함된 볼륨 그룹에는 **fast** 블록 장치에서 사용되지 않은 물리 볼륨도 포함되어 있습니다.*

절차

1. *빠른 장치에 **cachepool** 볼륨을 생성합니다.*

```
# lvcreate --type cache-pool --size <cachepool-size> --name <fastpool> <vg /dev/fast>
```

다음 값을 바꿉니다.

cachepool-size

캐시 풀 의 크기 (예: 5G)

fastpool

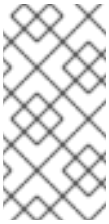
***cachepool** 볼륨의 이름*

vg

볼륨 그룹 이름

/dev/fast

빠른 블록 장치 경로(예: /dev/sdf1)



참고

cache-pool을 생성할 때 **--poolmetadata** 옵션을 사용하여 풀 메타데이터의 위치를 지정할 수 있습니다.

예 **12.3.** 캐시 풀 볼륨 생성

```
# lvcreate --type cache-pool --size 5G --name fastpool vg /dev/sde
Logical volume "fastpool" created.
```

2. 캐시풀 을 기본 논리 볼륨에 연결하여 캐싱을 시작합니다.

```
# lvconvert --type cache --cachepool <fastpool> <vg/main>
```

다음 값을 바꿉니다.

fastpool

cachepool 볼륨의 이름

vg

볼륨 그룹 이름

main

느린 논리 볼륨의 이름입니다.

예 **12.4.** 기본 **LV**에 **cachepool** 연결

```
# lvconvert --type cache --cachepool fastpool vg/main
Do you want wipe existing metadata of cache pool vg/fastpool? [y/n]: y
Logical volume vg/main is now cached.
```

검증 단계

- **cache-pool** 유형을 사용하여 새로 생성된 **devicevolume**을 검사합니다.

```
# lvs --all --options +devices <vg>
```

```
LV                Pool            Type       Devices
[fastpool_cpool]                  cache-pool  fastpool_pool_cdata(0)
[fastpool_cpool_cdata]              linear     /dev/sdf1(4)
[fastpool_cpool_cmeta]              linear     /dev/sdf1(2)
[lvol0_pmspare]                   linear     /dev/sdf1(0)
main              [fastpoool_cpool]  cache      main_corig(0)
[main_corig]                      linear     /dev/sdf1(O)
```

추가 리소스

- lvcreate(8) 매뉴얼 페이지
- lvmcache(7) man page
- lvconvert(8) man page

12.5. 논리 볼륨에 대한 DM-WRITECACHE 캐싱 활성화

이 절차에서는 dm-writecache 메서드를 사용하여 논리 볼륨에 쓰기 I/O 작업을 캐싱할 수 있습니다.

사전 요구 사항

- dm-writecache 를 사용하여 속도를 높일 수 있는 느린 논리 볼륨이 시스템에 있습니다.
- 느린 논리 볼륨이 포함된 볼륨 그룹에는 fast 블록 장치에서 사용되지 않은 물리 볼륨도 포함되어 있습니다.
- 느린 논리 볼륨이 활성 상태이면 비활성화합니다.

절차

1. 느린 논리 볼륨이 활성화된 경우 비활성화합니다.

```
# lvchange --activate n <vg>/<main-lv>
```

다음 값을 바꿉니다.

vg

볼륨 그룹 이름

main-lv

느린 논리 볼륨의 이름입니다.

2. 빠른 장치에서 비활성화된 **cachevol** 볼륨을 생성합니다.

```
# lvcreate --activate n --size <cachevol-size> --name <fastvol> <vg> </dev/fast-pv>
```

다음 값을 바꿉니다.

cachevol-size

cachevol 볼륨의 크기 (예: **5G**)

fastvol

cachevol 볼륨의 이름

vg

볼륨 그룹 이름

/dev/fast-pv

빠른 블록 장치의 경로(예: **/dev/sdf**)

예 **12.5.** 비활성화된 **cachevol** 볼륨 생성

```
# lvcreate --activate n --size 5G --name fastvol vg /dev/sdf
WARNING: Logical volume vg/fastvol not zeroed.
Logical volume "fastvol" created.
```

3. **cachevol** 볼륨을 기본 논리 볼륨에 연결하여 캐싱을 시작합니다.

```
# lvconvert --type writecache --cachevol <fastvol> <vg/main-lv>
```

다음 값을 바꿉니다.

fastvol

***cachevol** 볼륨의 이름*

vg

볼륨 그룹 이름

main-lv

느린 논리 볼륨의 이름입니다.

*예 **12.6.** 기본 **LV**에 **cachevol** 볼륨 연결*

```
# lvconvert --type writecache --cachevol fastvol vg/main-lv
Erase all existing data on vg/fastvol? [y/n]?: y
Using writecache block size 4096 for unknown file system block size, logical block size 512, physical block size 512.
WARNING: unable to detect a file system block size on vg/main-lv
WARNING: using a writecache block size larger than the file system block size may corrupt the file system.
Use writecache block size 4096? [y/n]: y
Logical volume vg/main-lv now has writecache.
```

4. *결과 논리 볼륨을 활성화합니다.*

```
# lvchange --activate y <vg/main-lv>
```

다음 값을 바꿉니다.

vg

볼륨 그룹 이름

main-lv

느린 논리 볼륨의 이름입니다.

검증 단계

- *새로 생성된 장치를 확인합니다.*

```
# lvs --all --options +devices vg

LV              VG Attr       LSize   Pool          Origin          Data%  Meta%  Move Log
Cpy%Sync Convert Devices
 main-lv         vg Cwi-a-C--- 500.00m [fastvol_cvol] [main-lv_wcorig] 0.00
main-lv_wcorig(0)
 [fastvol_cvol]   vg Cwi-aoC--- 252.00m
/dev/sdc1(0)
 [main-lv_wcorig] vg owi-aoC--- 500.00m
/dev/sdb1(0)
```

추가 리소스

- ***lvmcache(7) man page***

12.6. 논리 볼륨의 캐싱 비활성화

*이 절차에서는 현재 논리 볼륨에서 활성화된 **dm-cache** 또는 **dm-writecache** 캐싱을 비활성화합니다.*

사전 요구 사항

- *논리 볼륨에서 캐싱이 활성화됩니다.*

절차

1. *논리 볼륨을 비활성화합니다.*

```
# lvchange --activate n <vg>/<main-lv>
```

   ***skopeo**를 볼륨 그룹 이름으로 바꾸고 **main-lv** 를 캐싱이 활성화된 논리 볼륨의 이름으로 바꿉니다.*

2. ***cachevol** 또는 **cachepool** 볼륨을 분리합니다.*

```
# lvconvert --splitcache <vg>/<main-lv>
```

다음 값을 바꿉니다.

skopeo를 볼륨 그룹 이름으로 바꾸고 **main-lv** 를 캐싱이 활성화된 논리 볼륨의 이름으로 바꿉니다.

예 12.7. cachevol 또는 cachepool 볼륨 분리

```
# lvconvert --splitcache vg/main-lv
Detaching writecache already clean.
Logical volume vg/main-lv writecache has been detached.
```

검증 단계

- 논리 볼륨이 더 이상 함께 연결되지 않았는지 확인합니다.

```
# lvs --all --options +devices <vg>

LV      Attr       Type   Devices
fastvol -wi------- linear /dev/fast-pv
main-lv -wi------- linear /dev/slow-pv
```

추가 리소스

- **lvmcache(7)** 도움말 페이지

13장. 논리 볼륨 활성화

기본적으로 논리 볼륨을 생성하면 활성 상태입니다. 활성 상태인 논리 볼륨은 블록 장치를 통해 사용할 수 있습니다. 활성화된 논리 볼륨에 액세스할 수 있으며 변경될 수 있습니다.

개별 논리 볼륨을 비활성화해야 하므로 커널에 알 수 없는 다양한 상황이 있습니다. **lvchange** 명령의 **-a** 옵션을 사용하여 개별 논리 볼륨을 활성화하거나 비활성화할 수 있습니다.

다음은 개별 논리 볼륨을 비활성화하는 형식입니다.

```
# lvchange -an vg/lv
```

다음은 개별 논리 볼륨을 활성화하는 형식입니다.

```
# lvchange -ay vg/lv
```

skopeo change 명령의 **-a** 옵션을 사용하여 볼륨 그룹의 모든 논리 볼륨을 활성화하거나 비활성화할 수 있습니다. 이는 볼륨 그룹의 개별 논리 볼륨에서 **lvchange -a** 명령을 실행하는 것과 동일합니다.

다음은 볼륨 그룹의 모든 논리 볼륨을 비활성화하는 형식입니다.

```
# vgchange -an vg
```

다음은 볼륨 그룹의 모든 논리 볼륨을 활성화하는 형식입니다.

```
# vgchange -ay vg
```

참고

수동 활성화 중에 **systemd**는 **systemd -mount** 장치를 마스킹하지 않는 한 **/etc/fstab** 파일에서 해당 마운트 지점을 사용하여 **LVM** 볼륨을 자동으로 마운트합니다.

13.1. 논리 볼륨 및 볼륨 그룹의 자동 활성화 제어

논리 볼륨 자동 활성화는 시스템을 시작하는 동안 논리 볼륨의 이벤트 기반 자동 활성화를 나타냅니다. 시스템(**device** 온라인 이벤트)에서 장치를 사용할 수 있게 되면 **systemd/udev** 는 각 장치에 대해 **lvm2-**

pvscan 서비스를 실행합니다. 이 서비스는 이름이 지정된 장치를 읽는 pvscan --cache -aay device 명령을 실행합니다. 장치가 볼륨 그룹에 속하는 경우 pvscan 명령은 해당 볼륨 그룹의 모든 물리 볼륨이 시스템에 있는지 확인합니다. 이 경우 명령은 해당 볼륨 그룹에서 논리 볼륨을 활성화합니다.

VG 또는 LV에서 autoactivation 속성을 설정할 수 있습니다. autoactivation 속성이 비활성화되면 -aay 옵션을 사용하여 자동 활성화를 수행하는 명령으로 VG 또는 LV가 활성화되지 않습니다(예: -aay 옵션). VG에서 자동 활성화를 비활성화하면 해당 VG에서 LV가 자동으로 활성화되지 않으며 autoactivation 속성이 적용되지 않습니다. VG에서 자동 활성화가 활성화된 경우 개별 LV에 대해 자동 활성화를 비활성화할 수 있습니다.

절차

- *다음 방법 중 하나로 자동 활성화 설정을 업데이트할 수 있습니다.*
  - *명령줄을 사용하여 VG의 자동 활성화를 제어합니다.*

```
# vgchange --setautoactivation <y|n>
```

  - *명령줄을 사용하여 LV 자동 활성화를 제어합니다.*

```
# lvchange --setautoactivation <y|n>
```

  - *다음 구성 옵션 중 하나를 사용하여 /etc/lvm/lvm.conf 구성 파일에서 LV 자동 활성화를 제어합니다.*
    - *global/event_activation*

      event_activation 가 비활성화되면 systemd/udev 는 시스템을 시작하는 동안 여러 물리 볼륨에서만 논리 볼륨을 자동으로 활성화합니다. 모든 물리 볼륨이 아직 표시되지 않은 경우 일부 논리 볼륨이 자동으로 활성화되지 않을 수 있습니다.

    - *activation/auto_activation_volume_list*

      auto_activation_volume_list 를 빈 목록으로 설정하면 자동 활성화가 완전히 비활성화됩니다. auto_activation_volume_list 를 특정 논리 볼륨으로 설정하고 볼륨 그룹은 해당 논리 볼륨으로 자동 활성화됩니다.

추가 리소스

- */etc/lvm/lvm.conf 구성 파일*

- *lvmautoactivation(7) 도움말 페이지*

13.2. 논리 볼륨 활성화 제어

다음과 같은 방법으로 논리 볼륨의 활성화를 제어할 수 있습니다.

- */etc/lvm/conf 파일의 activation/volume_list 설정을 통해. 이를 통해 활성화되는 논리 볼륨을 지정할 수 있습니다. 이 옵션 사용에 대한 자세한 내용은 /etc/lvm/lvm.conf 구성 파일을 참조하십시오.*

- *논리 볼륨에 대한 활성화 건너뛰기 플래그를 의미합니다. 이 플래그가 논리 볼륨에 대해 설정된 경우 정상적인 활성화 명령 중에 볼륨을 건너뜁니다.*

또는 lvcreate 또는 lvchange 명령에 --setactivationskip y|n 옵션을 사용하여 활성화 건너뛰기 플래그를 활성화하거나 비활성화할 수 있습니다.

절차

- *다음과 같은 방법으로 논리 볼륨에 활성화 건너뛰기 플래그를 설정할 수 있습니다.*

  - *논리 볼륨에 활성화 건너뛰기 플래그가 설정되어 있는지 확인하려면 다음 예와 같이 k 속성을 표시하는 lvs 명령을 실행합니다.*

```
# lvs vg/thin1s1
  LV       VG  Attr       LSize Pool  Origin
  thin1s1  vg  Vwi---tz-k 1.00t pool0 thin1
```

standard -ay 또는 --activate y 옵션 외에도 -K 또는 --ignoreactivationskip 옵션을 사용하여 k 속성이 설정된 논리 볼륨을 활성화할 수 있습니다.

기본적으로 씬 스냅샷 볼륨은 생성될 때 활성화 건너뛰기 위해 플래그가 지정됩니다. /etc/lvm/lvm.conf 파일의 auto_set_activation_skip 설정을 사용하여 새 씬 스냅샷 볼륨에서 기본 활성화 건너뛰 설정을 제어할 수 있습니다.

- *다음 명령은 **activation skip** 플래그가 설정된 **thin snapshot** 논리 볼륨을 활성화합니다.*

```
# lvchange -ay -K VG/SnapLV
```

- *다음 명령은 활성화 **skip** 플래그 없이 **thin** 스냅샷을 생성합니다.*

```
# lvcreate -n SnapLV -kn -s vg/ThinLV --thinpool vg/ThinPoolLV
```

- *다음 명령은 스냅샷 논리 볼륨에서 활성화 **skip** 플래그를 제거합니다.*

```
# lvchange -kn VG/SnapLV
```

검증 단계

- *활성화 건너뛰기 플래그가 없는 **thin** 스냅샷이 생성되었는지 확인합니다.*

```
# lvs -a -o +devices,segtype
  LV              VG          Attr       LSize    Pool       Origin Data%  Meta%  Move Log
Cpy%Sync Convert Devices             Type
  SnapLV          vg          Vwi-a-tz--  100.00m ThinPoolLV ThinLV 0.00
thin
  ThinLV          vg          Vwi-a-tz--  100.00m ThinPoolLV        0.00
thin
  ThinPoolLV      vg          twi-aotz--  100.00m                   0.00   10.94
ThinPoolLV_tdata(0) thin-pool
  [ThinPoolLV_tdata] vg         Twi-ao----  100.00m
/dev/sdc1(1)       linear
  [ThinPoolLV_tmeta] vg         ewi-ao----    4.00m
/dev/sdd1(0)       linear
  [lvol0_pmspare]    vg         ewi-------    4.00m
/dev/sdc1(0)       linear
```

13.3. 공유 논리 볼륨 활성화

*다음과 같이 **lvchange** 및 **Cryostat change** 명령의 **-a** 옵션을 사용하여 공유 논리 볼륨의 논리 볼륨 활성화를 제어할 수 있습니다.*

명령	활성화

명령	활성화
lvchange -ay|-aey	전용 모드에서 공유 논리 볼륨을 활성화하여 단일 호스트만 논리 볼륨을 활성화할 수 있습니다. 활성화가 실패하면 다른 호스트에서 논리 볼륨이 활성화된 경우 오류가 보고됩니다.
lvchange -asy	공유 모드에서 공유 논리 볼륨을 활성화하여 여러 호스트가 논리 볼륨을 동시에 활성화할 수 있습니다. 활성화에 실패하면 논리 볼륨이 다른 호스트에서 독점적으로 활성 상태인 경우 오류가 보고됩니다. 논리 유형이 스냅샷과 같은 공유 액세스를 금지하면 명령에서 오류를 보고하지 않습니다. 여러 호스트에서 동시에 사용할 수 없는 논리 볼륨 유형에는 thin, cache, raid, snapshot이 있습니다.
lvchange -an	논리 볼륨을 비활성화합니다.

13.4. 누락된 장치가 있는 논리 볼륨 활성화

***lvchange** 명령을 **--activationmode partial|degraded|complete** 옵션과 함께 사용하여 장치 누락된 LV를 활성화할 수 있는지 여부를 제어할 수 있습니다. 값은 다음과 같습니다.*

활성화 모드	meaning
complete	물리 볼륨이 없는 논리 볼륨만 활성화할 수 있습니다. 이것이 가장 제한적인 모드입니다.
Degraded	물리 볼륨이 누락된 RAID 논리 볼륨을 활성화할 수 있습니다.
부분적	물리 볼륨이 없는 논리 볼륨을 활성화할 수 있습니다. 이 옵션은 복구 또는 복구에만 사용해야 합니다.

활성화 모드 의 기본값은 /etc/lvm/lvm.conf 파일의 activationmode 설정에 따라 결정됩니다. 명령줄 옵션이 제공되지 않는 경우 사용됩니다.

추가 리소스

- ***lvmraid(7) 도움말 페이지***

14장. LVM 장치 가시성 및 사용 제한

LVM에서 스캔할 수 있는 장치를 제어하여 표시되는 장치를 LVM(Logical Volume Manager)에서 표시하고 사용할 수 있는 장치를 제한할 수 있습니다.

LVM 장치 스캔 구성을 조정하려면 /etc/lvm/lvm.conf 파일에서 LVM 장치 필터 설정을 편집합니다. lvm.conf 파일의 필터는 일련의 간단한 정규식으로 구성됩니다. 시스템은 이러한 표현식을 /dev 디렉토리의 각 장치 이름에 적용하여 감지된 각 블록 장치를 수락할지 또는 거부할지 결정합니다.

14.1. LVM 필터링을 위한 영구 식별자

/dev/sda 와 같은 기존 Linux 장치 이름은 시스템을 수정하고 재부팅하는 동안 변경될 수 있습니다. WWID(WWID), UUID(Universally Unique Identifier) 및 경로 이름과 같은 PNA(영구 이름 지정 속성)는 스토리지 장치의 고유한 특성을 기반으로 하며 하드웨어 구성 변경에 탄력적입니다. 이로 인해 시스템 재부팅 시 보다 안정적이고 예측 가능합니다.

LVM 필터링에 영구 장치 식별자를 구현하면 LVM 구성의 안정성과 안정성이 향상됩니다. 또한 장치 이름의 동적 특성과 관련된 시스템 부팅 실패의 위험을 줄입니다.

추가 리소스

- 영구 이름 지정 속성
- 로컬 디스크 이름이 영구적이지 않은 경우 lvm 필터를 구성하는 방법은 무엇입니까?

14.2. LVM 장치 필터

LVM(Logical Volume Manager) 장치 필터는 장치 이름 패턴 목록입니다. 이를 사용하여 시스템에서 장치를 평가할 수 있는 필수 기준 세트를 지정하고 LVM과 함께 사용하기 위해 이를 유효한 것으로 간주할 수 있습니다. LVM 장치 필터를 사용하면 LVM에서 사용하는 장치를 제어할 수 있습니다. 이는 실수로 데이터 손실 또는 저장 장치에 대한 무단 액세스를 방지하는 데 도움이 될 수 있습니다.

14.2.1. LVM 장치 필터 패턴 특성

LVM 장치 필터 패턴은 정규식의 형태로 되어 있습니다. 정규 표현식은 문자로 구분되며 수락을 위해 r 또는 거부 인 경우 r로 구분됩니다. 장치와 일치하는 목록의 첫 번째 정규 표현식은 LVM이 특정 장치를 수락하거나 거부(ignore)하는지 여부를 결정합니다. 그러면 LVM에서 장치의 경로와 일치하는 초기 정규식

을 목록에서 찾습니다. LVM은 이 정규 표현식을 사용하여 장치를 결과를 통해 승인해야 하는지 r 결과를 통해 거부해야하는지 여부를 결정합니다.

단일 장치에 여러 경로 이름이 있는 경우 LVM은 목록 순서에 따라 이러한 경로 이름에 액세스합니다. r 패턴 전에 하나 이상의 경로 이름이 패턴과 일치하는 경우 LVM에서 장치를 승인합니다. 그러나 모든 경로 이름이 패턴을 발견하기 전에 r 패턴과 일치하면 장치가 거부됩니다.

패턴과 일치하지 않는 경로 이름은 장치의 승인 상태에 영향을 미치지 않습니다. 장치 패턴에 해당하는 경로 이름이 없는 경우 LVM에서는 장치를 계속 승인합니다.

시스템의 각 장치에 대해 udev 규칙은 여러 개의 심볼릭 링크를 생성합니다. 디렉터리에는 /dev/disk/by-id/, /dev/disk/by-uuid/, /dev/disk/by-path/ 와 같은 심볼릭 링크가 포함되어 여러 경로 이름을 통해 시스템의 각 장치에 액세스할 수 있습니다.

필터에서 장치를 거부하려면 특정 장치와 연결된 모든 경로 이름이 해당 reject r 표현식과 일치해야 합니다. 그러나 거부할 수 있는 모든 경로 이름을 식별하는 것은 어려울 수 있습니다. 따라서 특정 경로를 구체적으로 수락하고 다른 모든 경로를 거부하는 필터를 생성하는 것이 좋습니다. 일련의 특정 표현식과 단일 r|.| 표현식이 다른 모든 항목을 거부하는 것입니다.*

필터에 특정 장치를 정의하는 동안 커널 이름 대신 해당 장치에 symlink 이름을 사용합니다. 장치의 커널 이름은 /dev/sda 와 같이 변경될 수 있지만 특정 심볼릭 링크 이름은 /dev/disk/by-id/wwn- 과 같은 변경되지 않습니다.*

기본 장치 필터는 시스템에 연결된 모든 장치를 허용합니다. 이상적인 사용자 구성 장치 필터는 하나 이상의 패턴을 허용하고 다른 모든 패턴을 거부합니다. 예를 들어 r|.| 로 끝나는 패턴 목록입니다.*

LVM 장치는 lvm.conf 파일의 devices/filter 및 devices/global_filter 구성 필드에서 LVM 장치 필터 구성을 찾을 수 있습니다. devices/filter 및 devices/global_filter 구성 필드는 동일합니다.

추가 리소스

- *lvm.conf(5) man page*

14.2.2. LVM 장치 필터 구성의 예

다음 예제에서는 LVM에서 스캔하고 나중에 사용하는 장치를 제어하는 필터 구성을 표시합니다. lvm.conf 파일에서 장치 필터를 구성하려면 LVM 장치 필터 구성 적용을참조하십시오.

참고

복사 또는 복제된 PV를 처리할 때 중복된 PV(물리 볼륨) 경고가 표시될 수 있습니다. 이 문제를 해결하기 위해 필터를 설정할 수 있습니다. 중복 PV 경고를 방지하는 Example LVM 장치 필터의 예제 필터 구성을 참조하십시오.

- 모든 장치를 검사하려면 다음을 입력합니다.

```
filter = [ "a|.*|" ]
```

- 드라이브에 미디어가 없는 경우 지연을 방지하기 위해 cdrom 장치를 제거하려면 다음을 입력하십시오.

```
filter = [ "r|^/dev/cdrom$|" ]
```

- 모든 루프 장치를 추가하고 다른 모든 장치를 제거하려면 다음을 입력합니다.

```
filter = [ "a|loop|", "r|.*|" ]
```

- 모든 루프 및 SCSI 장치를 추가하고 다른 모든 블록 장치를 제거하려면 다음을 입력합니다.

```
filter = [ "a|loop|", "a|/dev/sd.*|", "r|.*|" ]
```

- 첫 번째 SCSI 드라이브에 파티션 8만 추가하고 다른 모든 블록 장치를 제거하려면 다음을 입력합니다.

```
filter = [ "a|^/dev/sda8$|", "r|.*|" ]
```

- 모든 다중 경로 장치와 함께 WWID로 식별되는 특정 장치의 모든 파티션을 추가하려면 다음을 입력합니다.

```
filter = [ "a|/dev/disk/by-id/<disk-id>.|", "a|/dev/mapper/mpath.|", "r|.*|" ]
```

명령은 다른 블록 장치도 제거합니다.

추가 리소스

- *lvm.conf(5) man page*
- *LVM 장치 필터 구성 적용*
- *중복된 PV 경고를 방지하는 LVM 장치 필터의 예*

14.2.3. LVM 장치 필터 구성 적용

lvm.conf 구성 파일에 필터를 설정하여 LVM 스캔 장치를 제어할 수 있습니다.

사전 요구 사항

- *사용하려는 장치 필터 패턴을 준비합니다.*

절차

1. *다음 명령을 사용하여 실제로 /etc/lvm/lvm.conf 파일을 수정하지 않고 장치 필터 패턴을 테스트합니다. 다음은 필터 구성 예제를 포함합니다.*

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|*.|" ] }'
```

2. */etc/lvm/lvm.conf 파일의 구성 섹션 장치에 장치 필터 패턴을 추가합니다.*

```
filter = [ "a|/dev/emcpower.*|", "r|*.|" ]
```

3. *재부팅 시 필요한 장치만 스캔합니다.*

```
# dracut --force --verbose
```

이 명령은 재부팅 시 LVM이 필요한 장치만 스캔하도록 initramfs 파일 시스템을 다시 빌드합니다.

15장. LVM 할당 제어

기본적으로 볼륨 그룹은 일반 할당 정책을 사용합니다. 이렇게 하면 동일한 물리 볼륨에 병렬 스트라이프를 배치하지 않는 등의 공통 밀도 규칙에 따라 물리 확장 영역을 할당합니다. **Cryostat create** 명령의 **--alloc** 인수를 사용하여 다른 할당 정책(,연속적인 ,어디에서나)을 지정할 수 있습니다. 일반적으로 일반 이외의 할당 정책은 비정상적 또는 비표준 범위 할당을 지정해야 하는 특수한 경우에만 필요합니다.

15.1. 지정된 장치의 확장 영역 할당

명령줄 끝에 있는 장치 인수를 **lvcreate** 및 **lvconvert** 명령과 함께 사용하여 특정 장치에서 할당을 제한할 수 있습니다. 더 많은 제어를 위해 각 장치의 실제 범위 범위를 지정할 수 있습니다. 명령은 지정된 물리 볼륨(**PV**)을 인수로 사용하여 새 논리 볼륨(**LV**)에 대한 **Extent**만 할당합니다. 각 **PV**에서 사용 가능한 확장 영역을 실행한 다음 나열된 다음 **PV**의 확장 영역을 사용합니다. 요청된 **LV** 크기에 나열된 모든 **PV**에 공간이 충분하지 않으면 명령이 실패합니다. 이 명령은 이름이 지정된 **PV**에서만 할당합니다. **RAID LV**는 별도의 **raid** 이미지 또는 별도의 스트라이프에 순차적 **PV**를 사용합니다. 전체 **RAID** 이미지에 **PV**가 충분히 크지 않으면 결과 장치 사용을 완전히 예측할 수 없습니다.

절차

1. 볼륨 그룹(**VG**)을 생성합니다.

```
# vgcreate <vg_name> <PV> ...
```

다음과 같습니다.

- **<VG_NAME >**은 **VG**의 이름입니다.
- **& lt;PV& gt;**는 **PV**입니다.

2. **PV**를 할당하여 선형 또는 **raid**와 같은 다양한 볼륨 유형을 생성할 수 있습니다.

a. 확장 영역을 할당하여 선형 볼륨을 만듭니다.

```
# lvcreate -n <lv_name> -L <lv_size> <vg_name> [ <PV> ... ]
```

다음과 같습니다.

- *<lv_name >은 LV의 이름입니다.*

- *<lv_size >는 LV의 크기입니다. 기본 단위는 메가바이트입니다.*

- *<VG_NAME >은 VG의 이름입니다.*

- *[<PV ...>] 는 PV입니다.*

  PV 중 하나, 모두 또는 명령줄에서 none을 지정할 수 있습니다.

  - *하나의 PV를 지정하면 해당 LV의 확장 영역이 할당됩니다.*

    참고

    PV에 전체 LV에 사용 가능한 확장 영역이 충분하지 않으면 lvcreate 이 실패합니다.

  - *두 PV를 지정하면 해당 LV의 확장 영역이 해당 LV 중 하나에서 할당되거나 둘 다 조합됩니다.*

  - *PV를 지정하지 않으면 VG의 PV 중 하나 또는 VG에 있는 모든 PV의 조합에서 Extent가 할당됩니다.*

    참고

    이 경우 LVM에서 이름이 지정된 PV 또는 사용 가능한 PV를 모두 사용하지 못할 수 있습니다. 첫 번째 PV에 전체 LV에 사용 가능한 확장 영역이 충분한 경우 다른 PV가 사용되지 않을 수 있습니다. 그러나 첫 번째 PV의 할당 크기가 설정된 여유 Extent가 없는 경우 LV는 첫 번째 PV에서 부분적으로 할당되고 두 번째 PV에서 부분적으로 할당될 수 있습니다.

> *예 15.1. 하나의 PV에서 확장 영역 할당*

*이 예에서는 **lv1 Extent**가 **sda** 에서 할당됩니다.*

```
# lvcreate -n lv1 -L1G vg /dev/sda
```

*예 **15.2.** 두 **PV**의 확장 영역 할당*

*이 예에서 **lv2 Extent**는 **sda** 또는 **sdb** 또는 둘 다 조합에서 할당됩니다.*

```
# lvcreate -n lv2 L1G vg /dev/sda /dev/sdb
```

*예 **15.3. PV**를 지정하지 않고 확장 영역 할당*

*이 예에서 **lv3 Extent**는 **VG**의 **PV** 중 하나 또는 **VG**에 있는 모든 **PV**의 조합에서 할당됩니다.*

```
# lvcreate -n lv3 -L1G vg
```

또는

b. *확장 영역을 할당하여 **raid** 볼륨을 생성합니다.*

```
# lvcreate --type <segment_type> -m <mirror_images> -n <lv_name> -L <lv_size>
<vg_name> [ <PV> ... ]
```

다음과 같습니다.

- ***<segment_type >**은 지정된 세그먼트 유형입니다(예: **raid5,mirror,snapshot**).*
- ***<mirror_images >**는 지정된 수의 이미지를 사용하여 **raid1** 또는 미러링된 **LV**를 생성합니다. 예를 들어 **-m 1** 을 사용하면 두 개의 이미지가 있는 **raid1 LV**가 생성됩니다.*
- ***<lv_name >**은 **LV**의 이름입니다.*

- *<lv_size >는 LV의 크기입니다. 기본 단위는 메가바이트입니다.*
- *<VG_NAME >은 VG의 이름입니다.*
- *<[PV ...]> 는 PV입니다.*

첫 번째 RAID 이미지는 첫 번째 PV, 두 번째 PV의 두 번째 raid 이미지 등에서 할당됩니다.

> *예 15.4. 두 PV에서 raid 이미지 할당*
>
> *이 예에서 lv4 첫 번째 raid 이미지는 sda 에서 할당되고 두 번째 이미지는 sdb 에서 할당됩니다.*
>
> ```
> # lvcreate --type raid1 -m 1 -n lv4 -L1G vg /dev/sda /dev/sdb
> ```

> *예 15.5. 세 PV에서 raid 이미지 할당*
>
> *이 예에서 lv5 첫 번째 raid 이미지는 sda 에서 할당되고 두 번째 이미지는 sdb 에서 할당되고, 세 번째 이미지는 sdc 에서 할당됩니다.*
>
> ```
> # lvcreate --type raid1 -m 2 -n lv5 -L1G vg /dev/sda /dev/sdb /dev/sdc
> ```

추가 리소스

- *lvcreate(8) 매뉴얼 페이지*
- *lvconvert(8) man page*
- *lvmraid(7) 도움말 페이지*

15.2. LVM 할당 정책

LVM 작업에서 하나 이상의 논리 볼륨(LV)에 물리 확장 영역을 할당해야 하는 경우 할당은 다음과 같이 진행됩니다.

- *볼륨 그룹에서 할당되지 않은 물리 확장 영역의 전체 집합이 고려되도록 생성됩니다. 명령줄 끝에 물리 확장 영역 범위를 제공하는 경우 지정된 PV(물리 볼륨)에서 해당 범위 내에서 할당되지 않은 물리 확장 영역만 고려합니다.*

- *각 할당 정책은 가장 엄격한 정책(지속적인)부터 시작하여 --alloc 옵션을 사용하여 지정된 할당 정책으로 끝나거나 특정 LV 또는 볼륨 그룹(VG)의 기본값으로 설정됩니다. 각 정책에 대해 할당 정책에 따른 제한 사항에 따라 가능한 한 많은 공간을 채워야 하는 빈 LV 공간의 가장 낮은 숫자 논리 범위에서 작업하는 것이 좋습니다. 공간이 더 필요한 경우 LVM은 다음 정책으로 이동합니다.*

할당 정책 제한은 다음과 같습니다.

- *연속 정책을 사용하려면 LV의 첫 번째 논리 범위를 제외하고 논리 영역의 물리적 위치가 바로 앞의 논리 확장 영역의 물리적 위치에 있어야 합니다.*

  LV를 제거하거나 미러링하면 연속 할당 제한이 공간이 필요한 각 스트라이프 또는 raid 이미지에 독립적으로 적용됩니다.

- *클링 할당 정책을 사용하려면 논리 범위에 사용된 PV를 해당 LV에서 하나 이상의 논리 범위에서 이미 사용 중인 기존 LV에 추가해야 합니다.*

- *일반 할당 정책은 해당 병렬 LV 내의 동일한 오프셋에서 병렬 LV(즉, 다른 스트라이프 또는 raid 이미지)에 이미 할당된 논리 범위와 동일한 PV를 공유하는 물리적 범위를 선택하지 않습니다.*

- *할당 요청을 충족할 수 있는 사용 가능한 확장 영역이 충분하지만 일반 할당 정책에서 사용하지 않는 경우 동일한 PV에 두 개의 스트라이프를 배치하여 성능이 저하되더라도 아무나의 할당 정책이 사용됩니다.*

Cryostatchange 명령을 사용하여 할당 정책을 변경할 수 있습니다.

참고

향후 업데이트에서는 정의된 할당 정책에 따라 레이아웃 동작에 코드 변경을 가져올 수 있습니다. 예를 들어, 할당에 사용할 수 있는 동일한 수의 물리 확장 영역이 동일한 명령줄에 있는 두 개의 빈 물리 볼륨을 제공하는 경우 **LVM**은 현재 나열된 순서대로 각 물리 볼륨을 사용합니다. 향후 릴리스에서는 해당 속성을 유지 관리할 것이라는 보장이 없습니다. 특정 **LV**에 대한 특정 레이아웃이 필요한 경우 일련의 **lvcreate** 및 **lvconvert** 단계를 통해 이를 빌드하여 각 단계에 적용된 할당 정책이 레이아웃에 대한 재량에 관계없이 **LVM**을 남기지 않도록 합니다.

15.3. 물리 볼륨에서 할당 방지

pvchange 명령을 사용하여 하나 이상의 물리 볼륨의 사용 가능한 공간에 물리 확장 영역을 할당하지 못할 수 있습니다. 디스크 오류가 있거나 물리 볼륨을 제거하는 경우 이 작업이 필요할 수 있습니다.

절차

- 다음 명령을 사용하여 **device_name** 에서 물리 확장 영역 할당을 허용하지 않습니다.

```
# pvchange -x n /dev/sdk1
```

pvchange 명령의 **-xy** 인수를 사용하여 이전에 허용하지 않은 위치에서 할당을 허용할 수도 있습니다.

추가 리소스

- **pvchange(8)** 도움말 페이지

16장. 태그가 있는 LVM 오브젝트 그룹화

논리 볼륨 관리(LVM) 오브젝트에 태그를 할당하여 그룹화할 수 있습니다. 이 기능을 사용하면 그룹에서 활성화와 같은 LVM 동작의 제어를 자동화할 수 있습니다. LVM 오브젝트의 태그를 명령으로 사용할 수도 있습니다.

16.1. LVM 오브젝트 태그

LVM(Logical Volume Management) 태그는 동일한 유형의 LVM2 오브젝트를 그룹화하는 데 사용되는 단어입니다. 물리 볼륨, 볼륨 그룹, 논리 볼륨과 같은 오브젝트에 태그를 연결할 수 있으며 클러스터 구성의 호스트에 태그를 연결할 수 있습니다.

모호성을 방지하려면 각 태그 앞에 @. 각 태그는 해당 태그를 보유하는 모든 오브젝트로 대체하여 확장되며 명령줄에서 해당 위치에서 예상되는 유형입니다.

LVM 태그는 최대 1024자까지의 문자열입니다. LVM 태그는 하이픈으로 시작할 수 없습니다.

유효한 태그는 제한된 문자 범위로 구성됩니다. 허용되는 문자는 A-Z a-z 0-9 _ + . - / = ! : # &.입니다.

볼륨 그룹의 오브젝트만 태그를 지정할 수 있습니다. 물리 볼륨은 볼륨 그룹에서 제거된 경우 태그를 손실합니다. 태그는 볼륨 그룹 메타데이터의 일부로 저장되고 물리 볼륨이 제거될 때 삭제됩니다.

동일한 태그가 있는 모든 볼륨 그룹(VG), 논리 볼륨(LV) 또는 PV(물리 볼륨)에 일부 명령을 적용할 수 있습니다. 지정된 명령의 man 페이지에는 VG|Tag,LV|Tag, PV|Tag 라는 구문이 표시됩니다. VG, LV 또는 PV 이름의 태그 이름을 대체할 수 있는 경우.

16.2. LVM 태그 나열

다음 예는 LVM 태그를 나열하는 방법을 보여줍니다.

절차

- 다음 명령을 사용하여 database 태그가 있는 모든 논리 볼륨을 나열합니다.

```
# lvs @database
```

- 다음 명령을 사용하여 현재 활성화된 호스트 태그를 나열합니다.

```
# lvm tags
```

16.3. LVM 오브젝트에 태그 추가

다양한 볼륨 관리 명령과 함께 **--addtag** 옵션을 사용하여 **LVM** 오브젝트에 태그를 추가하여 그룹화할 수 있습니다.

사전 요구 사항

- **lvm2** 패키지가 설치되어 있습니다.

절차

- 기존 **PV**에 태그를 추가하려면 다음을 사용합니다.

```
# pvchange --addtag <@tag> <PV>
```

- 기존 **VG**에 태그를 추가하려면 다음을 사용합니다.

```
# vgchange --addtag <@tag> <VG>
```

- 생성 중에 **VG**에 태그를 추가하려면 다음을 사용합니다.

```
# vgcreate --addtag <@tag> <VG>
```

- 기존 **LV**에 태그를 추가하려면 다음을 사용합니다.

```
# lvchange --addtag <@tag> <LV>
```

- 생성 중에 **LV**에 태그를 추가하려면 다음을 사용합니다.

```
# lvcreate --addtag <@tag> ...
```

16.4. LVM 오브젝트에서 태그 제거

더 이상 **LVM** 오브젝트를 그룹화하지 않으려면 다양한 볼륨 관리 명령과 함께 **--deltag** 옵션을 사용하여 오브젝트에서 태그를 제거할 수 있습니다.

사전 요구 사항

- **lvm2** 패키지가 설치되어 있습니다.
- 물리 볼륨(**PV**), 볼륨 그룹(**VG**) 또는 논리 볼륨(**LV**)에 태그가 생성되었습니다.

절차

- 기존 **PV**에서 태그를 제거하려면 다음을 사용합니다.

```
# pvchange --deltag @tag PV
```

- 기존 **VG**에서 태그를 제거하려면 다음을 사용합니다.

```
# vgchange --deltag @tag VG
```

- 기존 **LV**에서 태그를 제거하려면 다음을 사용합니다.

```
# lvchange --deltag @tag LV
```

16.5. LVM 호스트 태그 정의

이 절차에서는 클러스터 구성에서 **LVM** 호스트 태그를 정의하는 방법을 설명합니다. 구성 파일에 호스트 태그를 정의할 수 있습니다.

절차

- 시스템의 호스트 이름을 사용하여 호스트 태그를 자동으로 정의하도록 **tags** 섹션에 **hosttags = 1** 을 설정합니다.

  이를 통해 모든 시스템에서 복제할 수 있는 공통 구성 파일을 사용할 수 있지만 호스트 이름에 따라 동일한 파일 사본을 보유할 수 있지만 동작은 시스템마다 다를 수 있습니다.

각 호스트 태그에 대해 추가 구성 파일이 있는 경우 해당 파일을 읽습니다. **lvm_hosttag.conf.** 해당 파일이 새 태그를 정의하는 경우 읽을 파일 목록에 추가 구성 파일이 추가됩니다.

예를 들어 구성 파일의 다음 항목은 항상 **tag1** 을 정의하고 호스트 이름이 **host1** 인 경우 **tag2** 를 정의합니다.

```
tags { tag1 { }  tag2 { host_list = ["host1"] } }
```

16.6. 태그를 사용하여 논리 볼륨 활성화 제어

이 절차에서는 특정 논리 볼륨만 해당 호스트에서 활성화해야 하는 구성 파일에서 지정하는 방법을 설명합니다.

절차

예를 들어 다음 항목은 활성화 요청(예: **KnativeServing change -ay**)에 대한 필터 역할을 하며 해당 호스트의 메타데이터에서 **database** 태그가 있는 논리 볼륨 또는 볼륨 그룹만 활성화합니다.

```
activation { volume_list = ["vg1/lvol0", "@database" ] }
```

메타데이터 태그가 해당 시스템의 호스트 태그와 일치하는 경우에만 일치하는 특수한 **@*** 입니다.

또 다른 예로 클러스터의 모든 시스템에 구성 파일에 다음 항목이 있는 상황을 고려하십시오.

```
tags { hosttags = 1 }
```

host db2 에서만 **KnativeServing 1/lvol2** 를 활성화하려면 다음을 수행하십시오.

1. 클러스터의 모든 호스트에서 **lvchange --addtag @db21/lvol2** 를 실행합니다.

2. **lvchange -ay1/lvol2** 를 실행합니다.

이 솔루션을 사용하려면 볼륨 그룹 메타데이터 내에 호스트 이름을 저장해야 합니다.

17장. LVM 문제 해결

LVM(Logical Volume Manager) 도구를 사용하여 LVM 볼륨 및 그룹의 다양한 문제를 해결할 수 있습니다.

17.1. LVM에서 진단 데이터 수집

LVM 명령이 예상대로 작동하지 않는 경우 다음과 같은 방법으로 진단을 수집할 수 있습니다.

절차

- 다음 방법을 사용하여 다양한 진단 데이터를 수집합니다.
  - LVM 명령에 -v 인수를 추가하여 명령 출력의 상세 수준을 높입니다. v의 추가를 추가하여 세부 정보를 더 늘릴 수 있습니다. 이러한 v는 최대 4개까지 허용됩니다(예: -vvv).
  - /etc/lvm/lvm.conf 구성 파일의 log 섹션에서 level 옵션의 값을 늘립니다. 이로 인해 LVM에서 시스템 로그에 세부 정보를 제공합니다.
  - 문제가 논리 볼륨 활성화와 관련된 경우 활성화 중 LVM을 활성화하도록 활성화합니다.
    i. /etc/lvm/lvm.conf 구성 파일의 log 섹션에 activation = 1 옵션을 설정합니다.
    ii. -vvv 옵션을 사용하여 LVM 명령을 실행합니다.
    iii. 명령 출력을 검사합니다.
    iv. 활성화 옵션을 0 으로 재설정합니다.

    옵션을 0 으로 재설정하지 않으면 메모리 부족 상태에서 시스템이 응답하지 않을 수 있습니다.

- 진단 목적으로 정보 덤프를 표시합니다.

```
# lvmdump
```

- 추가 시스템 정보를 표시합니다.

```
# lvs -v
```

```
# pvs --all
```

```
# dmsetup info --columns
```

- **/etc/lvm/backup/** 디렉터리에서 **LVM** 메타데이터의 마지막 백업과 **/etc/lvm/archive/** 디렉터리에 보관된 버전을 검사합니다.

- 현재 구성 정보를 확인합니다.

```
# lvmconfig
```

- **/run/lvm/hints** 캐시 파일에서 물리 볼륨이 있는 장치에 대한 레코드를 확인합니다.

추가 리소스

- **lvmdump(8) man page**

17.2. 실패한 LVM 장치에 대한 정보 표시

실패한 **LVM(Logical Volume Manager)** 볼륨에 대한 정보를 해결하면 오류 원인을 파악하는 데 도움이 될 수 있습니다. 가장 일반적인 **LVM** 볼륨 실패의 다음 예를 확인할 수 있습니다.

예 17.1. 실패한 볼륨 그룹

이 예에서는 볼륨 그룹 **myvg** 를 구성하는 장치 중 하나가 실패했습니다. 그런 다음 볼륨 그룹 유용성은 오류 유형에 따라 달라집니다. 예를 들어 **RAID** 볼륨도 관련된 경우에도 볼륨 그룹을 계속 사용할 수 있습니다. 실패한 장치에 대한 정보도 볼 수 있습니다.

```
# vgs --options +devices
```

```
 /dev/vdb1: open failed: No such device or address
 /dev/vdb1: open failed: No such device or address
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
/dev/sdb1).
  WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
devices.

VG    #PV #LV #SN Attr   VSize  VFree  Devices
myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](0)
myvg   2   2   0 wz-pn- <3.64t <3.60t [unknown](5120),/dev/vdb1(0)
```

예 17.2. 실패한 논리 볼륨

이 예에서는 장치 중 하나가 실패했습니다. 이는 볼륨 그룹의 논리 볼륨이 실패하는 이유가 될 수 있습니다. 명령 출력에는 실패한 논리 볼륨이 표시됩니다.

```
# lvs --all --options +devices

  /dev/vdb1: open failed: No such device or address
  /dev/vdb1: open failed: No such device or address
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last written to
/dev/sdb1).
  WARNING: Couldn't find all devices for LV myvg/mylv while checking used and assumed
devices.

  LV    VG  Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Devices
  mylv myvg -wi-a---p- 20.00g                                             [unknown](0)
[unknown](5120),/dev/sdc1(0)
```

예 17.3. RAID 논리 볼륨에서 실패한 이미지

다음 예제에서는 **RAID** 논리 볼륨의 이미지에 실패한 경우 **pvs** 및 **lvs** 유틸리티의 명령 출력을 보여줍니다. 논리 볼륨은 계속 사용할 수 있습니다.

```
# pvs

  Error reading device /dev/sdc1 at 0 length 4.

  Error reading device /dev/sdc1 at 4096 length 4.

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and
assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and
```

```
assumed devices.

  PV         VG        Fmt  Attr PSize    PFree
  /dev/sda2  rhel_bp-01 lvm2 a--  <464.76g    4.00m
  /dev/sdb1  myvg       lvm2 a--  <836.69g  736.68g
  /dev/sdd1  myvg       lvm2 a--  <836.69g <836.69g
  /dev/sde1  myvg       lvm2 a--  <836.69g <836.69g
  [unknown]  myvg       lvm2 a-m  <836.69g  736.68g
```

```
# lvs -a --options name,vgname,attr,size,devices myvg

  Couldn't find device with uuid b2J8oD-vdjw-tGCA-ema3-iXob-Jc6M-TC07Rn.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rimage_1 while checking used and
assumed devices.

  WARNING: Couldn't find all devices for LV myvg/my_raid1_rmeta_1 while checking used and
assumed devices.

  LV                VG   Attr       LSize   Devices
  my_raid1          myvg rwi-a-r-p- 100.00g my_raid1_rimage_0(0),my_raid1_rimage_1(0)
  [my_raid1_rimage_0] myvg iwi-aor--- 100.00g /dev/sdb1(1)
  [my_raid1_rimage_1] myvg Iwi-aor-p- 100.00g [unknown](1)
  [my_raid1_rmeta_0]  myvg ewi-aor---   4.00m /dev/sdb1(0)
  [my_raid1_rmeta_1]  myvg ewi-aor-p-   4.00m [unknown](0)
```

17.3. 볼륨 그룹에서 손실된 LVM 물리 볼륨 제거

물리 볼륨이 실패하면 볼륨 그룹의 나머지 물리 볼륨을 활성화하고 볼륨 그룹에서 해당 물리 볼륨을 사용하는 모든 논리 볼륨을 제거할 수 있습니다.

절차

1. 볼륨 그룹에서 나머지 물리 볼륨을 활성화합니다.

```
# vgchange --activate y --partial myvg
```

2. 제거될 논리 볼륨을 확인합니다.

```
# vgreduce --removemissing --test myvg
```

3. 볼륨 그룹에서 손실된 물리 볼륨을 사용하는 모든 논리 볼륨을 제거합니다.

```
# vgreduce --removemissing --force myvg
```

4.

*선택 사항: 유지하려는 논리 볼륨을 실수로 제거한 경우 **KnativeServing reduce** 작업을 취소할 수 있습니다.*

```
# vgcfgrestore myvg
```

주의

***thin** 풀을 제거하면 **LVM**에서 작업을 취소할 수 없습니다.*

17.4. 누락된 LVM 물리 볼륨의 메타데이터 찾기

*볼륨 그룹의 물리 볼륨의 메타데이터 영역이 실수로 덮어 쓰기되거나 그렇지 않으면 삭제된 경우 메타데이터 영역이 잘못되었거나 시스템이 특정 **UUID**가 있는 물리 볼륨을 찾을 수 없음을 나타내는 오류 메시지가 표시됩니다.*

이 절차에서는 누락되거나 손상된 물리 볼륨의 최신 아카이브 메타데이터를 찾습니다.

절차

1.

*물리 볼륨이 포함된 볼륨 그룹의 아카이브 메타데이터 파일을 찾습니다. 아카이브된 메타데이터 파일은 **/etc/lvm/archive/volume-group-name_backup-number.vg** 경로에 있습니다.*

```
# cat /etc/lvm/archive/myvg_00000-1248998876.vg
```

***00000-1248998876** 을 백업 번호로 바꿉니다. 볼륨 그룹 수가 가장 많은 마지막 알려진 유효한 메타데이터 파일을 선택합니다.*

2.

*물리 볼륨의 **UUID**를 찾습니다. 다음 방법 중 하나를 사용합니다.*

- *논리 볼륨을 나열합니다.*

```
# lvs --all --options +devices

  Couldn't find device with uuid 'FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk'.
```

-

- 아카이브 메타데이터 파일을 검사합니다. 볼륨 그룹 구성의 **physical_volumes** 섹션에서 **id =** 레이블이 지정된 값으로 **UUID**를 찾습니다.

- **--partial** 옵션을 사용하여 볼륨 그룹을 비활성화합니다.

```
# vgchange --activate n --partial myvg

  PARTIAL MODE. Incomplete logical volumes will be processed.
  WARNING: Couldn't find device with uuid 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-
z1lf4s.
  WARNING: VG myvg is missing PV 42B7bu-YCMp-CEVD-CmKH-2rk6-fiO9-z1lf4s (last
written to /dev/vdb1).
  0 logical volume(s) in volume group "myvg" now active
```

17.5. LVM 물리 볼륨에서 메타데이터 복원

이 절차에서는 손상되거나 새 장치로 교체된 물리 볼륨의 메타데이터를 복원합니다. 물리 볼륨의 메타데이터 영역을 다시 작성하여 물리적 볼륨에서 데이터를 복구할 수 있습니다.

주의

LVM 논리 볼륨 작동 시 이 절차를 시도하지 마십시오. 잘못된 **UUID**를 지정하면 데이터가 손실됩니다.

사전 요구 사항

- 누락된 물리 볼륨의 메타데이터를 확인했습니다. 자세한 내용은 누락된 **LVM** 물리 볼륨의 메타데이터 찾기를 참조하십시오.

절차

1. 물리 볼륨에서 메타데이터를 복원합니다.

```
# pvcreate --uuid physical-volume-uuid \
        --restorefile /etc/lvm/archive/volume-group-name_backup-number.vg \
        block-device
```

참고

명령은 **LVM** 메타데이터 영역만 덮어쓰므로 기존 데이터 영역에는 영향을 미치지 않습니다.

예 17.4. /dev/vdb1에서 물리 볼륨 복원

다음 예제에서는 다음 속성을 사용하여 **/dev/vdb1** 장치에 물리 볼륨으로 레이블을 지정합니다.

- **FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk**의 **UUID**
- **VG_00050.vg** 에 포함된 메타데이터 정보, 볼륨 그룹의 가장 최근 보관된 메타데이터입니다.

```
# pvcreate --uuid "FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk" \
        --restorefile /etc/lvm/archive/VG_00050.vg \
        /dev/vdb1

 ...
 Physical volume "/dev/vdb1" successfully created
```

2. 볼륨 그룹의 메타데이터를 복원합니다.

```
# vgcfgrestore myvg

 Restored volume group myvg
```

3. 볼륨 그룹에 논리 볼륨을 표시합니다.

```
# lvs --all --options +devices myvg
```

논리 볼륨이 현재 비활성화되어 있습니다. 예를 들면 다음과 같습니다.

```
LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
mylv myvg   -wi--- 300.00G                       /dev/vdb1 (0),/dev/vdb1(0)
mylv myvg   -wi--- 300.00G                       /dev/vdb1 (34728),/dev/vdb1(0)
```

4. 논리 볼륨의 세그먼트 유형이 **RAID**인 경우 논리 볼륨을 다시 동기화합니다.

```
# lvchange --resync myvg/mylv
```

5. 논리 볼륨을 활성화합니다.

```
# lvchange --activate y myvg/mylv
```

6. 디스크상의 **LVM** 메타데이터가 과당한 공간을 차지하는 경우, 이 절차는 물리 볼륨을 복구할 수 있습니다. 메타데이터가 메타데이터 영역보다 오래된 경우 볼륨의 데이터가 영향을 받을 수 있습니다. **fsck** 명령을 사용하여 해당 데이터를 복구할 수 있습니다.

검증 단계

- 활성 논리 볼륨을 표시합니다.

```
# lvs --all --options +devices

 LV     VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
mylv myvg   -wi--- 300.00G                             /dev/vdb1 (0),/dev/vdb1(0)
mylv myvg   -wi--- 300.00G                             /dev/vdb1 (34728),/dev/vdb1(0)
```

17.6. LVM 출력에서 오류 반올림

볼륨 그룹에서 공간 사용량을 보고하는 **LVM** 명령은 보고된 숫자를 **2** 진수 위치로 반올림하여 사람이 읽을 수 있는 출력을 제공합니다. 여기에는**display** 및 **ECDHE s** 유틸리티가 포함됩니다.

반올림된 결과 볼륨 그룹의 물리 확장 영역보다 보고된 여유 공간의 값이 더 클 수 있습니다. 보고된 사용 가능한 공간의 크기를 논리 볼륨을 생성하려고 하면 다음과 같은 오류가 발생할 수 있습니다.

```
Insufficient free extents
```

오류를 해결하려면 볼륨 그룹에서 사용 가능한 물리 확장 영역 수를 검사해야 합니다. 이는 여유 공간의 정확한 값입니다. 그런 다음 확장 영역 수를 사용하여 논리 볼륨을 성공적으로 만들 수 있습니다.

17.7. LVM 볼륨을 만들 때 반올림 오류 방지

LVM 논리 볼륨을 생성할 때 논리 볼륨의 논리 **Extent** 수를 지정하여 반올림 오류를 방지할 수 있습니

다.

절차

1. 볼륨 그룹에서 사용 가능한 물리 확장 영역의 수를 찾습니다.

```
# vgdisplay myvg
```

예 17.5. 볼륨 그룹에서 사용 가능한 확장 영역

예를 들어 다음 볼륨 그룹에는 **8780**의 사용 가능한 물리 확장 영역이 있습니다.

```
--- Volume group ---
 VG Name               myvg
 System ID
 Format                lvm2
 Metadata Areas        4
 Metadata Sequence No  6
 VG Access             read/write
[...]
Free  PE / Size       8780 / 34.30 GB
```

2. 논리 볼륨 생성. 볼륨 크기를 바이트 대신 확장 영역으로 입력합니다.

예 17.6. 확장 영역 수를 지정하여 논리 볼륨 생성

```
# lvcreate --extents 8780 --name mylv myvg
```

예 17.7. 나머지 공간을 차지하기 위해 논리 볼륨 생성

또는 볼륨 그룹에서 남은 사용 가능한 공간의 백분율을 사용하도록 논리 볼륨을 확장할 수 있습니다. 예를 들면 다음과 같습니다.

```
# lvcreate --extents 100%FREE --name mylv myvg
```

검증 단계

- 볼륨 그룹이 이제 사용하는 확장 영역 수를 확인합니다.

```
# vgs --options +vg_free_count,vg_extent_count

 VG    #PV #LV #SN  Attr   VSize   VFree  Free  #Ext
 myvg   2   1   0   wz--n- 34.30G    0    0     8780
```

17.8. LVM 메타데이터 및 디스크의 위치

LVM 헤더 및 메타데이터 영역은 다른 오프셋 및 크기에서 사용할 수 있습니다.

기본 **LVM** 디스크 헤더:

- **label_header** 및 **pv_header** 구조에 있습니다.
- 디스크의 두 번째 **512**바이트 섹터에 있습니다. **PV(**물리 볼륨**)**를 생성할 때 기본이 아닌 위치가 지정된 경우 헤더가 첫 번째 또는 세 번째 섹터에 있을 수도 있습니다.

표준 **LVM** 메타데이터 영역:

- 디스크 시작부터 **4096**바이트를 시작합니다.
- 디스크 시작부터 **1MiB**를 종료합니다.
- **mda_header** 구조를 포함하는 **512**바이트 섹터로 시작합니다.

메타데이터 텍스트 영역은 **mda_header** 섹터 후에 시작하여 메타데이터 영역의 끝으로 이동합니다. **LVM VG** 메타데이터 텍스트는 순환 방식으로 메타데이터 텍스트 영역에 작성됩니다. **mda_header** 는 텍스트 영역 내의 최신 **VG** 메타데이터의 위치를 가리킵니다.

 # pvck --dump headers /dev/sda 명령을 사용하여 디스크에서 **LVM** 헤더를 출력할 수 있습니다. 이 명령은 **label_header,pv_header,mda_header** 및 존재하는 경우 메타데이터 텍스트 위치를 출력합니다. 잘못된 필드는 **CHECK** 접두사를 사용하여 출력됩니다.

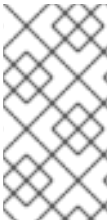
LVM 메타데이터 영역 오프셋은 **PV**를 생성한 시스템의 페이지 크기와 일치하므로 메타데이터 영역은 디스크 시작 시 **8K, 16K** 또는 **64K**를 시작할 수 있습니다.

PV를 생성할 때 더 크거나 작은 메타데이터 영역을 지정할 수 있습니다. 이 경우 메타데이터 영역은 **1MiB**가 아닌 위치에서 종료될 수 있습니다. **pv_header** 는 메타데이터 영역의 크기를 지정합니다.

PV를 생성할 때 디스크 끝에 두 번째 메타데이터 영역을 선택적으로 활성화할 수 있습니다. **pv_header** 에는 메타데이터 영역의 위치가 포함되어 있습니다.

17.9. 디스크에서 VG 메타데이터 추출

상황에 따라 디스크에서 **VG** 메타데이터를 추출하려면 다음 절차 중 하나를 선택합니다. 추출된 메타데이터를 저장하는 방법에 대한 자세한 내용은 추출된 메타데이터를 파일에 저장을 참조하십시오.



참고

복구의 경우 디스크에서 메타데이터를 추출하지 않고 **/etc/lvm/backup/** 에서 백업 파일을 사용할 수 있습니다.

절차

- 유효한 **mda_header** 에서 참조된 현재 메타데이터 텍스트를 인쇄 :

```
# pvck --dump metadata <disk>
```

예 17.8. 유효한 **mda_header**의 메타데이터 텍스트

```
# pvck --dump metadata /dev/sdb
  metadata text at 172032 crc Oxc627522f # vgname test segno 59
  ---
  <raw metadata from disk>
  ---
```

- 유효한 **mda_header** 를 찾는 방법에 따라 메타데이터 영역에 있는 모든 메타데이터 복사본의 위치를 인쇄합니다.

```
# pvck --dump metadata_all <disk>
```

예 17.9. 메타데이터 영역에 있는 메타데이터 복사본의 위치

```
# pvck --dump metadata_all /dev/sdb
```

```
  metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
```

- ***헤더가 없거나 손상된 경우 mda_header 를 사용하지 않고 메타데이터 영역에 있는 모든 메타데이터 복사본을 검색합니다.***

```
# pvck --dump metadata_search <disk>
```

예 17.10. mda_header를 사용하지 않고 메타데이터 영역의 메타데이터 복사본

```
# pvck --dump metadata_search /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 4608 length 815 crc 29fcd7ab vg test seqno 1 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
  metadata at 7168 length 1450 crc 5652ea55 vg test seqno 3 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
```

- ***dump 명령에 -v 옵션을 포함하여 메타데이터 복사본의 각 설명을 표시합니다.***

```
# pvck --dump metadata -v <disk>
```

예 17.11. 각 메타데이터 사본에서 설명 표시

```
# pvck --dump metadata -v /dev/sdb
  metadata text at 199680 crc 0x628cf243 # vgname my_vg seqno 40
  ---
my_vg {
id = "dmEbPi-gsgx-VbvS-Uaia-HczM-iu32-Rb7iOf"
seqno = 40
format = "lvm2"
status = ["RESIZEABLE", "READ", "WRITE"]
flags = []
extent_size = 8192
max_lv = 0
max_pv = 0
metadata_copies = 0

physical_volumes {

pv0 {
id = "8gn0is-Hj8p-njgs-NM19-wuL9-mcB3-kUDiOQ"
```

```
device = "/dev/sda"

device_id_type = "sys_wwid"
device_id = "naa.6001405e635dbaab125476d88030a196"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}

pv1 {
id = "E9qChJ-5ElL-HVEp-rc7d-U5Fg-fHxL-2QLyID"
device = "/dev/sdb"

device_id_type = "sys_wwid"
device_id = "naa.6001405f3f9396fddcd4012a50029a90"
status = ["ALLOCATABLE"]
flags = []
dev_size = 125829120
pe_start = 8192
pe_count = 15359
}
```

*이 파일은 복구에 사용할 수 있습니다. 첫 번째 메타데이터 영역은 기본적으로 덤프 메타데이터에 사용됩니다. 디스크 끝에 두 번째 메타데이터 영역이 있는 경우 **--settings "mda_num=2"** 옵션을 사용하여 덤프 메타데이터에 두 번째 메타데이터 영역을 대신 사용할 수 있습니다.*

17.10. 추출된 메타데이터를 파일에 저장

*복구에 덤프된 메타데이터를 사용해야 하는 경우 추출된 메타데이터를 **-f** 옵션과 **--setings** 옵션을 사용하여 파일에 저장해야 합니다.*

절차

- ***-f <filename >**이 **--dump** 메타데이터에 추가되면 원시 메타데이터가 이름이 지정된 파일에 작성됩니다. 이 파일을 사용하여 복구할 수 있습니다.*

- ***-f <filename >**이 **--dump metadata_all** 또는 **--dump metadata_search** 에 추가된 경우 모든 위치의 원시 메타데이터가 이름이 지정된 파일에 작성됩니다.*

- ***--dump metadata_all|metadata_search** 의 메타데이터 텍스트 인스턴스 하나를 저장하려면 **--settings "metadata_offset=<offset>"** 을 추가합니다. 여기서 **< offset >**은 목록 출력의 **"metadata at <offset>"**입니다.*

예 17.12. 명령 출력

```
# pvck --dump metadata_search --settings metadata_offset=5632 -f meta.txt /dev/sdb
  Searching for metadata at offset 4096 size 1044480
  metadata at 5632 length 1144 crc 50ea61c3 vg test seqno 2 id FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv
# head -2 meta.txt
test {
id = "FaCsSz-1ZZn-mTO4-Xl4i-zb6G-BYat-u53Fxv"
```

17.11. PVCREATE 및 CRYOSTATCFGRESTORE 명령을 사용하여 손상된 LVM 헤더 및 메타데이터로 디스크 복구

손상된 물리 볼륨에서 메타데이터 및 헤더를 복원하거나 새 장치로 교체할 수 있습니다. 물리 볼륨의 메타데이터 영역을 다시 작성하여 물리적 볼륨에서 데이터를 복구할 수 있습니다.

주의

이러한 명령은 매우 주의하여 사용해야 하며, 각 명령의 영향, 볼륨의 현재 레이아웃, 달성해야 하는 레이아웃 및 백업 메타데이터 파일의 콘텐츠에 대해 잘 알고 있는 경우에만 사용해야 합니다. 이러한 명령에는 데이터가 손상될 가능성이 있으므로 문제 해결에 도움이 필요한 경우 **Red Hat** 글로벌 지원 서비스에 문의하는 것이 좋습니다.

사전 요구 사항

- 누락된 물리 볼륨의 메타데이터를 확인했습니다. 자세한 내용은 누락된 **LVM** 물리 볼륨의 메타데이터 찾기를 참조하십시오.

절차

1. **pvcreate** 및 **Cryostat cfgrestore** 명령에 필요한 다음 정보를 수집합니다. **# pvs -o+uuid** 명령을 실행하여 디스크 및 **UUID**에 대한 정보를 수집할 수 있습니다.

   - **metadata-file** 은 VG의 최신 메타데이터 백업 파일의 경로입니다(예: **/etc/lvm/backup/<vg-name>**).

- *VG-name 은 PV가 손상되거나 누락된 VG의 이름입니다.*
- *이 장치에서 손상된 PV의 UUID 는 # pvs -i+uuid 명령의 출력에서 가져온 값입니다.*
- *disk 는 PV가 있어야 하는 디스크 이름입니다(예: /dev/sdb). 이 디스크가 올바른 디스크인지 확인하거나 도움을 구하십시오. 그렇지 않으면 이러한 단계를 수행하면 데이터가 손실될 수 있습니다.*

2. *디스크에서 LVM 헤더를 다시 생성합니다.*

```
# pvcreate --restorefile <metadata-file> --uuid <UUID> <disk>
```

필요한 경우 헤더가 유효한지 확인합니다.

```
# pvck --dump headers <disk>
```

3. *디스크에서 VG 메타데이터를 복원합니다.*

```
# vgcfgrestore --file <metadata-file> <vg-name>
```

필요한 경우 메타데이터가 복원되었는지 확인합니다.

```
# pvck --dump metadata <disk>
```

VG에 대한 메타데이터 백업 파일이 없는 경우 추출된 메타데이터를 파일에 저장하는 절차를 사용하여 하나를 가져올 수 있습니다.

검증

- *새 물리 볼륨이 손상되지 않고 볼륨 그룹이 올바르게 작동하는지 확인하려면 다음 명령의 출력을 확인합니다.*

```
# vgs
```

추가 리소스

- *pvck(8) 도움말 페이지*

- *물리 볼륨에서 LVM 메타데이터 백업 추출*

- *온라인에서 물리적 볼륨의 메타데이터를 복구하는 방법은 무엇입니까?*

- *볼륨 그룹을 구성하는 물리 볼륨 중 하나가 실패한 경우 Red Hat Enterprise Linux에서 볼륨 그룹을 복원하려면 어떻게 해야 합니까?*

17.12. PVCK 명령을 사용하여 손상된 LVM 헤더 및 메타데이터로 디스크 복구

이는 pvcreate 및 pvcfgrestore 명령을 사용하여 손상된 LVM 헤더 및 메타데이터가 있는 디스크를 복구하는 대신 사용할 수 있습니다. pvcreate 및 Cryostat cfgrestore 명령이 작동하지 않는 경우가 있을 수 있습니다. 이 방법은 손상된 디스크를 더 대상으로 합니다.

이 방법은 pvck --dump 에서 추출한 메타데이터 입력 파일 또는 /etc/lvm/backup 의 백업 파일을 사용합니다. 가능한 경우 동일한 VG의 다른 PV 또는 PV의 두 번째 메타데이터 영역에서 pvck --dump 에 저장된 메타데이터를 사용합니다. 자세한 내용은 추출된 메타데이터를 파일에 저장을 참조하십시오.

절차

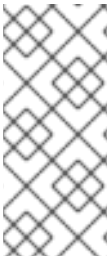
- *디스크에서 헤더 및 메타데이터를 복구합니다.*

```
# pvck --repair -f <metadata-file> <disk>
```

다음과 같습니다.

  - *<metadata-file >은 VG에 대한 최신 메타데이터를 포함하는 파일입니다. /etc/lvm/backup/ Cryostat-name 이거나 pvck --dump metadata_search 명령 출력에서 원시 메타데이터 텍스트가 포함된 파일일 수 있습니다.*

  - *<disk >는 PV가 있어야 하는 디스크 이름입니다(예: /dev/sdb). 데이터 손실을 방지하려면 가 올바른 디스크인지 확인합니다. 디스크가 올바른지 확실하지 않은 경우 Red Hat 지원팀에 문의하십시오.*



참고

메타데이터 파일이 백업 파일인 경우 **VG**에서 메타데이터를 보유하는 각 **PV**에서 **pvck --repair** 를 실행해야 합니다. 메타데이터 파일이 다른 **PV**에서 추출된 원시 메타데이터인 경우 **pvck --repair** 는 손상된 **PV**에서만 실행해야 합니다.

검증

- 새 물리 볼륨이 손상되지 않고 볼륨 그룹이 올바르게 작동하는지 확인하려면 다음 명령의 출력을 확인합니다.

```
# vgs <vgname>
# pvs <pvname>
# lvs <lvname>
```

추가 리소스

- **pvck(8)** 도움말 페이지
- 물리 볼륨에서 **LVM** 메타데이터 백업 추출.
- 온라인 물리 볼륨의 메타데이터를 복구하는 방법은 무엇입니까?.
- 볼륨 그룹을 구성하는 물리 볼륨 중 하나가 실패한 경우 **Red Hat Enterprise Linux**에서 볼륨 그룹을 복원하려면 어떻게 해야 합니까?

17.13. LVM RAID 문제 해결

LVM RAID 장치의 다양한 문제를 해결하여 데이터 오류를 수정하거나 장치를 복구하거나, 실패한 장치를 교체할 수 있습니다.

17.13.1. RAID 논리 볼륨에서 데이터 일관성 확인

LVM은 **RAID** 논리 볼륨에 대한 스크럽 지원을 제공합니다. **RAID** 스크럽은 배열의 모든 데이터 및 패리티 블록을 읽고 일치 여부를 확인하는 프로세스입니다. **lvchange --syncaction repair** 명령은 배열에

서 백그라운드 동기화 작업을 시작합니다. 다음 속성은 데이터 일관성에 대한 세부 정보를 제공합니다.

- **raid_sync_action** 필드에는 **RAID** 논리 볼륨이 수행하는 현재 동기화 작업이 표시됩니다. 다음 값 중 하나일 수 있습니다.

  idle

  모든 동기화 작업을 완료했습니다(없음).

  resync

  정리되지 않은 시스템 종료 후 배열을 초기화하거나 재동기화합니다.

  recover

  배열에서 장치를 교체합니다.

  Check

  배열 불일치를 찾습니다.

  복구

  불일치를 찾고 복구합니다.

- **raid_mismatch_count** 필드에는 검사 작업 중에 발견된 불일치 수가 표시됩니다.

- **Cpy%Sync** 필드는 동기화 작업의 진행 상황을 표시합니다.

- **lv_attr** 필드는 추가 지표를 제공합니다. 이 필드의 비트 **9**는 논리 볼륨의 상태를 표시하고 다음 지표를 지원합니다.

  m 또는 **mismatches**

  RAID 논리 볼륨에 불일치가 있음을 나타냅니다. 스크러블링 작업이 **RAID**의 일부를 감지한 후 이 문자를 볼 수 있습니다. 이 문자는 일관성이 없습니다.

  R 또는 **refresh**

  LVM에서 장치 레이블을 읽고 장치가 작동하는 것으로 간주하더라도 **RAID** 배열에서 실패한 장치를 나타냅니다. 논리 볼륨을 새로 고쳐 장치를 사용할 수 있음을 커널에 알리거나 실패한 것으로 의심되는 경우 장치를 교체합니다.

절차

1. *선택 사항: 스크럽 프로세스에서 사용하는 I/O 대역폭을 제한합니다. RAID 스크러블링 작업을 수행할 때 동기화 작업에 필요한 백그라운드 I/O는 볼륨 그룹 메타데이터 업데이트 등 LVM 장치에 대한 다른 I/O의 충돌을 줄일 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다.*

   복구 제한을 구현하여 스크럽 작업의 비율을 제어할 수 있습니다. lvchange --syncaction 명령과 함께 --maxrecoveryrate Rate[bBsSkKmMgG] 또는 --minrecoveryrate Rate[bBsSkKmMgG] 를 사용하여 복구 속도를 설정할 수 있습니다. 자세한 내용은 최소 및 최대 I/O 속도 옵션을 참조하십시오.

   Rate 값을 배열의 각 장치에 대한 초당 양으로 지정합니다. 접미사를 제공하지 않으면 옵션은 장치당 초당 kiB를 가정합니다.

2. *복구하지 않고 배열의 불일치를 표시합니다.*

```
# lvchange --syncaction check my_vg/my_lv
```

   이 명령은 배열에서 백그라운드 동기화 작업을 시작합니다.

3. *선택 사항: 커널 메시지의 var/log/syslog 파일을 확인합니다.*

4. *배열의 불일치를 수정하십시오.*

```
# lvchange --syncaction repair my_vg/my_lv
```

   이 명령은 RAID 논리 볼륨에서 실패한 장치를 복구하거나 교체합니다. 이 명령을 실행한 후 커널 메시지의 var/log/syslog 파일을 볼 수 있습니다.

검증

1. *스크럽 작업에 대한 정보를 표시합니다.*

```
# lvs -o +raid_sync_action,raid_mismatch_count my_vg/my_lv
LV    VG    Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
SyncAction Mismatches
my_lv my_vg rwi-a-r--- 500.00m                    100.00           idle       0
```

추가 리소스

- lvchange(8) 및 lvmraid(7) 도움말 페이지
- 최소 및 최대 I/O 속도 옵션

17.13.2. 논리 볼륨에서 실패한 RAID 장치 교체

RAID는 기존 LVM 미러링과 동일하지 않습니다. LVM 미러링의 경우 실패한 장치를 제거합니다. 그렇지 않으면 RAID 배열이 실패한 장치에서 계속 실행되는 동안 미러링된 논리 볼륨이 중단됩니다. RAID1 이외의 RAID 수준의 경우 장치를 제거하면 RAID6에서 RAID5로 또는 RAID4 또는 RAID0으로의 낮은 RAID 수준으로의 변환을 의미합니다.

실패한 장치를 제거하고 교체를 LVM으로 할당하는 대신 lvconvert 명령의 --repair 인수를 사용하여 RAID 논리 볼륨에서 물리 볼륨으로 사용되는 실패한 장치를 교체할 수 있습니다.

사전 요구 사항

- 볼륨 그룹에는 실패한 장치를 교체할 수 있는 충분한 여유 용량을 제공하는 물리 볼륨이 포함되어 있습니다.

  볼륨 그룹에서 사용 가능한 확장 영역이 충분한 물리 볼륨이 없는 경우 Cryostatextend 유틸리티를 사용하여 충분히 큰 새 물리 볼륨을 추가합니다.

절차

1. RAID 논리 볼륨을 확인합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
  LV              Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdc1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdc1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

2. /dev/sdc 장치가 실패한 후 RAID 논리 볼륨을 확인합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vlzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking
used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking
used and assumed devices.
  LV            Cpy%Sync Devices
  my_lv           100.00   my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         [unknown](1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          [unknown](0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

3. 실패한 장치를 교체합니다.

```
# lvconvert --repair my_vg/my_lv
  /dev/sdc: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vlzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rimage_1 while checking
used and assumed devices.
  WARNING: Couldn't find all devices for LV my_vg/my_lv_rmeta_1 while checking
used and assumed devices.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
Faulty devices in my_vg/my_lv successfully replaced.
```

4. 선택 사항: 실패한 장치를 대체하는 물리 볼륨을 수동으로 지정합니다.

```
# lvconvert --repair my_vg/my_lv replacement_pv
```

5. 교체를 사용하여 논리 볼륨을 검사합니다.

```
# lvs --all --options name,copy_percent,devices my_vg

  /dev/sdc: open failed: No such device or address
  /dev/sdc1: open failed: No such device or address
  Couldn't find device with uuid A4kRl2-vlzA-uyCb-cci7-bOod-H5tX-IzH4Ee.
  LV            Cpy%Sync Devices
  my_lv           43.79    my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdb1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdb1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

볼륨 그룹에서 실패한 장치를 제거할 때까지 **LVM** 유틸리티에서 오류가 발생한 장치를 찾을 수 없음을 계속 표시합니다.

6. 볼륨 그룹에서 실패한 장치를 제거합니다.

```
# vgreduce --removemissing my_vg
```

검증

1. 실패한 장치를 제거한 후 사용 가능한 물리 볼륨을 확인합니다.

```
# pvscan
PV /dev/sde1 VG rhel_virt-506 lvm2 [<7.00 GiB / 0 free]
PV /dev/sdb1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
PV /dev/sdd1 VG my_vg lvm2 [<60.00 GiB / 59.50 GiB free]
```

2. 실패한 장치를 교체한 후 논리 볼륨을 검사합니다.

```
# lvs --all --options name,copy_percent,devices my_vg
my_lv_rimage_0(0),my_lv_rimage_1(0),my_lv_rimage_2(0)
  [my_lv_rimage_0]         /dev/sde1(1)
  [my_lv_rimage_1]         /dev/sdb1(1)
  [my_lv_rimage_2]         /dev/sdd1(1)
  [my_lv_rmeta_0]          /dev/sde1(0)
  [my_lv_rmeta_1]          /dev/sdb1(0)
  [my_lv_rmeta_2]          /dev/sdd1(0)
```

추가 리소스

- **lvconvert(8)** 및 **Cryo statreduce(8)** 도움말 페이지

17.14. 다중 경로 LVM 장치에 대한 중복 물리 볼륨 경고 문제 해결

다중 경로 스토리지에서 **LVM**을 사용하는 경우 볼륨 그룹 또는 논리 볼륨을 나열하는 **LVM** 명령에 다음과 같은 메시지가 표시될 수 있습니다.

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/dm-5 not /dev/sdd
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowerb not /dev/sde
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sddlmab not /dev/sdf
```

*이러한 경고의 문제를 해결하여 **LVM**에서 표시하는 이유를 이해하거나 경고를 숨길 수 있습니다.*

17.14.1. PV 경고 중복의 근본 원인

***Device Mapper Multipath(DM Multipath), EMC PowerPath** 또는 **Hitachi Dynamic Link Manager(HDLM)**와 같은 다중 경로 소프트웨어가 시스템의 스토리지 장치를 관리하는 경우 특정 논리 장치(**LUN**)에 대한 각 경로가 다른 **SCSI** 장치로 등록됩니다.*

*그런 다음 다중 경로 소프트웨어는 해당 개별 경로에 매핑되는 새 장치를 생성합니다. 각 **LUN**에는 동일한 기본 데이터를 가리키는 **/dev** 디렉터리에 여러 장치 노드가 있으므로 모든 장치 노드에 동일한 **LVM** 메타데이터가 있습니다.*

*표 **17.1.** 서로 다른 다중 경로 소프트웨어의 장치 매핑 예*

다중 경로 소프트웨어	LUN에 대한 SCSI 경로	경로에 다중 경로 장치 매핑
DM Multipath	**/dev/sdb** 및 **/dev/sdc**	**/dev/mapper/mpath1** 또는 **/dev/mapper/mpatha**
EMC PowerPath		**/dev/emcpowera**
HDLM		**/dev/sddlmab**

*여러 장치 노드의 결과로 **LVM** 툴은 동일한 메타데이터를 여러 번 찾아 중복으로 보고합니다.*

17.14.2. 중복 PV 경고의 경우

***LVM**은 다음 경우 중 하나에 중복된 **PV** 경고를 표시합니다.*

동일한 장치에 대한 단일 경로

출력에 표시되는 두 장치는 모두 동일한 장치에 대한 단일 경로입니다.

*다음 예제에서는 중복 장치가 동일한 장치에 대한 단일 경로인 중복된 **PV** 경고를 보여줍니다.*

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/sdd not /dev/sdf
```

***multipath -ll** 명령을 사용하여 현재 **DM Multipath** 토폴로지를 나열하는 경우 동일한 다중 경로 맵에서 **/dev/sdd** 및 **/dev/sdf** 를 모두 찾을 수 있습니다.*

*이러한 중복 메시지는 경고일 뿐이며 **LVM** 작업이 실패했음을 의미하는 것은 아닙니다. 대신 **LVM**에서 장치 중 하나만 물리 볼륨으로 사용하고 다른 장치를 무시합니다.*

*메시지가 표시되면 **LVM**에서 잘못된 장치를 선택하거나 경고가 사용자에게 중단되는 경우 필터를 적용할 수 있습니다. 필터는 물리 볼륨에 필요한 장치만 검색하고 다중 경로 장치에 대한 기본 경로를 종료하도록 **LVM**을 구성합니다. 이로 인해 경고가 더 이상 표시되지 않습니다.*

다중 경로 맵

출력에 표시되는 두 장치는 모두 다중 경로 맵입니다.

*다음 예제에서는 두 다중 경로 맵인 두 장치에 대해 중복된 **PV** 경고를 보여줍니다. 중복 물리 볼륨은 동일한 장치에 대한 두 개의 다른 경로가 아닌 두 개의 다른 장치에 있습니다.*

```
Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/mapper/mpatha not
/dev/mapper/mpathc

Found duplicate PV GDjTZf7Y03GJHjteqOwrye2dcSCjdaUi: using /dev/emcpowera not
/dev/emcpowerh
```

*이 상황은 동일한 장치에 대한 단일 경로인 장치에 대한 중복 경고보다 심각합니다. 이러한 경고는 종종 시스템이 액세스할 수 없는 장치에 액세스한다는 것을 의미합니다(예: **LUN** 복제 또는 미러).*

*시스템에서 어떤 장치를 제거해야 하는지 명확하게 모르는 경우 이 상황을 복구할 수 없습니다. 이 문제를 해결하기 위해 **Red Hat** 기술 지원에 문의할 것을 권장합니다.*

17.14.3. 중복된 PV 경고를 방지하는 LVM 장치 필터의 예

*다음 예제에서는 단일 **LUN(Logical Unit)**에 대한 여러 스토리지 경로로 인해 발생하는 중복 물리 볼륨 경고를 방지하는 **LVM** 장치 필터를 보여줍니다.*

*모든 장치의 메타데이터를 확인하도록 **LVM(Logical Volume Manager)**의 필터를 구성할 수 있습니다. 메타데이터에는 로컬 하드 디스크 드라이브와 여기에 있는 루트 볼륨 그룹과 다중 경로 장치가 포함됩니다. **LVM**이 다중 경로 장치 자체에서 각각의 고유한 메타데이터 영역을 발견하므로 다중 경로 장치의 기본 경로(예: **/dev/sdb,/dev/sdd**)의 기본 경로를 거부하면 이러한 중복 **PV** 경고를 방지할 수 있습니다.*

- *첫 번째 하드 디스크 드라이브의 두 번째 파티션과 **DM(Device mapper) Multipath** 장치를 수*

락하고 다른 모든 항목을 거부하려면 다음을 입력합니다.

```
filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

- *모든 **HP SmartArray** 컨트롤러 및 **EMC PowerPath** 장치를 수락하려면 다음을 입력합니다.*

```
filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|" ]
```

- *첫 번째 **IDE** 드라이브 및 다중 경로 장치의 파티션을 수락하려면 다음을 입력합니다.*

```
filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

추가 리소스

- ***LVM** 장치 필터 구성의 예*

17.14.4. 추가 리소스

- ***LVM** 장치 가시성 및 사용 제한*
- ***LVM** 장치 필터*