Red Hat Enterprise Linux 8

논리 볼륨 구성 및 관리

LVM 논리 볼륨 구성 및 관리 가이드

Last Updated: 2022-08-15

Red Hat Enterprise Linux 8 논리 볼륨 구성 및 관리

LVM 논리 볼륨 구성 및 관리 가이드

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

이 설명서 컬렉션은 Red Hat Enterprise Linux 8에서 LVM 논리 볼륨을 관리하는 방법에 대한 지침을 제공합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체 6

RED HAT 문서에 관한 피드백 제공 7

1장. 논리 볼륨 관리 개요 8
1.1. LVM 아키텍처 8
1.2. LVM의 이점 9

2장. RHEL 시스템 역할을 사용하여 로컬 스토리지 관리 11
2.1. 스토리지 역할 소개 11
2.2. 스토리지 시스템 역할에서 스토리지 장치를 식별하는 매개변수 11
2.3. 예제 ANSIBLE PLAYBOOK 블록 장치에 XFS 파일 시스템을 생성 12
2.4. 파일 시스템을 영구적으로 마운트하는 ANSIBLE 플레이북의 예 12
2.5. 논리 볼륨을 관리하는 ANSIBLE 플레이북 예 13
2.6. 온라인 블록 삭제 활성화를 위한 ANSIBLE PLAYBOOK 예 14
2.7. 예시 ANSIBLE PLAYBOOK: EXT4 파일 시스템을 생성 및 마운트 14
2.8. EXT3 파일 시스템을 생성하고 마운트하는 예제 ANSIBLE PLAYBOOK 15
2.9. 스토리지 RHEL 시스템 역할을 사용하여 기존 EXT4 또는 EXT3 파일 시스템의 크기를 조정하는 ANSIBLE 플레이북의 예 15
2.10. 스토리지 RHEL 시스템 역할을 사용하여 LVM의 기존 파일 시스템의 크기를 조정하는 ANSIBLE 플레이북 예 17
2.11. 스토리지 RHEL 시스템 역할을 사용하여 스왑 파티션을 생성하는 ANSIBLE 플레이북 예 18
2.12. 스토리지 시스템 역할을 사용하여 RAID 볼륨 구성 18
2.13. 스토리지 시스템 역할을 사용하여 RAID로 LVM 풀 구성 19
2.14. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 VDO 볼륨을 압축하고 중복하기 위한 ANSIBLE 플레이북 예 20
2.15. 스토리지 시스템 역할을 사용하여 LUKS 암호화된 볼륨 생성 21
2.16. 스토리지 RHEL 시스템 역할을 사용하여 풀 볼륨 크기를 백분율로 표시하는 ANSIBLE 플레이북의 예 22
2.17. 추가 리소스 23

3장. LVM 물리 볼륨 관리 24
3.1. 물리 볼륨 개요 24
3.2. 디스크의 여러 파티션 25
3.3. LVM 물리 볼륨 생성 26
3.4. LVM 물리 볼륨 제거 28
3.5. 추가 리소스 28

4장. LVM 볼륨 그룹 관리 29
4.1. LVM 볼륨 그룹 생성 29
4.2. LVM 볼륨 그룹 결합 30
4.3. 볼륨 그룹에서 물리 볼륨 제거 31
4.4. LVM 볼륨 그룹 분할 32
4.5. LVM 볼륨 그룹 이름 34
4.6. 볼륨 그룹을 다른 시스템으로 이동 34
4.7. LVM 볼륨 그룹 제거 35

5장. LVM 논리 볼륨 관리 37
5.1. 논리 볼륨 개요 37
5.2. CLI 명령 사용 38
명령줄 인수에서 단위 지정 38
볼륨 그룹 및 논리 볼륨 지정 38
출력 세부 정보 표시 증가 39
LVM CLI 명령에 대한 도움말 표시 40

5.3. LVM 논리 볼륨 생성 40
5.4. RAID0(STRIPED) 논리 볼륨 생성 42
5.5. LVM 논리 볼륨 이름 44
5.6. 논리 볼륨에서 디스크 제거 45
5.7. LVM 논리 볼륨 제거 47
5.8. 영구 장치 번호 구성 48
5.9. LVM 확장 영역 크기 지정 48
5.10. RHEL 시스템 역할을 사용하여 LVM 논리 볼륨 관리 48
5.10.1. 논리 볼륨을 관리하는 Ansible 플레이북 예 49
5.10.2. 추가 리소스 50
5.11. LVM 볼륨 그룹 제거 50

6장. 논리 볼륨의 크기 수정 52
6.1. 논리 볼륨 및 파일 시스템 증가 52
6.2. 논리 볼륨 축소 54
6.3. 스트라이핑된 논리 볼륨 확장 56

7장. LVM에 대한 사용자 정의 보고 59
7.1. LVM 디스플레이 형식 제어 59
7.2. LVM 오브젝트 필드 표시 61
7.3. LVM 보고서 정렬 70
7.4. LVM 보고서 디스플레이의 단위 지정 71
7.5. LVM 명령 출력을 JSON 형식으로 표시 72
7.6. LVM 명령 로그 표시 73

8장. RAID 논리 볼륨 구성 75
8.1. RAID 논리 볼륨 75
8.2. RAID 수준 및 선형 지원 76
8.3. LVM RAID 세그먼트 유형 78
8.4. RAID 논리 볼륨 생성 79
8.5. RAID0(STRIPED) 논리 볼륨 생성 80
8.6. RAID LV에서 DM 무결성 사용 82
8.6.1. 소프트 데이터 손상 83
8.6.2. DM 무결성으로 RAID LV 생성 84
8.6.3. 기존 RAID LV에 DM 무결성 추가 85
8.6.4. RAID LV에서 무결성 제거 86
8.6.5. DM 무결성 정보 보기 86
8.6.6. 추가 리소스 88
8.7. RAID 볼륨이 초기화되는 속도 제어 89
8.8. 선형 장치를 RAID 장치로 변환 89
8.9. LVM RAID1 논리 볼륨을 LVM 선형 논리 볼륨으로 변환 90
8.10. 미러링된 LVM 장치를 RAID1 장치로 변환 91
8.11. RAID 논리 볼륨 크기 조정 92
8.12. 기존 RAID1 장치의 이미지 수 변경 92
8.13. RAID 이미지를 별도의 논리 볼륨으로 분할 95
8.14. RAID 이미지 분할 및 병합 96
8.15. RAID 오류 정책 설정 98
8.15.1. 할당 RAID 오류 정책 98
8.15.2. 경고 RAID 오류 정책 99
8.16. 논리 볼륨에서 RAID 장치 교체 100
8.16.1. 실패하지 않은 RAID 장치 교체 100
8.16.2. LVM RAID에서 실패한 장치 104
8.16.3. 논리 볼륨에서 실패한 RAID 장치 복구 104
8.16.4. 논리 볼륨에서 실패한 RAID 장치 교체 104

8.17. RAID 논리 볼륨(RAID 스크럽링)에서 데이터 일관성 확인 106
8.18. RAID 수준 변환(RAID 사용) 109
8.19. RAID 볼륨의 속성 변경 (RAID RESHAPE) 109
8.20. RAID1 논리 볼륨에서 I/O 작업 제어 109
8.21. RAID 논리 볼륨에서 영역 크기 변경 110

9장. 논리 볼륨의 스냅샷 111
9.1. 스냅샷 볼륨 개요 111
9.2. 원래 볼륨의 스냅샷 생성 112
9.3. 원래 볼륨에 스냅샷 병합 114

10장. 씬 프로비저닝된 볼륨(볼륨 내) 생성 및 관리 116
10.1. 씬 프로비저닝 개요 116
10.2. 씬 프로비저닝된 논리 볼륨 생성 118
10.3. 씬 프로비저닝된 스냅샷 볼륨 122
10.4. 씬 프로비저닝된 스냅샷 볼륨 생성 123

11장. 캐싱을 활성화하여 논리 볼륨 성능 개선 127
11.1. LVM의 캐싱 방법 127
11.2. LVM 캐싱 구성 요소 127
11.3. 논리 볼륨에 대한 DM-CACHE 캐싱 활성화 128
11.4. 논리 볼륨의 CACHEPOOL을 사용하여 DM-CACHE 캐싱 활성화 130
11.5. 논리 볼륨에 대한 DM-WRITECACHE 캐싱 활성화 132
11.6. 논리 볼륨의 캐싱 비활성화 134

12장. 논리 볼륨 활성화 136
12.1. 논리 볼륨의 자동 비활성화 제어 136
12.2. 논리 볼륨 활성화 제어 137
12.3. 공유 논리 볼륨 활성화 138
12.4. 누락된 장치가 있는 논리 볼륨 활성화 139

13장. LVM 장치 스캔 제어 140
13.1. LVM 장치 필터 140
13.2. LVM 장치 필터 구성의 예 140
13.3. LVM 장치 필터 구성 적용 141

14장. 논리 볼륨 상단에 LVM 물리 볼륨 계층화 143

15장. LVM 할당 제어 144
15.1. LVM 할당 정책 144
15.2. 물리 볼륨에서 할당 방지 145
15.3. CLING 할당 정책을 사용하여 논리 볼륨 확장 146
15.4. 태그를 사용하여 LVM RAID 오브젝트 간 차별화 147

16장. 태그가 있는 LVM 오브젝트 그룹화 149
16.1. LVM 오브젝트 태그 149
16.2. LVM 태그 나열 149
16.3. LVM 오브젝트 태그 추가 150
16.4. LVM 오브젝트 태그 제거 150
16.5. LVM 호스트 태그 정의 151
16.6. 태그를 사용하여 논리 볼륨 활성화 제어 151

17장. LVM 문제 해결 153
17.1. LVM에서 진단 데이터 수집 153
17.2. 실패한 LVM 장치에 대한 정보 표시 154
17.3. 볼륨 그룹에서 손실된 LVM 물리 볼륨 제거 156

17.4. 누락된 LVM 물리 볼륨의 메타데이터 찾기 157
17.5. LVM 물리 볼륨에서 메타데이터 복원 158
17.6. LVM 출력에서 오류 반올림 160
17.7. LVM 볼륨을 만들 때 반올림 오류 방지 160
17.8. LVM RAID 문제 해결 162
17.8.1. RAID 논리 볼륨(RAID 스크럽링)에서 데이터 일관성 확인 162
17.8.2. LVM RAID에서 실패한 장치 164
17.8.3. 논리 볼륨에서 실패한 RAID 장치 복구 164
17.8.4. 논리 볼륨에서 실패한 RAID 장치 교체 165
17.9. 다중 경로 LVM 장치에 대한 중복 물리 볼륨 경고 문제 해결 167
17.9.1. PV 경고 중복의 근본 원인 167
17.9.2. 중복 PV 경고의 경우 168
17.9.3. LVM 장치 필터 169
17.9.4. 중복된 PV 경고를 방지하는 LVM 장치 필터의 예 169
17.9.5. LVM 장치 필터 구성 적용 170
17.9.6. 추가 리소스 171

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 어떻게 이를 개선할 수 있는지 알려 주십시오.

- 특정 문구에 대한 간단한 의견 작성 방법은 다음과 같습니다.
  1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **피드백** 버튼이 있는지 확인합니다.
  2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
  3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
  4. 표시된 지침을 따릅니다.
- Bugzilla를 통해 피드백을 제출하려면 새 티켓을 생성합니다.
  1. Bugzilla 웹 사이트로 이동하십시오.
  2. 구성 요소로 **문서**를 사용합니다.
  3. **설명** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
  4. **버그 제출**을 클릭합니다.

1장. 논리 볼륨 관리 개요

LVM(Logical Volume Management)은 논리 스토리지 볼륨을 생성하는 데 도움이 되는 물리 스토리지에 대한 추상화 계층을 생성합니다. 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

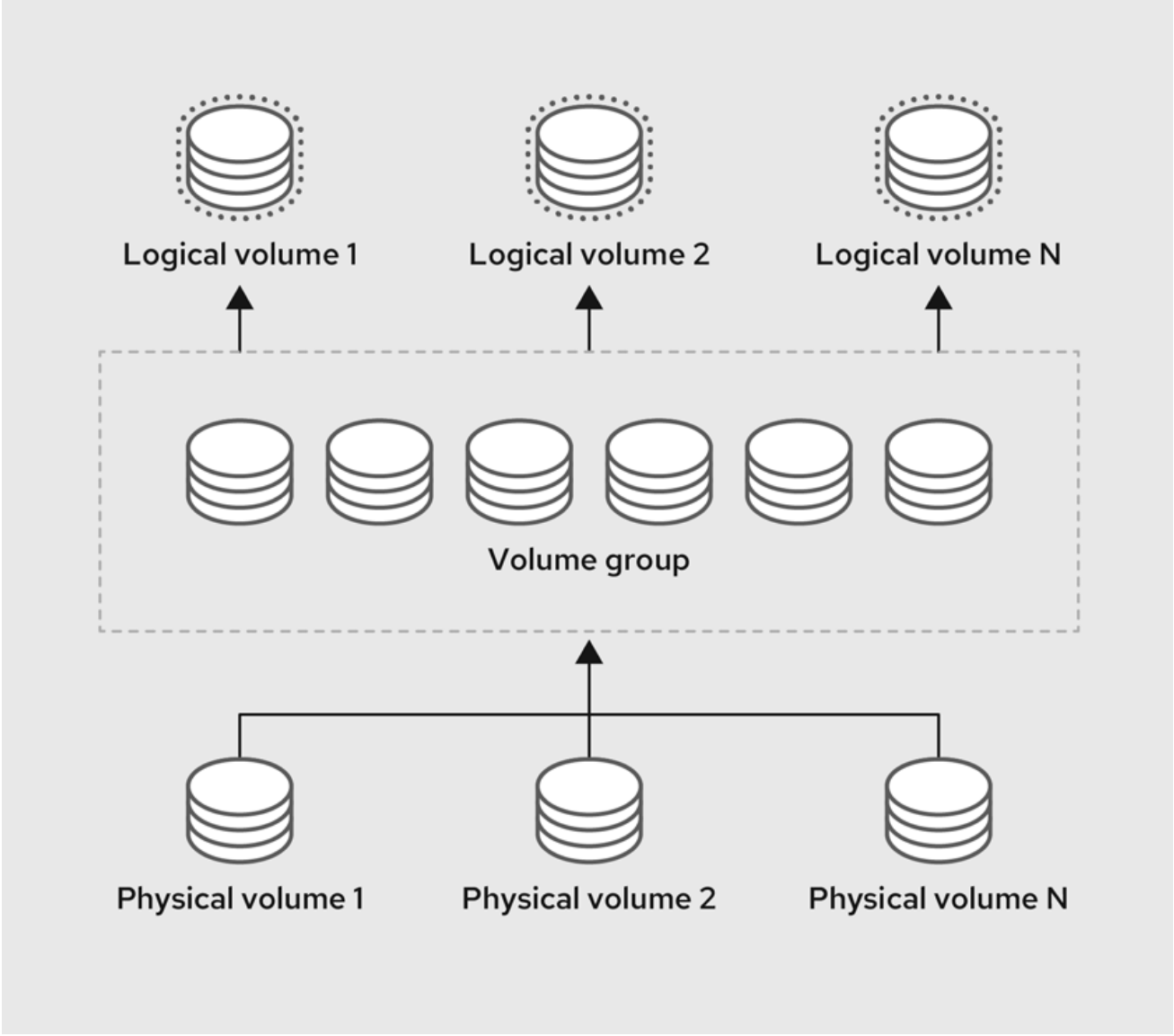
볼륨 그룹(VG)은 논리 볼륨을 할당할 수 있는 디스크 공간 풀을 생성하는 PV(물리 볼륨) 컬렉션입니다. 자세한 내용은 LVM 볼륨 그룹 관리를 참조하십시오.

논리 볼륨

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

다음 다이어그램은 LVM 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 장치 및 파티션을 단일 논리 볼륨으로 집계할 수 있습니다. 이 기능을 통해 파일 시스템은 마치 하나의 큰 단일 장치인 것처럼 여러 장치에서 확장할 수 있습니다.

크기 조정 가능한 스토리지 볼륨

기본 장치를 다시 포맷하고 다시 분할하지 않고 간단한 소프트웨어 명령으로 논리 볼륨을 확장하거나 논리 볼륨을 줄일 수 있습니다.

온라인 데이터 재배치

최신의 빠르고 탄력적인 스토리지 하위 시스템을 배포하려면 시스템이 활성화된 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.

편리한 장치 이름 지정

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

제거된 볼륨

두 개 이상의 장치에서 데이터를 스트라이프하는 논리 볼륨을 생성할 수 있습니다. 이로 인해 처리량이 크게 증가할 수 있습니다.

RAID 볼륨

논리 볼륨은 데이터에 RAID를 구성하는 편리한 방법을 제공합니다. 이를 통해 장치 장애를 보호하고 성능이 향상됩니다.

볼륨 스냅샷

일관된 백업을 위해 논리 볼륨의 특정 시점 복사본인 스냅샷을 찍거나 실제 데이터에 영향을 주지 않고 변경의 효과를 테스트할 수 있습니다.

씬 볼륨

논리 볼륨은 씬 프로비저닝할 수 있습니다. 그러면 사용 가능한 물리 공간보다 큰 논리 볼륨을 생성할 수 있습니다.

볼륨 캐시

캐시 논리 볼륨은 SSD 드라이브와 같은 빠른 블록 장치를 사용하여 더 크고 느린 블록 장치의 성능을 향상시킵니다.

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 Storage 역할을 사용할 수 있습니다.

Storage 역할을 사용하면 여러 시스템에서 디스크 및 논리 볼륨에서 파일 시스템 관리를 자동화할 수 있으며 RHEL 7.7부터 시작하는 모든 RHEL 버전에서 사용할 수 있습니다.

RHEL 시스템 역할 및 해당 역할을 적용하는 방법에 대한 자세한 내용은 RHEL 시스템 역할 소개를 참조하십시오.

2.1. 스토리지 역할 소개

스토리지 역할은 다음을 관리할 수 있습니다.

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

Storage 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
- RAID를 사용하여 LVM 풀 생성
- RAID를 사용하여 LVM 풀 제거

2.2. 스토리지 시스템 역할에서 스토리지 장치를 식별하는 매개변수

스토리지 역할 구성은 다음 변수에서 나열하는 파일 시스템, 볼륨 및 풀에만 영향을 미칩니다.

storage_volumes

관리할 모든 파티션되지 않은 디스크의 파일 시스템 목록입니다.
파티션은 현재 지원되지 않습니다.

storage_pools

관리할 풀 목록입니다.

현재 지원되는 유일한 풀 유형은 LVM입니다. LVM을 사용하면 풀이 볼륨 그룹(VG)을 나타냅니다. 각 풀에는 역할에서 관리할 볼륨 목록이 있습니다. LVM을 사용하면 각 볼륨이 파일 시스템의 논리 볼륨(LV)에 해당합니다.

2.3. 예제 ANSIBLE PLAYBOOK 블록 장치에 XFS 파일 시스템을 생성

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 기본 매개변수를 사용하여 블록 장치에 XFS 파일 시스템을 생성하는 Storage 역할을 적용합니다.

주의

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

예 2.1. /dev/sdb에서 XFS를 생성하는 플레이북

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
  roles:
    - rhel-system-roles.storage
```

- 볼륨 이름(예의 ***barefs***)은 현재 임의로 사용할 수 있습니다. Storage 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 Example Ansible Playbook to manage logical volumes 에서 참조하십시오.
  LV 장치의 경로를 제공하지 마십시오.

추가 리소스

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

2.4. 파일 시스템을 영구적으로 마운트하는 ANSIBLE 플레이북의 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 Storage 역할을 적용하여 XFS 파일 시스템을 즉시 영구적으로 마운트합니다.

예 2.2. /dev/sdb에 파일 시스템을 /mnt/data에 마운트하는 플레이북

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - sdb
        fs_type: xfs
        mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

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

추가 리소스

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

2.5. 논리 볼륨을 관리하는 ANSIBLE 플레이북 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 스토리지 역할을 적용하여 볼륨 그룹에 LVM 논리 볼륨을 만듭니다.

예 2.3. myvg 볼륨 그룹에 mylv 논리 볼륨을 생성하는 플레이북

```
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt
  roles:
    - rhel-system-roles.storage
```

- **myvg** 볼륨 그룹은 다음 디스크로 구성됩니다.
  - **/dev/sda**
  - **/dev/sdb**
  - **/dev/sdc**

- **myvg** 볼륨 그룹이 이미 있는 경우 Playbook은 볼륨 그룹에 논리 볼륨을 추가합니다.
- **myvg** 볼륨 그룹이 없으면 플레이북에서 해당 그룹을 생성합니다.
- 이 플레이북은 **mylv** 논리 볼륨에 Ext4 파일 시스템을 생성하고 **/mnt** 에 파일 시스템을 영구적으로 마운트합니다.

추가 리소스

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

2.6. 온라인 블록 삭제 활성화를 위한 ANSIBLE PLAYBOOK 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 온라인 블록 삭제가 활성화된 XFS 파일 시스템을 마운트하는 Storage 역할을 적용합니다.

예 2.4. /mnt/data/에서 온라인 블록 삭제를 활성화하는 플레이북

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

추가 리소스

- 파일 시스템을 영구적으로 마운트하는 Ansible 플레이북의 예
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 파일.

2.7. 예시 ANSIBLE PLAYBOOK: EXT4 파일 시스템을 생성 및 마운트

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 Ext4 파일 시스템을 만들고 마운트하기 위해 Storage 역할을 적용합니다.

예 2.5. /dev/sdb에 Ext4를 생성하고 /mnt/data에 마운트하는 플레이북

```
---
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
```

```
      disks:
        - sdb
      fs_type: ext4
      fs_label: label-name
      mount_point: /mnt/data
  roles:
    - rhel-system-roles.storage
```

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

추가 리소스

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

2.8. EXT3 파일 시스템을 생성하고 마운트하는 예제 ANSIBLE PLAYBOOK

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 Ext3 파일 시스템을 만들고 마운트하기 위해 Storage 역할을 적용합니다.

예 2.6. /dev/sdb 에 Ext3를 생성하여/mnt/data에 마운트하는 플레이북

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

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

추가 리소스

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

2.9. 스토리지 RHEL 시스템 역할을 사용하여 기존 EXT4 또는 EXT3 파일 시스템의 크기를 조정하는 ANSIBLE 플레이북의 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 Playbook은 블록 장치의 기존 Ext4 또는 Ext3 파일 시스템의 크기를 조정하는 Storage 역할을 적용합니다.

예 2.7. 디스크에 단일 볼륨을 설정하는 플레이북

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
  size: 12 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- 이전 예제의 볼륨이 이미 있는 경우 볼륨 크기를 조정하려면 매개 변수 **크기에** 다른 값을 사용하여 동일한 플레이북을 실행해야 합니다. 예를 들면 다음과 같습니다.

예 2.8. /dev/sdb에서 ext4 의 크기를 조정하는 플레이북

```
---
- name: Create a disk device mounted on /opt/barefs
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
          - /dev/sdb
  size: 10 GiB
        fs_type: ext4
        mount_point: /opt/barefs
  roles:
    - rhel-system-roles.storage
```

- 볼륨 이름(예의barefs)은 현재 임의로입니다. Storage 역할은 disks: 속성 아래에 나열된 디스크 장치에서 볼륨을 식별합니다.

참고

다른 파일 시스템의 작업 **크기** 조정을 사용하면 작업 중인 장치의 데이터가 손상될 수 있습니다.

추가 리소스

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

2.10. 스토리지 RHEL 시스템 역할을 사용하여 LVM의 기존 파일 시스템의 크기를 조정하는 ANSIBLE 플레이북 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 스토리지 RHEL 시스템 역할을 적용하여 파일 시스템으로 LVM 논리 볼륨의 크기를 조정합니다.

주의

다른 파일 시스템의 작업 **크기** 조정을 사용하면 작업 중인 장치의 데이터가 손상될 수 있습니다.

예 2.9. myvg 볼륨 그룹의 기존 mylv1 및 myvl2 논리 볼륨의 크기를 조정하는 플레이북

```
---

- hosts: all
  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

- name: Create LVM pool over three disks
  incude_role:
    name: rhel-system-roles.storage
```

- 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.

추가 리소스

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

2.11. 스토리지 RHEL 시스템 역할을 사용하여 스왑 파티션을 생성하는 ANSIBLE 플레이북 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 Storage 역할을 적용하여 스왑 파티션을 생성하거나, 스왑 파티션이 없는 경우, 이미 존재하는 경우 기본 매개 변수를 사용하여 블록 장치에 수정합니다.

예 2.10. /dev/sdb에서 기존 XFS를 생성하거나 수정하는 플레이북

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

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

추가 리소스

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

2.12. 스토리지 시스템 역할을 사용하여 RAID 볼륨 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform을 사용하여 RHEL에서 RAID 볼륨을 구성할 수 있습니다. 이 섹션에서는 요구 사항에 맞게 RAID 볼륨을 구성하는 데 사용 가능한 매개 변수를 사용하여 Ansible 플레이북을 설정하는 방법을 배웁니다.

사전 요구 사항

- Ansible Core 패키지는 제어 시스템에 설치됩니다.
- 플레이북을 실행할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.
- 스토리지 시스템 역할을 사용하여 RAID 볼륨을 배포하려는 시스템을 자세히 설명하는 인벤토리 파일이 있습니다.

절차

1. 다음 내용으로 새 ***playbook.yml*** 파일을 생성합니다.

```
- hosts: all
  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
roles:
  - name: rhel-system-roles.storage
```

주의

예를 들어 시스템에 새 디스크를 추가하는 경우와 같이 장치 이름은 특정 상황에서 변경될 수 있습니다. 따라서 데이터 손실을 방지하기 위해 플레이북에서 특정 디스크 이름을 사용하지 않는 것이 좋습니다.

2. 선택 사항: 플레이북 구문을 확인합니다.

```
# ansible-playbook --syntax-check playbook.yml
```

3. 인벤토리 파일에서 플레이북을 실행합니다.

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

추가 리소스

- RAID 관리.
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 파일.

2.13. 스토리지 시스템 역할을 사용하여 RAID로 LVM 풀 구성

스토리지 시스템 역할을 사용하면 Red Hat Ansible Automation Platform을 사용하여 RHEL에서 RAID를 사용하여 LVM 풀을 구성할 수 있습니다. 이 섹션에서는 사용 가능한 매개 변수를 사용하여 Ansible 플레이북을 설정하여 RAID로 LVM 풀을 구성하는 방법을 배웁니다.

사전 요구 사항

- Ansible Core 패키지는 제어 시스템에 설치됩니다.
- 플레이북을 실행할 시스템에 **rhel-system-roles** 패키지가 설치되어 있습니다.
- 스토리지 시스템 역할을 사용하여 RAID로 LVM 풀을 구성할 시스템을 자세히 설명하는 인벤토리 파일이 있습니다.

절차

1. 다음 내용으로 새 ***playbook.yml*** 파일을 생성합니다.

```
- hosts: all
  vars:
    storage_safe_mode: false
    storage_pools:
      - name: my_pool
        type: lvm
        disks: [sdh, sdi]
        raid_level: raid1
        volumes:
          - name: my_pool
            size: "1 GiB"
            mount_point: "/mnt/app/shared"
            fs_type: xfs
            state: present
  roles:
    - name: rhel-system-roles.storage
```

참고

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

2. 선택 사항: 플레이북 구문을 확인합니다.

```
# ansible-playbook --syntax-check playbook.yml
```

3. 인벤토리 파일에서 플레이북을 실행합니다.

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

추가 리소스

- RAID 관리.
- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** 파일.

2.14. 스토리지 RHEL 시스템 역할을 사용하여 LVM에서 VDO 볼륨을 압축하고 중복하기 위한 ANSIBLE 플레이북 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 플레이북은 스토리지 RHEL 시스템 역할을 적용하여 VDO(가상 데이터 최적화 도구) 볼륨을 사용하여 압축 및 중복 제거를 LVM(Logical Manager 볼륨)에 적용할 수 있습니다.

예 2.11. myvg 볼륨 그룹에 mylv1 LVM VDO 볼륨을 생성하는 플레이북

```
---
- name: Create LVM VDO volume under volume group 'myvg'
  hosts: all
  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 예에서 **압축** 및 **중복 제거** 풀은 VDO가 사용되도록 지정하는 true로 설정됩니다. 다음은 이러한 매개변수를 사용하는 방법에 대해 설명합니다.

- **중복** 제거는 스토리지 볼륨에 저장된 중복된 데이터를 중복하는 데 사용됩니다.
- 압축은 스토리지 볼륨에 저장된 데이터를 압축하는 데 사용되며 더 많은 스토리지 용량이 생성됩니다.
- pxe_pool_size는 볼륨에서 사용하는 실제 크기를 지정합니다. VDO 볼륨의 가상 크기는 **size** 매개변수로 설정됩니다. 알림: LVM VDO의 스토리지 역할 때문에 풀당 하나의 볼륨만 압축 및 중복 제거를 사용할 수 있습니다.

2.15. 스토리지 시스템 역할을 사용하여 LUKS 암호화된 볼륨 생성

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

사전 요구 사항

- Policies 시스템 역할로 구성하려는 시스템인 하나 이상의 *관리형* 노드에 대한 액세스 및 권한.
- Red Hat Ansible Core가 다른 시스템을 구성하는 시스템인 제어 노드에 대한 액세스 및 권한.
  제어 노드에서 다음을 수행합니다.
  - **ansible-core** 및 **rhel-system-roles** 패키지가 설치됩니다.

중요

RHEL 8.0-8.5는 Ansible 기반 자동화를 위해 Ansible Engine 2.9가 포함된 별도의 Ansible 리포지토리에 대한 액세스를 제공했습니다. Ansible Engine에는 ansible , **ansible-playbook**, **docker** 및 **podman** 과 같은 커넥터, 여러 플러그인 및 모듈과 같은 명령줄 유틸리티가 포함되어 있습니다. Ansible Engine을 확보하고 설치하는 방법에 대한 자세한 내용은 Red Hat Ansible Engine 지식베이스를 다운로드하고 설치하는 방법 문서를 참조하십시오.

RHEL 8.6 및 9.0에서는 Ansible 명령줄 유틸리티, 명령 및 소규모의 기본 제공 Ansible 플러그인 세트가 포함된 Ansible Core(**ansible-core** 패키지로 제공)를 도입했습니다. RHEL은 AppStream 리포지토리를 통해 이 패키지를 제공하며 제한된 지원 범위를 제공합니다. 자세한 내용은 RHEL 9 및 RHEL 8.6 이상 AppStream 리포지토리 지식 베이스에 포함된 Ansible Core 패키지에 대한 지원 범위를 참조하십시오.

- 관리 노드를 나열하는 인벤토리 파일.

절차

절차

1. 다음 내용으로 새 ***playbook.yml*** 파일을 생성합니다.

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

2. 선택 사항: 플레이북 구문을 확인합니다.

```
# ansible-playbook --syntax-check playbook.yml
```

3. 인벤토리 파일에서 플레이북을 실행합니다.

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

추가 리소스

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

2.16. 스토리지 RHEL 시스템 역할을 사용하여 풀 볼륨 크기를 백분율로 표시하는 ANSIBLE 플레이북의 예

이 섹션에서는 예제 Ansible 플레이북을 제공합니다. 이 Playbook은 스토리지 시스템 역할을 적용하여 풀의 총 크기의 백분율로 LVM(Logical Manager 볼륨) 볼륨 크기를 표시할 수 있도록 합니다.

예 2.12. 볼륨 크기를 풀의 총 크기 백분율로 표시하는 플레이북

```
---
- name: Express volume sizes as a percentage of the pool's total size
  hosts: all
  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%". 또한 LVM 볼륨의 크기를 사람이 읽을 수 있는 파일 시스템의 크기(예: "10g" 또는 "50GiB")에서 풀 크기의 백분율로 지정할 수도 있습니다.

2.17. 추가 리소스

- **/usr/share/doc/rhel-system-roles/storage/**
- **/usr/share/ansible/roles/rhel-system-roles.storage/**

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. idrac 디스크 파티션의 경우,workspace 또는 **cfdisk** 명령 또는 이와 동등한 명령을 사용하여 파티션 id를 0x8e로 설정해야 합니다. **물리 볼륨에 전체 디스크 장치를 사용하는 경우 디스크에 파티션 테이블이 없어야 합니다. 기존 파티션 테이블을 삭제해야 합니다. 그러면 해당 디스크의 모든 데이터가 효과적으로 삭제됩니다. 와제 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 장치에 대한 올바른 식별 및 장치 순서를 제공합니다. 레이블이 지정되지 않은 LVM 장치는 부팅 중에 시스템에서 검색한 순서에 따라 재부팅 후 이름을 변경할 수 있습니다. LVM 레이블은 재부팅과 클러스터 전체에서 계속 유지됩니다.**

- **LVM 레이블은 장치를 LVM 물리 볼륨으로 식별합니다. 물리 볼륨의 UUID인 임의의 고유 식별자를 포함합니다. 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 섹터에 레이블을 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.**

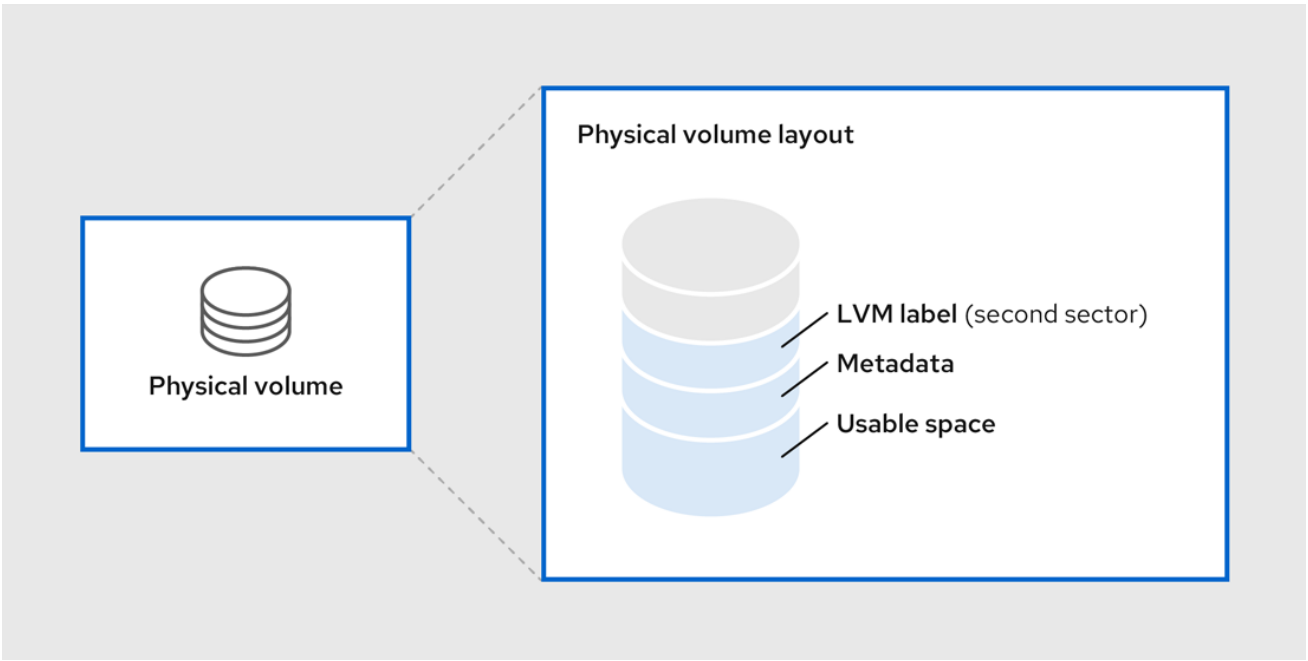
- **현재 LVM을 사용하면 각 물리 볼륨에 0, 1 또는 2 개의 메타데이터의 동일한 사본을 저장할 수 있습니다. 기본값은 1 copy입니다. 물리 볼륨에서 메타데이터 복사본 수를 구성하면 나중에 해당 수를 변경할 수 없습니다. 첫 번째 복사본은 장치 시작 부분에 저장되며 레이블 바로 뒤에 저장됩니다. 두 번째 사본이 있는 경우 장치의 끝에 배치됩니다. 의도한 것과 다른 디스크에 작성하여 디스크 시작 부분에 있는 영역을 실수로 덮어 쓰면 장치 끝에 있는 메타데이터의 두 번째 복사본을 통해 메타데이터를 복구할 수 있습니다.**

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

참고

Linux 커널과 이 문서 전체에서 섹터는 512바이트로 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 사용하여 LVM 물리 볼륨으로 레이블을 지정하는 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 있는 경우 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. 둘 이상의 파티션이 있고 두 파티션이 모두 동일한 볼륨 그룹에 있는 디스크가 있는 경우 볼륨을 생성할 때 논리 볼륨에 포함할 파티션을 지정합니다.

LVM에서 파티션되지 않은 디스크를 물리 볼륨으로 사용하도록 지원하지만 파티션 없이 PV를 생성하는 것이 혼합된 운영 체제 환경에서 문제가 발생할 수 있으므로 전체 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 바꿉니다.

사전 요구 사항

- lvm2 패키지가 설치되어 있습니다.

절차

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

```
# 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 속한 물리 볼륨으로 표시됩니다.

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

a. **pvdisplay** 명령은 각 물리 볼륨에 대한 자세한 멀티 라인 출력을 제공합니다. 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 시스템에서 지원되는 모든 **LVM** 블록 장치를 검사하여 물리 볼륨을 검사합니다. 이 명령으로 특정 물리 볼륨 스캔을 방지하도록 **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) 및 lvm(8) 도움말 페이지

3.4. LVM 물리 볼륨 제거

LVM에서 장치가 더 이상 필요하지 않은 경우 pvremove 명령을 사용하여 LVM 레이블을 제거할 수 있습니다. pvremove 명령을 실행하면 빈 물리 볼륨에서 LVM 메타데이터를 0으로 나눕니다.

절차

1. 물리 볼륨 제거:

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

2. 기존 물리 볼륨을 확인하고 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 일부인 경우cess reduce 명령을 사용하여 볼륨 그룹에서 제거해야 합니다. 자세한 내용은 볼륨 그룹에서 물리 볼륨 제거를참조하십시오.

추가 리소스

- pvremove(8) 도움말 페이지

3.5. 추가 리소스

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

4장. LVM 볼륨 그룹 관리

볼륨 그룹(VG)은 논리 볼륨(LV)을 할당할 수 있는 디스크 공간 풀을 생성하는 PV(물리 볼륨) 컬렉션입니다.

볼륨 그룹 내에서 할당에 사용할 수 있는 디스크 공간은 Extent라는 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 크기가 동일합니다. 볼륨 그룹은 논리 확장 영역을 물리 확장 영역에 매핑합니다.

4.1. LVM 볼륨 그룹 생성

이 절차에서는 */dev/vdb1* 및 */dev/vdb2* 물리 볼륨을 사용하여 LVM 볼륨 그룹(VG) *myvg* 를 생성하는 방법을 설명합니다.

사전 요구 사항

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

절차

1. 볼륨 그룹을 생성합니다.

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

그러면 이름이 *myvg* 인 VG가 생성됩니다. PV */dev/vdb1* 및 */dev/vdb2* 는 *myvg* VG의 기본 스토리지 수준입니다.

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

a.

Knative Servings 명령은 볼륨 그룹별로 한 줄을 표시하는 구성 가능한 형식으로 볼륨 그룹 정보를 제공합니다.

```
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  myvg   4   1   0 wz--n-   3.98g 1008.00m
```

b.

Knative Serving display 명령은 크기, 확장 영역, 물리 볼륨 수, 고정된 형식의 기타 옵션과 같은 볼륨 그룹 속성을 표시합니다. 다음 예제에서는 볼륨 그룹 ***myvg*** 에 대한 **vgdisplay** 명령의 출력을 보여줍니다. 볼륨 그룹을 지정하지 않으면 기존 볼륨 그룹이 모두 표시됩니다.

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

c.

Knative Servingscan 명령은 볼륨 그룹에 대해 시스템에서 지원되는 모든 **LVM** 블록 장치를 검사합니다.

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

3.

선택 사항: 하나 이상의 사용 가능한 물리 볼륨을 추가하여 볼륨 그룹의 용량을 늘립니다.

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

추가 리소스

- **pvcreate(8), arrow extend(8), arrow display(8), arrows (8), arrowscan (8), lvm(8)** 도움말 페이지

4.2. LVM 볼륨 그룹 결합

두 개의 볼륨 그룹을 단일 볼륨 그룹으로 결합하려면 **Vertical merge** 명령을 사용합니다. 볼륨의 물리 확장 크기가 동일하고 대상 볼륨 그룹의 물리 및 논리 볼륨 합계가 대상 볼륨 그룹 제한에 적합한 경우 비

활성 "소스" 볼륨을 활성 또는 비활성 "대상 "대상" 볼륨과 병합할 수 있습니다.

절차

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

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

추가 리소스

- KnativeServingmerge(8) 도움말 페이지

4.3. 볼륨 그룹에서 물리 볼륨 제거

볼륨 그룹에서 사용하지 않는 물리 볼륨을 제거하려면 KnativeServing reduce 명령을 사용합니다. Knative Servingreduce 명령은 하나 이상의 빈 물리 볼륨을 제거하여 볼륨 그룹의 용량을 줄입니다. 그러면 이러한 물리 볼륨을 다른 볼륨 그룹에서 사용하거나 시스템에서 제거할 수 있습니다.

절차

1. 물리 볼륨을 계속 사용하는 경우 동일한 볼륨 그룹의 다른 물리 볼륨으로 데이터를 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 다음을 수행하십시오.

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

- pxereduce(8), pvhiera(8), pvs(8) 매뉴얼 페이지

4.4. LVM 볼륨 그룹 분할

다음 절차에서는 기존 볼륨 그룹을 분할하는 방법을 설명합니다. 물리 볼륨에 사용되지 않는 공간이 충분한 경우 새 디스크를 추가하지 않고 새 볼륨 그룹을 생성할 수 있습니다.

초기 설정에서 *myvg* 볼륨 그룹은 */dev/vdb1*,*/dev/vdb2*, */dev/vdb3* 로 구성됩니다. 이 절차를 완료하면 *myvg* 볼륨 그룹은 */dev/vdb1* 및 */dev/vdb2*, 두 번째 볼륨 그룹인 *vg* 로 구성됩니다. */dev/vdb3*.

사전 요구 사항

- 볼륨 그룹에 충분한 공간이 있습니다. **vgscan** 명령을 사용하여 볼륨 그룹에서 현재 사용 가능한 공간 크기를 확인합니다.

- 기존 물리 볼륨의 사용 가능한 용량에 따라 **pv hiera** 명령을 사용하여 사용된 모든 물리 확장 영역을 다른 물리 볼륨으로 이동합니다. 자세한 내용은 볼륨 그룹에서 물리 볼륨 제거를 참조하십시오.

절차

1. 기존 볼륨 그룹 *myvg* 를 새 볼륨 그룹 *yourvg* 로 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 개의 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 그룹이 */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

- **pxesplit(8) ,virtualizations(8)** 및 **pvs(8)** 도움말 페이지

4.5. LVM 볼륨 그룹 이름

이 절차에서는 기존 볼륨 그룹 **myvg** 의 이름을 **myvg1** 로 변경합니다.

절차

1. 볼륨 그룹을 비활성화합니다. 클러스터형 볼륨 그룹인 경우 이러한 각 노드에서 다음 명령을 사용하여 활성화된 모든 노드에서 볼륨 그룹을 비활성화합니다.

```
# vgchange --activate n myvg
```

2. 기존 볼륨 그룹의 이름을 변경합니다.

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

장치에 대한 전체 경로를 지정하여 볼륨 그룹의 이름을 변경할 수도 있습니다.

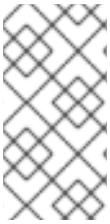
```
# vgrename /dev/myvg /dev/myvg1
```

추가 리소스

- **netrename(8)** 도움말 페이지

4.6. 볼륨 그룹을 다른 시스템으로 이동

전체 **LVM** 볼륨 그룹을 다른 시스템으로 이동할 수 있습니다. 이 작업을 수행할 때 **Knative Servingexport** 및 **pxeimport** 명령을 사용하는 것이 좋습니다.



참고

ble import 명령의 **--force** 인수를 사용할 수 있습니다. 이를 통해 물리 볼륨이 누락된 볼륨 그룹을 가져와서 각각 **--removemissing** 명령을 실행할 수 있습니다.

Knative Servingexport 명령을 사용하면 시스템에 대한 비활성 볼륨 그룹에 액세스할 수 없으므로 물리 볼륨을 분리할 수 있습니다. pxe import 명령을 실행하면 KnativeServing export 명령이 비활성 상태가 된 후 시스템에서 볼륨 그룹에 다시 액세스할 수 있습니다.

볼륨 그룹을 한 시스템에서 다른 시스템으로 이동하려면 다음 단계를 수행합니다.

1. *사용자가 볼륨 그룹의 활성 볼륨의 파일에 액세스하고 있지 않은지 확인한 다음 논리 볼륨을 마운트 해제합니다.*

2. *VG change 명령의 -a n 인수를 사용하여 볼륨 그룹을 비활성으로 표시하여 볼륨 그룹의 추가 활동을 방지합니다.*

3. *KnativeServing export 명령을 사용하여 볼륨 그룹을 내보냅니다. 이렇게 하면 제거할 시스템에서 액세스할 수 없습니다.*

   볼륨 그룹을 내보낸 후 다음 예와 같이 pvscan 명령을 실행할 때 물리 볼륨이 내보낸 볼륨 그룹에 있는 것으로 표시됩니다.

   ```
   # 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]
     ...
   ```

   시스템이 다음에 종료되면 볼륨 그룹을 구성하는 디스크를 분리하고 새 시스템에 연결할 수 있습니다.

4. *디스크를 새 시스템에 연결할 때cess import 명령을 사용하여 볼륨 그룹을 가져와 새 시스템에서 액세스할 수 있습니다.*

5. *pxe change 명령의 -a y 인수를 사용하여 볼륨 그룹을 활성화합니다.*

6. *파일 시스템을 마운트하여 사용할 수 있도록 합니다.*

4.7. LVM 볼륨 그룹 제거

이 절차에서는 기존 볼륨 그룹을 제거하는 방법을 설명합니다.

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 그룹을 제거합니다.*

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

추가 리소스

- ***netremove(8)** 도움말 페이지*

5장. LVM 논리 볼륨 관리

논리 볼륨은 파일 시스템, 데이터베이스 또는 애플리케이션에서 사용할 수 있는 가상 블록 스토리지 장치입니다. LVM 논리 볼륨을 생성하기 위해 PV(물리 볼륨)가 볼륨 그룹(VG)으로 결합됩니다. 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 볼륨에서 별도의 / boot 파티션을 사용하여 항상 LVM 볼륨 내에 / 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 프로비저닝된 논리 볼륨(볼륨 내)

씬 프로비저닝된 논리 볼륨을 사용하면 사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다. 씬 프로비저닝된 볼륨 세트를 생성하면 시스템이 요청된 전체 스토리지 용량을 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 프로비저닝된 스냅샷 볼륨

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

볼륨 캐시

LVM에서는 더 큰 느린 블록 장치를 위한 SSD 드라이브 또는 쓰기 캐시와 같은 빠른 블록 장치를 사용할 수 있습니다. 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. CLI 명령 사용

다음 섹션에서는 LVM CLI 명령의 몇 가지 일반적인 작동 기능에 대해 설명합니다.

명령줄 인수에서 단위 지정

명령줄 인수에 크기가 필요한 경우 단위를 항상 명시적으로 지정할 수 있습니다. 단위를 지정하지 않으면 기본값이 KB 또는 MB로 가정합니다. LVM CLI 명령은 분수를 허용하지 않습니다.

명령줄 인수에서 단위를 지정하는 경우 LVM은 대소문자를 구분하지 않습니다. M 또는 m을 지정하는 것은 동일합니다(예: 2개 수 1024개)가 사용됩니다. 그러나 명령에 --units 인수를 지정할 때 소문자는 단위가 1024의 복수에 있고 대문자는 장치가 1000의 복수에 있음을 나타냅니다.

볼륨 그룹 및 논리 볼륨 지정

LVM CLI 명령에서 볼륨 그룹 또는 논리 볼륨을 지정할 때 다음 사항에 유의하십시오.

- *여기서 명령은 볼륨 그룹 또는 논리 볼륨 이름을 인수로 사용하는 경우 전체 경로 이름은 선택 사항입니다. KnativeServing0이라는 볼륨 그룹에서 lvol0 이라는 논리 볼륨은 KnativeServing 0 /lvol0 으로 지정할 수 있습니다.*

- *볼륨 그룹 목록이 필요하지만 비어 있는 경우 모든 볼륨 그룹 목록이 대체됩니다.*

- *논리 볼륨 목록이 필요하지만 볼륨 그룹이 제공되는 경우 해당 볼륨 그룹의 모든 논리 볼륨 목록이 대체됩니다. 예를 들어 lvdisplayvirtualization0 명령은 볼륨 그룹 pxe 0 의 모든 논리 볼륨을 표시합니다.*

출력 세부 정보 표시 증가

모든 LVM 명령은 출력 세부 정보를 높이기 위해 여러 번 입력할 수 있는 -v 인수를 허용합니다. 다음 예제에서는 lvcreate 명령의 기본 출력을 보여줍니다.

```
# lvcreate -L 50MB new_vg
  Rounding up size to full physical extent 52.00 MB
  Logical volume "lvol0" created
```

다음 명령은 -v 인수를 사용하여 lvcreate 명령의 출력을 보여줍니다.

```
# lvcreate -v -L 50MB new_vg
  Rounding up size to full physical extent 52.00 MB
    Archiving volume group "new_vg" metadata (seqno 1).
    Creating logical volume lvol0
    Creating volume group backup "/etc/lvm/backup/new_vg" (seqno 2).
    Activating logical volume new_vg/lvol0.
    activation/volume_list configuration setting not defined: Checking only host tags for
new_vg/lvol0.
    Creating new_vg-lvol0
    Loading table for new_vg-lvol0 (253:0).
    Resuming new_vg-lvol0 (253:0).
    Wiping known signatures on logical volume "new_vg/lvol0"
    Initializing 4.00 KiB of logical volume "new_vg/lvol0" with value 0.
  Logical volume "lvol0" created
```

-vv,-vvv 및 -vv v 인수는 명령 실행에 대한 세부 정보를 점점 더 많이 표시합니다. vvvv 인수는 현재 최대 정보를 제공합니다. 다음 예제는 지정된 -vvvv 인수를 사용하여 lvcreate 명령에 대한 출력의 처음 몇 줄을 보여줍니다.

```
# lvcreate -vvvv -L 50MB new_vg
#lvmcmdline.c:913       Processing: lvcreate -vvvv -L 50MB new_vg
#lvmcmdline.c:916       O_DIRECT will be used
#config/config.c:864     Setting global/locking_type to 1
#locking/locking.c:138     File-based locking selected.
#config/config.c:841     Setting global/locking_dir to /var/lock/lvm
#activate/activate.c:358     Getting target version for linear
#ioctl/libdm-iface.c:1569       dm version   OF   [16384]
#ioctl/libdm-iface.c:1569       dm versions   OF   [16384]
#activate/activate.c:358     Getting target version for striped
#ioctl/libdm-iface.c:1569       dm versions   OF   [16384]
#config/config.c:864     Setting activation/mirror_region_size to 512
...
```

LVM CLI 명령에 대한 도움말 표시

명령의 **--help** 인수를 사용하여 **LVM CLI** 명령에 대한 도움말을 표시할 수 있습니다.

```
# commandname --help
```

명령의 도움말 페이지를 표시하려면 **man** 명령을 실행합니다.

```
# man commandname
```

man lvm 명령은 **LVM**에 대한 일반적인 온라인 정보를 제공합니다.

5.3. LVM 논리 볼륨 생성

이 절차에서는 **/dev/vdb1,/dev/vdb2, /dev/vdb3** 물리 볼륨을 사용하여 생성되는 **myvg** 볼륨(LV)을 생성하는 방법을 설명합니다.

사전 요구 사항

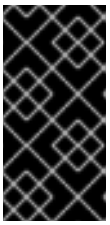
- **lvm2** 패키지가 설치되어 있습니다.
- 볼륨 그룹이 생성됩니다. 자세한 내용은 *LVM 볼륨 그룹 만들기*를 참조하십시오.

절차

1. 논리 볼륨 생성:
   -

```
# lvcreate -n mylv -L 500M myvg
```

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



중요

VG에 요청된 크기 및 유형에 대한 사용 가능한 물리 확장 영역 수가 충분하지 않으면 명령이 실패합니다.

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

   a. *lvs 명령은 논리 볼륨별로 한 줄을 표시하는 구성 가능한 형식으로 논리 볼륨 정보를 제공합니다.*

   ```
   # 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 스캔하여 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

5.4. RAID0(STRIPED) 논리 볼륨 생성

RAID0 논리 볼륨은 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 스트라이프 크기 단위로 분산합니다.

RAID0 볼륨을 생성하는 명령의 형식은 다음과 같습니다.

```
lvcreate --type raid0[_meta] --stripes Stripes --stripesize StripeSize VolumeGroup
[PhysicalVolumePath ...]
```

표 5.1. RAID0 명령 생성 매개변수

매개변수	설명

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 무해하기 때문에 미러링된 데이터 블록을 RAID1/10으로 저장하거나 패리티 블록을 RAID4/5/6으로 계산하고 저장할 필요가 없습니다. 따라서 미러링되거나 패리티 블록의 동기화 진행 상태를 유지하기 위해 메타데이터 볼륨이 필요하지 않습니다. 그러나 메타데이터 볼륨은 RAID0에서 RAID4/5/6/10으로 변환하고 각 할당 실패를 방지하기 위해 **raid0_meta** 를 미리 할당하도록 해당 메타데이터 볼륨을 미리 할당해야 합니다.
--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.
VolumeGroup	사용할 볼륨 그룹을 지정합니다.
PhysicalVolumePath ...	사용할 장치를 지정합니다. 이 값을 지정하지 않으면 LVM에서 *Stripes* 옵션에 지정된 장치 수를 선택합니다.

이 예제 절차에서는 /dev/sda1,/dev/sdb1, /dev/sdc1 의 디스크에서 데이터를 스트라이프하는 mylv 라는 LVM RAID0 논리 볼륨을 생성합니다.

1. ***볼륨 그룹에서 사용할 디스크에 pvcreate 명령을 사용하여 LVM 물리 볼륨으로 레이블을 지정합니다.***

주의

이 명령은 /dev/sda1,/dev/sdb1, /dev/sdc1 의 모든 데이터를 삭제합니다.

```
# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdc1" successfully created
```

2. 볼륨 그룹 **myvg** 를 생성합니다. 다음 명령은 볼륨 그룹 **myvg** 를 생성합니다.

```
# vgcreate myvg /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "myvg" successfully created
```

KnativeServing s 명령을 사용하여 새 볼륨 그룹의 속성을 표시할 수 있습니다.

```
# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  myvg   3   0   0 wz--n- 51.45G 51.45G
```

3. 생성한 볼륨 그룹에서 **RAID0** 논리 볼륨을 생성합니다. 다음 명령은 볼륨 그룹 **myvg** 에서 **RAID0** 볼륨 **mylv** 를 생성합니다. 이 예에서는 스트라이프 **3**개와 스트라이프 크기가 **4**킬로바이트인 **2GB** 크기의 논리 볼륨을 생성합니다.

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

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

```
# mkfs.ext4 /dev/myvg/mylv
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 525312 4k blocks and 131376 inodes
Filesystem UUID: 9d4c0704-6028-450a-8b0a-8875358c0511
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

다음 명령은 논리 볼륨을 마운트하고 파일 시스템 디스크 사용량을 보고합니다.

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

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

```
# lvchange --activate n myvg/mylv
```

3. 기존 논리 볼륨 이름 변경:

```
# lvrename myvg mylv mylv1
Logical volume "mylv" successfully renamed to "mylv1"
```

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

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

추가 리소스

- **lvrename(8)** 도움말 페이지

5.6. 논리 볼륨에서 디스크 제거

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

디스크를 제거하려면 먼저 **LVM** 물리 볼륨의 **Extent**를 다른 디스크 또는 디스크 세트로 이동해야 합니다.

절차

1. *LV를 사용할 때 사용한 물리 볼륨 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 그룹에 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 데이터를 이 물리 볼륨으로 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

KnativeServing reduce 명령의 --removemissing 매개변수를 사용할 수 있습니다.

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

추가 리소스

- pvhiera(8) ,tektonextend(8), vereduce(8), pvs(8) 도움말 페이지

5.7. LVM 논리 볼륨 제거

이 절차에서는 볼륨 그룹 myvg 에서 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) 도움말 페이지*

5.8. 영구 장치 번호 구성

메이저 및 마이너 장치 번호는 모듈 로드에서 동적으로 할당됩니다. 일부 애플리케이션은 블록 장치가 항상 동일한 장치(major 및 minor) 번호로 활성화되는 경우 가장 잘 작동합니다. 다음 인수를 사용하여 lvcreate 및 lvchange 명령을 사용하여 지정할 수 있습니다.

```
--persistent y --major major --minor minor
```

작은 번호를 사용하여 이미 다른 장치에 동적으로 할당되지 않았는지 확인하십시오.

NFS를 사용하여 파일 시스템을 내보내는 경우 내보내기 파일에 fsid 매개변수를 지정하면 LVM 내에서 영구 장치 번호를 설정할 필요가 없을 수 있습니다.

5.9. LVM 확장 영역 크기 지정

물리 볼륨을 사용하여 볼륨 그룹을 만들면 디스크 공간은 기본적으로 4MB 확장 영역으로 나뉩니다. 이 범위는 논리 볼륨을 늘리거나 줄일 수 있는 최소 크기입니다. 많은 수의 Extent는 논리 볼륨의 I/O 성능에 영향을 미치지 않습니다.

기본 확장 영역 크기가 적합하지 않은 경우, netcreate 명령에 -s 옵션을 사용하여 확장 영역 크기를 지정할 수 있습니다. 볼륨 그룹이 보유할 수 있는 물리 볼륨 또는 논리 볼륨 수를 제한할 수 있습니다. -p 및 -l 명령의 인수를 사용합니다.

5.10. RHEL 시스템 역할을 사용하여 LVM 논리 볼륨 관리

이 섹션에서는 스토리지 역할을 적용하여 다음 작업을 수행하는 방법을 설명합니다.

- *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**

Ansible 플레이북을 적용하는 방법에 대한 자세한 내용은 역할 적용을 참조하십시오.

5.10.1. 논리 볼륨을 관리하는 Ansible 플레이북 예

이 섹션에서는 예제 **Ansible** 플레이북을 제공합니다. 이 플레이북은 스토리지 역할을 적용하여 볼륨 그룹에 **LVM** 논리 볼륨을 만듭니다.

예 5.1. myvg 볼륨 그룹에 mylv 논리 볼륨을 생성하는 플레이북

```
- hosts: all
  vars:
    storage_pools:
      - name: myvg
        disks:
          - sda
          - sdb
          - sdc
        volumes:
          - name: mylv
            size: 2G
            fs_type: ext4
            mount_point: /mnt
  roles:
    - rhel-system-roles.storage
```

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

- myvg 볼륨 그룹이 없으면 플레이북에서 해당 그룹을 생성합니다.
- 이 플레이북은 mylv 논리 볼륨에 Ext4 파일 시스템을 생성하고 /mnt 에 파일 시스템을 영구적으로 마운트합니다.

추가 리소스

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

5.10.2. 추가 리소스

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

5.11. LVM 볼륨 그룹 제거

이 절차에서는 기존 볼륨 그룹을 제거하는 방법을 설명합니다.

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 그룹을 제거합니다.

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

추가 리소스

- *netremove(8) 도움말 페이지*

6장. 논리 볼륨의 크기 수정

논리 볼륨을 만든 후에는 볼륨의 크기를 수정할 수 있습니다.

6.1. 논리 볼륨 및 파일 시스템 증가

다음 절차에서는 동일한 논리 볼륨에서 논리 볼륨을 확장하고 파일 시스템을 확장하는 방법을 설명합니다.

논리 볼륨의 크기를 늘리려면 **lvextend** 명령을 사용합니다. 논리 볼륨을 확장할 때 볼륨을 확장할 양 또는 확장하려는 양을 표시할 수 있습니다.

사전 요구 사항

1. 파일 시스템이 있는 기존 논리 볼륨(LV)이 있습니다. **df -Th** 명령을 사용하여 파일 시스템 유형을 확인합니다.

   LV 및 파일 시스템 생성에 대한 자세한 내용은 **LVM 논리 볼륨 생성** 을 참조하십시오.

2. 볼륨 그룹에 충분한 공간이 있어야 **LV** 및 파일 시스템을 확장할 수 있습니다. **vgs -o name,vgfree** 명령을 사용하여 사용 가능한 공간을 확인합니다.

절차

1. 선택 사항: 볼륨 그룹에 **LV**를 늘릴 공간이 충분하지 않으면 다음 명령을 사용하여 볼륨 그룹에 새 물리 볼륨을 추가합니다.

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

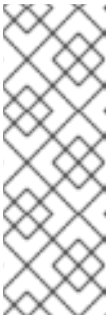
   자세한 내용은 **LVM 볼륨 그룹 만들기를** 참조하십시오.

2. 볼륨 그룹이 충분히 크기 때문에 요구 사항에 따라 다음 단계 중 하나를 실행합니다.

a.

*지정된 크기로 **LV**를 확장하려면 다음 명령을 사용합니다.*

```
# lvextend -L 3G /dev/myvg/mylv
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.
```



참고

lvextend 명령의 -r 옵션을 사용하여 논리 볼륨을 확장하고 단일 명령으로 기본 파일 시스템의 크기를 조정할 수 있습니다.

```
# lvextend -r -L 3G /dev/myvg/mylv
```

주의

동일한 인수와 함께 lvresize 명령을 사용하여 논리 볼륨을 확장할 수도 있지만, 이 명령은 실수로 축소된 것을 보장하지 않습니다.

b.

myvg 볼륨 그룹의 할당되지 않은 공간을 모두 채우도록 mylv 논리 볼륨을 확장하려면 다음 명령을 사용합니다.

```
# 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.
```

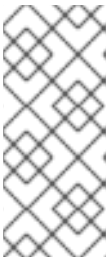
lvcreate 명령과 마찬가지로 lvextend 명령의 -l 인수를 사용하여 논리 볼륨의 크기를 늘리기 위해 확장 영역 수를 지정할 수 있습니다. 이 인수를 사용하여 볼륨 그룹의 백분율 또는 볼륨 그룹에서 나머지 사용 가능한 공간의 백분율을 지정할 수도 있습니다.

3.

lvextend 명령과 함께 r 옵션을 사용하여 LV를 확장하고 단일 명령으로 파일 시스템의 크기를 조정하지 않는 경우 다음 명령을 사용하여 논리 볼륨에서 파일 시스템의 크기를 조정합니다.

```
xfs_growfs /mnt/mnt1/
meta-data=/dev/mapper/myvg-mylv  isize=512    agcount=4, agsize=65536 blks
         =                       sectsz=512   attr=2, projid32bit=1
```

```
      =                 crc=1       finobt=1, sparse=1, rmapbt=0
      =                 reflink=1
data     =                bsize=4096   blocks=262144, imaxpct=25
      =                 sunit=0      swidth=0 blks
naming   =version 2           bsize=4096   ascii-ci=0, ftype=1
log      =internal log        bsize=4096   blocks=2560, version=2
      =                 sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                extsz=4096   blocks=0, rtextents=0
data blocks changed from 262144 to 524288
```



참고

d 옵션이 없으면 **xfs_growfs** 는 파일 시스템을 기본 장치에서 지원하는 최대 크기로 확장합니다. 자세한 내용은 **XFS** 파일 시스템의 크기 증가를 참조하십시오.

ext4 파일 시스템의 크기를 조정하려면 **ext4** 파일 시스템 재설정을 참조하십시오.

검증

- 다음 명령을 사용하여 파일 시스템이 증가하고 있는지 확인합니다.

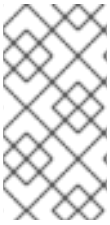
```
# df -Th
Filesystem          Type      Size  Used Avail Use% Mounted on
devtmpfs            devtmpfs  1.9G     0  1.9G   0% /dev
tmpfs               tmpfs     1.9G     0  1.9G   0% /dev/shm
tmpfs               tmpfs     1.9G  8.6M  1.9G   1% /run
tmpfs               tmpfs     1.9G     0  1.9G   0% /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

- **netextend(8), lvextend(8), xfs_growfs(8)** 도움말 페이지

6.2. 논리 볼륨 축소

lvreduce 명령을 사용하여 논리 볼륨의 크기를 줄일 수 있습니다.



참고

GFS2 또는 **XFS** 파일 시스템에서 축소는 지원되지 않으므로 **GFS2** 또는 **XFS** 파일 시스템이 포함된 논리 볼륨의 크기를 줄일 수 없습니다.

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 사용하는 것이 좋습니다.

이 옵션을 사용할 때 **lvreduce** 명령은 논리 볼륨을 축소하기 전에 파일 시스템을 축소하려고 합니다. 파일 시스템이 가득 차거나 파일 시스템이 축소를 지원하지 않는 경우 발생할 수 있는 파일 시스템 축소가 실패하는 경우 **lvreduce** 명령이 실패하고 논리 볼륨을 축소하려고 시도하지 않습니다.

주의

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

lvreduce 명령의 **--test** 옵션을 사용하면 파일 시스템을 확인하거나 파일 시스템의 크기를 테스트하지 않으므로 작업이 안전한 위치를 나타내지 않습니다.

절차

- **myvg** 볼륨 그룹의 **mylv** 논리 볼륨을 **64MB**로 축소하려면 다음 명령을 사용합니다.

```
# lvreduce --resizefs -L 64M myvg/mylv
fsck from util-linux 2.37.2
/dev/mapper/myvg-mylv: clean, 11/25688 files, 4800/102400 blocks
resize2fs 1.46.2 (28-Feb-2021)
Resizing the filesystem on /dev/mapper/myvg-mylv to 65536 (1k) blocks.
The filesystem on /dev/mapper/myvg-mylv is now 65536 (1k) blocks long.

Size of logical volume myvg/mylv changed from 100.00 MiB (25 extents) to 64.00 MiB (16 extents).
 Logical volume myvg/mylv successfully resized.
```

이 예에서 **mylv** 에는 파일 시스템이 포함되어 있으며 이 명령은 논리 볼륨과 함께 크기를 조정합니다.

- 크기 조정 값 앞에 **-** 기호를 지정하면 값이 논리 볼륨의 실제 크기에서 제거됩니다. 논리 볼륨을 **64MB**의 절대 크기로 축소하려면 다음 명령을 사용합니다.

```
# lvreduce --resizefs -L -64M myvg/mylv
```

추가 리소스

- **lvreduce(8)** 도움말 페이지

6.3. 스트라이핑된 논리 볼륨 확장

스트라이핑된 논리 볼륨의 크기를 늘리려면 스트라이프를 지원하기 위해 볼륨 그룹을 구성하는 기본 물리 볼륨에 충분한 여유 공간이 있어야 합니다. 예를 들어 전체 볼륨 그룹을 사용하는 양방향 스트라이프가 있는 경우 볼륨 그룹에 하나의 물리 볼륨을 추가하면 스트라이프를 확장할 수 없습니다. 대신 볼륨 그룹에 두 개 이상의 물리 볼륨을 추가해야 합니다.

예를 들어 다음**virtualization s** 명령과 함께 표시되는 두 개의 기본 물리 볼륨으로 구성된 볼륨 그룹 **virtualization**을 고려해 보십시오.

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

볼륨 그룹의 전체 공간을 사용하여 스트라이프를 생성할 수 있습니다.

```
# lvcreate -n stripe1 -L 271.31G -i 2 vg
  Using default stripesize 64.00 KB
  Rounding up size to full physical extent 271.31 GB
  Logical volume "stripe1" created
# lvs -a -o +devices
  LV      VG   Attr   LSize   Origin Snap%  Move Log Copy%  Devices
  stripe1 vg   -wi-a- 271.31G                       /dev/sda1(0),/dev/sdb1(0)
```

볼륨 그룹에 더 이상 여유 공간이 없습니다.

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

다음 명령은 볼륨 그룹에 또 다른 물리 볼륨을 추가합니다. 그러면 **135GB**의 추가 공간이 있습니다.

```
# vgextend vg /dev/sdc1
  Volume group "vg" successfully extended
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     3   1   0 wz--n- 406.97G 135.66G
```

이 시점에서 데이터를 스트라이프하기 위해 두 개의 기본 장치가 필요하므로 볼륨 그룹의 전체 크기로 스트라이핑된 논리 볼륨을 확장할 수 없습니다.

```
# lvextend vg/stripe1 -L 406G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 406.00 GB
  Insufficient suitable allocatable extents for logical volume stripe1: 34480
more required
```

스트라이핑된 논리 볼륨을 확장하려면 다른 물리 볼륨을 추가한 다음 논리 볼륨을 확장합니다. 이 예에서는 볼륨 그룹에 두 개의 물리 볼륨을 추가하여 논리 볼륨을 볼륨 그룹의 전체 크기로 확장할 수 있습니다.

```
# vgextend vg /dev/sdd1
  Volume group "vg" successfully extended
# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  vg     4   1   0 wz--n- 542.62G 271.31G
# lvextend vg/stripe1 -L 542G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 542.00 GB
  Logical volume stripe1 successfully resized
```

스트라이핑된 논리 볼륨을 확장할 수 있는 기본 물리 장치가 충분하지 않은 경우 확장 기능이 제거되지 않은 경우 볼륨을 확장할 수 있으므로 성능이 저하될 수 있습니다. 논리 볼륨에 공간을 추가할 때 기본 작업은 기존 논리 볼륨의 마지막 세그먼트에서 동일한 스트라이핑 매개 변수를 사용하지만 해당 매개 변수를 재정의할 수 있습니다. 다음 예제에서는 초기 **lvextend** 명령이 실패한 후 남은 여유 공간을 사용하도록 기존 **striped** 논리 볼륨을 확장합니다.

```
# lvextend vg/stripe1 -L 406G
  Using stripesize of last segment 64.00 KB
  Extending logical volume stripe1 to 406.00 GB
```

```
  Insufficient suitable allocatable extents for logical volume stripe1: 34480
more required
# lvextend -i1 -l+100%FREE vg/stripe1
```

7장. LVM에 대한 사용자 정의 보고

LVM은 사용자 지정 보고서를 생성하고 보고서의 출력을 필터링하는 다양한 구성 및 명령줄 옵션을 제공합니다. LVM 보고 기능 및 기능에 대한 자세한 설명은 lvmreport(7) 도움말 페이지를 참조하십시오.

pvs,lvs,virtualization s 명령을 사용하여 LVM 개체에 대한 간결하고 사용자 지정 가능한 보고서를 생성할 수 있습니다. 이러한 명령이 생성하는 보고서에는 각 오브젝트에 대한 한 줄의 출력이 포함됩니다. 각 줄에는 오브젝트와 관련된 속성의 순서가 지정된 필드 목록이 포함되어 있습니다. 물리 볼륨, 볼륨 그룹, 논리 볼륨, 물리 볼륨 세그먼트 및 논리 볼륨 세그먼트에서 보고할 오브젝트를 선택하는 5가지 방법이 있습니다.

lvm fullreport 명령을 사용하여 물리 볼륨, 볼륨 그룹, 논리 볼륨 세그먼트 및 논리 볼륨 세그먼트에 대한 정보를 한 번에 보고할 수 있습니다. 이 명령 및 해당 기능에 대한 자세한 내용은 lvm-fullreport(8) 매뉴얼 페이지를 참조하십시오.

LVM에서는 LVM 명령 실행 중에 수집된 완전한 오브젝트 식별을 통해 작업, 메시지 및 오브젝트별 상태가 포함된 로그 보고서를 지원합니다. LVM 로그 보고서에 대한 자세한 내용은 lvmreport(7) 도움말 페이지를 참조하십시오.

7.1. LVM 디스플레이 형식 제어

pvs,lvs 또는 KnativeServing s 명령을 사용할지 여부에 따라 표시되는 기본 필드 세트와 정렬 순서가 결정됩니다. 다음 인수를 사용하여 이러한 명령의 출력을 제어할 수 있습니다.

- -o 인수를 사용하여 기본적으로 표시되는 필드를 변경할 수 있습니다. 예를 들어 다음 명령은 물리 볼륨 이름과 크기만 표시합니다.

```
# pvs -o pv_name,pv_size
PV PSize
/dev/sdb1 17.14G
/dev/sdc1 17.14G
/dev/sdd1 17.14G
```

- -o 인수와 함께 사용되는 더하기 기호(+)를 사용하여 출력에 필드를 추가할 수 있습니다.

  다음 예제는 기본 필드 외에도 물리 볼륨의 UUID를 표시합니다.

```
# pvs -o +pv_uuid
PV VG Fmt Attr PSize PFree PV UUID
```

```
/dev/sdb1 new_vg lvm2 a- 17.14G 17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-
dqGeXY
/dev/sdc1 new_vg lvm2 a- 17.14G 17.09G Joqlch-yWSj-kuEn-ldwM-01S9-X08M-mcpsVe
/dev/sdd1 new_vg lvm2 a- 17.14G 17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-UqkCS
```

- 명령에 **-v** 인수를 추가하면 몇 가지 추가 필드가 포함됩니다. 예를 들어 **pvs -v** 명령은 기본 필드 외에도 **DevSize** 및 **PV UUID** 필드를 표시합니다.

```
# pvs -v
Scanning for physical volume names
PV VG Fmt Attr PSize PFree DevSize PV UUID
/dev/sdb1 new_vg lvm2 a- 17.14G 17.14G 17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-
dqGeXY
/dev/sdc1 new_vg lvm2 a- 17.14G 17.09G 17.14G Joqlch-yWSj-kuEn-ldwM-01S9-XO8M-
mcpsVe
/dev/sdd1 new_vg lvm2 a- 17.14G 17.14G 17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-
tUqkCS
```

- **--noheadings** 인수는 제목 행을 표시하지 않습니다. 스크립트 작성에 유용할 수 있습니다.

  다음 예제에서는 모든 물리 볼륨 목록을 생성하는 **pv_name** 인수와 함께 **--noheadings** 인수를 사용합니다.

```
# pvs --noheadings -o pv_name
/dev/sdb1
/dev/sdc1
/dev/sdd1
```

- **--separator** 구분 기호 인수는 구분자 를 사용하여 각 필드를 구분합니다.

  다음 예제에서는 **pvs** 명령의 기본 출력 필드를 등호(**=**)로 구분합니다.

```
# pvs --separator =
PV=VG=Fmt=Attr=PSize=PFree
/dev/sdb1=new_vg=lvm2=a-=17.14G=17.14G
/dev/sdc1=new_vg=lvm2=a-=17.14G=17.09G
/dev/sdd1=new_vg=lvm2=a-=17.14G=17.14G
```

  구분자 인수를 사용할 때 필드를 계속 정렬하려면 **--aligned** 인수와 함께 구분자 인수를 사용합니다.

```
# pvs --separator = --aligned
PV =VG =Fmt =Attr=PSize =PFree
/dev/sdb1 =new_vg=lvm2=a- =17.14G=17.14G
```

```
/dev/sdc1 =new_vg=lvm2=a- =17.14G=17.09G
/dev/sdd1 =new_vg=lvm2=a- =17.14G=17.14G
```

lvs 또는 pxe s 명령의 -P 인수를 사용하여 출력에 표시되지 않는 실패한 볼륨에 대한 정보를 표시할 수 있습니다.

표시 인수의 전체 목록은 pvs(8), pxes (8) 및 lvs(8) 매뉴얼 페이지를 참조하십시오.

볼륨 그룹 필드는 물리 볼륨(및 물리 볼륨 세그먼트) 필드 또는 논리 볼륨(및 논리 볼륨 세그먼트) 필드와 혼합할 수 있지만 물리 볼륨 및 논리 볼륨 필드는 혼합할 수 없습니다. 예를 들어 다음 명령은 각 물리 볼륨에 대해 한 줄의 출력을 표시합니다.

```
# vgs -o +pv_name
  VG     #PV #LV #SN Attr   VSize  VFree  PV
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdc1
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdd1
  new_vg   3   1   0 wz--n- 51.42G 51.37G /dev/sdb1
```

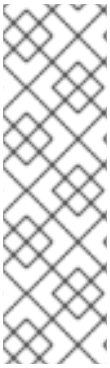
7.2. LVM 오브젝트 필드 표시

이 섹션에서는 pvs,tektons 및 lvs 명령을 사용하여 LVM 개체에 대해 표시할 수 있는 정보를 나열하는 일련의 테이블을 제공합니다.

편의를 위해 명령에 대한 기본값과 일치하는 경우 필드 이름 접두사를 삭제할 수 있습니다. 예를 들어, pvs 명령을 사용하면 name 은 pv_name 을 의미하지만, KnativeServing s 명령을 사용하면 name 이1/_name 으로 해석됩니다.

다음 명령을 실행하는 것은 pvs -o pv_free 를 실행하는 것과 같습니다.

```
# pvs -o free
  PFree
  17.14G
  17.09G
  17.14G
```



참고

***pvs** 및 **lvs** 출력의 특성 필드의 문자 수는 이후 릴리스에서 증가할 수 있습니다. 기존 문자 필드는 위치가 변경되지 않지만 새 필드가 마지막에 추가될 수 있습니다. 특정 특성 문자를 검색하는 스크립트를 작성할 때 필드 시작 위치에 대한 상대적 위치를 기준으로 문자를 검색하지만 필드 끝에 대한 상대적 위치를 검색할 때는 고려해야 합니다. 예를 들어 **lv_attr** 필드의 **ninth** 비트에서 **p** 문자를 검색하려면 문자열 "^/...을 검색할 수 있습니다....P/"이지만 "/*p$/" 문자열을 검색해서는 안 됩니다.*

*표 7.1. "pvs 명령 디스플레이 필드" 헤더 디스플레이에 표시되는 필드 이름과 필드에 대한 설명과 함께 **pvs** 명령의 표시 인수를 나열합니다.*

*표 7.1. **pvs** 명령 디스플레이 필드*

인수	header	설명
dev_size	DevSize	물리 볼륨이 생성된 기본 장치의 크기입니다.
pe_start	첫 번째 PE	기본 장치에서 첫 번째 물리 확장 영역의 시작 부분에 대한 오프셋입니다.
pv_attr	attr	물리 볼륨의 상태: (a)할 수 있거나 e(x)ported입니다.
pv_fmt	FMT	물리 볼륨의 메타데이터 형식(**lvm2** 또는 **lvm1**)
pv_free	PFree	물리 볼륨에 남아 있는 여유 공간
pv_name	PV	물리 볼륨 이름
pv_pe_alloc_count	alloc	사용된 물리 확장 영역 수
pv_pe_count	PE	물리 확장 영역 수
pvseg_size	SSize	물리 볼륨의 세그먼트 크기
pvseg_start	시작	물리 볼륨 세그먼트에서 시작 물리 범위
pv_size	PSize	물리 볼륨의 크기
pv_tags	PV 태그	물리 볼륨에 연결된 LVM 태그
pv_used	사용됨	물리 볼륨에서 현재 사용된 공간의 양
pv_uuid	PV UUID	물리 볼륨의 UUID

pvs 명령은 기본적으로 다음 필드를 표시합니다. pv_name, net_name ,pv_ fmt,pv_attr,pv_size,pv_free. 디스플레이는 pv_name 으로 정렬됩니다.

```
# pvs
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G
```

pvs 명령과 함께 -v 인수를 사용하여 기본 디스플레이 dev_size,pv_uuid 에 다음 필드를 추가합니다.

```
# pvs -v
    Scanning for physical volume names
  PV         VG     Fmt  Attr PSize  PFree  DevSize PV UUID
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G  17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-
dqGeXY
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G  17.14G Joqlch-yWSj-kuEn-IdwM-01S9-XO8M-
mcpsVe
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.13G  17.14G yvfvZK-Cf31-j75k-dECm-0RZ3-0dGW-
tUqkCS
```

pvs 명령의 --segments 인수를 사용하여 각 물리 볼륨 세그먼트에 대한 정보를 표시할 수 있습니다. 세그먼트는 확장 영역의 그룹입니다. 세그먼트 보기는 논리 볼륨이 조각화되어 있는지 확인하려는 경우 유용할 수 있습니다.

pvs --segments 명령은 기본적으로 다음 필드를 표시합니다. pv_name , pv_name,pv_fmt,pv_attr,pv_size,pv seg_start,pvseg_size. 디스플레이는 물리 볼륨 내의 pv_name 및 pvseg_size 에 따라 정렬됩니다.

```
# pvs --segments
  PV         VG         Fmt  Attr PSize  PFree  Start SSize
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M     0  1172
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M  1172    16
  /dev/hda2  VolGroup00 lvm2 a-   37.16G 32.00M  1188     1
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G     0    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    26    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    50    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G    76    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   100    26
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   126    24
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   150    22
  /dev/sda1  vg         lvm2 a-   17.14G 16.75G   172  4217
  /dev/sdb1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdc1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdd1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sde1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdf1  vg         lvm2 a-   17.14G 17.14G     0  4389
  /dev/sdg1  vg         lvm2 a-   17.14G 17.14G     0  4389
```

pvs -a 명령을 사용하여 LVM 물리 볼륨으로 초기화되지 않은 LVM에서 감지된 장치를 확인할 수 있습니다.

```
# pvs -a
  PV                    VG     Fmt  Attr PSize  PFree
  /dev/VolGroup00/LogVol01             --      0     0
  /dev/new_vg/lvol0                    --      0     0
  /dev/ram                             --      0     0
  /dev/ram0                            --      0     0
  /dev/ram2                            --      0     0
  /dev/ram3                            --      0     0
  /dev/ram4                            --      0     0
  /dev/ram5                            --      0     0
  /dev/ram6                            --      0     0
  /dev/root                            --      0     0
  /dev/sda                             --      0     0
  /dev/sdb                             --      0     0
  /dev/sdb1             new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc                             --      0     0
  /dev/sdc1             new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd                             --      0     0
  /dev/sdd1             new_vg lvm2 a-   17.14G 17.14G
```

표 7.2. "VGs 표시 필드"은 헤더 디스플레이에 나타나는 필드 이름과 필드에 대한 설명과 함께 vgs 명령의 표시 인수를 나열합니다.

표 7.2. VGs 표시 필드

인수	header	설명
lv_count	#LV	볼륨 그룹에 포함된 논리 볼륨 수
max_lv	MaxLV	볼륨 그룹에 허용되는 최대 논리 볼륨 수(0인 경우 무제한)
max_pv	MaxPV	볼륨 그룹에 허용되는 최대 물리 볼륨 수(0인 경우 무제한)
pv_count	#PV	볼륨 그룹을 정의하는 물리 볼륨 수
snap_count	#SN	볼륨 그룹에 포함된 스냅샷 수
vg_attr	attr	볼륨 그룹의 상태: (w)iteable, (r)eadonly, resi(z)eable, e(x)ported, (p)artial 및 (c)lustered.
vg_extent_count	#ext	볼륨 그룹의 물리 확장 영역 수
vg_extent_size	ext	볼륨 그룹의 물리 확장 영역의 크기
vg_fmt	FMT	볼륨 그룹의 메타데이터 형식(**lvm2** 또는 **lvm1**)

인수	header	설명
vg_free	VFree	볼륨 그룹에 남아 있는 여유 공간의 크기
vg_free_count	free	볼륨 그룹의 사용 가능한 물리 확장 영역 수
vg_name	VG	볼륨 그룹 이름
vg_seqno	seq	볼륨 그룹 리버전을 나타내는 수
vg_size	VSize	볼륨 그룹의 크기
vg_sysid	SYS ID	LVM1 시스템 ID
vg_tags	VG 태그	볼륨 그룹에 연결된 LVM 태그
vg_uuid	VG UUID	볼륨 그룹의 UUID

***KnativeServing s** 명령은 기본적으로 다음 필드를 표시합니다. **first _name ,pv_ count,lv_count,snap_count, KnativeServing_attr, com_size,1/_free.** 디스플레이는 **HEALTH _name** 에 따라 정렬됩니다.*

```
# vgs
  VG     #PV #LV #SN Attr   VSize  VFree
  new_vg   3   1   1 wz--n- 51.42G 51.36G
```

***common s** 명령과 함께 **-v** 인수를 사용하여 기본 디스플레이(**total _extent_size ,1/_ uuid)**에 다음 필드를 추가합니다.*

```
# vgs -v
    Finding all volume groups
    Finding volume group "new_vg"
  VG     Attr   Ext   #PV #LV #SN VSize  VFree  VG UUID
  new_vg wz--n- 4.00M   3   1   1 51.42G 51.36G jxQJ0a-ZKk0-OpMO-0118-nlwO-wwqd-fD5D32
```

*표 7.3. "LVs 표시 필드" 헤더 디스플레이에 표시되는 필드 이름과 필드의 설명과 함께 **lvs** 명령의 표시 인수를 나열합니다.*

참고

Red Hat Enterprise Linux의 이후 릴리스에서 lvs 명령 출력은 출력의 추가 필드와 함께 다를 수 있습니다. 그러나 필드의 순서는 동일하게 유지되며 디스플레이 종료 시 추가 필드가 표시됩니다.

표 7.3. LVs 표시 필드

인수	header	설명
* **CHUNKSIZE** * **chunk_size**	청크	스냅샷 볼륨의 단위 크기
copy_percent	copy%	미러링된 논리 볼륨의 동기화 백분율; **pv_** hiera 명령을 사용하여 물리 확장 영역을 이동하는 경우에도 사용됩니다.
devices	devices	논리 볼륨을 구성하는 기본 장치(물리 볼륨, 논리 볼륨, 물리 확장 영역 및 논리 확장 영역 시작)
lv_ancestors	Collectionss	씬 풀 스냅샷의 경우 논리 볼륨의 RuntimeClass
lv_descendants	하위 항목	씬 풀 스냅샷의 경우 논리 볼륨의 하위 항목
lv_attr	attr	논리 볼륨의 상태입니다. 논리 볼륨 특성 비트는 다음과 같습니다. * 1 비트 1 볼륨 유형: (m)irrored, (m)irrored without initial sync, (o)rigin, (O)rigin with merge snapshot, (r)aid, ®aid without initial sync, (s)napshot, merge (S)napshot, (virtual) mirror 또는 raid (i)mage, mirror 또는 raid (I)mage out-of-sync, mirror (l)og device, under (c)onversion, thin (V)olume, (t)olume, (t)hin pool data, raid pool, raid or thin pool m(e)data metadata, spare, metadata metadata, and * 비트 2개 권한: (w)iteable, (r)ead-only, ®ead-only activation of non-read-only volume * 비트 3: 할당 정책: (a)ontiguous, (i)nherited, c(l)ing, (n)ormal. 이는 볼륨의 현재 할당 변경에 대해 잠긴 경우(예: **pvhiera 명령을 실행하는 동안) 대문자로 지정** 됩니다. * 비트 4: 고정 (m)inor * 비트 5: 상태: (a)ctive, (s)uspended, (I)nvalid snapshot, invalid (S)uspended snapshot, snapshot (m)erge failed, suspend snapshot (M)erge failed, mapped snapshot (d)evice present without tables, mapped device present with (i)active table. * 비트 6 비트 6: 장치 (o)pen

인수	header	설명
		* 비트 7: 대상 유형: (m)irror, (r)aid, (s)napshot, (t)hin, (u)nknown, (v)irtual. 이는 동일한 커널 대상과 관련된 논리 볼륨을 함께 그룹화합니다. 따라서 예를 들어, 미러 이미지, 미러 로그는 원래 장치 매퍼 미러 커널 드라이버를 사용하는 경우 (m)로 표시되는 반면 md raid 커널 드라이버를 사용하는 raid는 모두 (r)로 표시됩니다. 원래 device-mapper 드라이버를 사용하는 스냅샷은 (s)로 표시되는 반면 씬 프로비저닝 드라이버를 사용하는 씬 볼륨 스냅샷은 (t)로 표시됩니다. * 비트 8: 새로 할당된 데이터 블록은 사용하기 전에 (z)ero 블록으로 덮어씁니다. * 비트 9: Volume Health: (p)artial, (r)efresh needed, (m)matches exist, (w)itely. (p)artial은 이 논리 볼륨에서 사용하는 물리 볼륨 중 하나 이상이 시스템에서 누락되었음을 나타냅니다. (r)efresh는 시스템에서 하나 이상의 물리 볼륨이 누락되었음을 나타냅니다. (r)efresh는 쓰기 오류가 있음을 나타냅니다. (p)artial은 시스템에서 하나 이상의 물리 볼륨이 없음을 나타냅니다. (r)efresh는 쓰기 오류가 발생했습니다. 쓰기 오류는 물리 볼륨의 일시적인 오류 또는 실패의 표시로 인해 발생할 수 있습니다. 장치를 새로 고치거나 교체해야 합니다. (m)는 RAID 논리 볼륨에 일치하지 않는 배열의 일부가 있음을 나타냅니다. RAID 논리 볼륨에서 **검사** 작업을 시작하여 불일치가 검색됩니다. (스크립트 작업, **검사** 및 **복구**)는 **lvchange** 명령을 통해 RAID 논리 볼륨에서 수행할 수 있습니다. (w)ritely는 쓰기로 표시된 RAID 1 논리 볼륨의 장치를 나타냅니다. * bit 10: s(k)ip 활성화: 이 볼륨은 활성화 중에 건너뛸 수 있습니다.
lv_kernel_major	KMaj	논리 볼륨의 실제 주요 장치 번호 (비활성인 경우-1)
lv_kernel_minor	KMIN	논리 볼륨의 실제 마이너 장치 번호(비활성인 경우-1)
lv_major	Maj	논리 볼륨의 영구 주요 장치 번호 (지정되지 않은 경우--1)
lv_minor	min	논리 볼륨의 영구 마이너 장치 번호(지정되지 않은 경우-1)
lv_name	LV	논리 볼륨의 이름
lv_size	LSize	논리 볼륨의 크기
lv_tags	pause 태그	논리 볼륨에 연결된 LVM 태그
lv_uuid	LV UUID	논리 볼륨의 UUID입니다.
mirror_log	log	미러 로그가 상주하는 장치
modules	모듈	이 논리 볼륨을 사용하는 데 필요한 커널 장치 매퍼 대상
move_pv	move	**pvhiera 명령을 사용하여 생성된 임시 논리 볼륨의 물리 볼륨** 소스

인수	header	설명
origin	origin	스냅샷 볼륨의 원본 장치
* **regionsize** * **region_size**	리전	미러링된 논리 볼륨의 단위 크기
seg_count	#Seg	논리 볼륨의 세그먼트 수
seg_size	SSize	논리 볼륨의 세그먼트 크기
seg_start	시작	논리 볼륨의 세그먼트 오프셋
seg_tags	Seg 태그	논리 볼륨의 세그먼트에 연결된 LVM 태그
segtype	유형	논리 볼륨의 세그먼트 유형(예: mirror, striped, linear)
snap_percent	snap%	사용 중인 스냅샷 볼륨의 현재 백분율
스트라이프	#Str	논리 볼륨의 스트라이프 또는 미러 수
* **스트라이프 크기** * **stripe_size**	스트라이프	스트라이핑된 논리 볼륨의 스트라이프 단위 크기

***lvs** 명령은 기본적으로 다음 디스플레이를 제공합니다. 기본 디스플레이는 **volumes** 그룹 내의 **tekton_name** 및 **lv_name** 에 의해 정렬됩니다.*

```
# lvs
  LV     VG          Attr       LSize    Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  origin VG          owi-a-s---    1.00g
  snap   VG          swi-a-s---  100.00m      origin 0.00
```

***lvs** 명령을 사용하는 일반적인 용도는 논리 볼륨을 구성하는 기본 장치를 표시하기 위해 장치를 명령에 추가하는 것입니다. 이 예에서는 괄호로 묶은 논리 볼륨(예: **RAID** 미러)의 구성 요소인 내부 볼륨을 표시하는 **-a** 옵션도 지정합니다. 이 예제에는 **RAID** 볼륨, 스트라이핑된 볼륨, 씬-풀링된 볼륨이 포함되어 있습니다.*

```
# lvs -a -o +devices
  LV              VG          Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  raid1           VG          rwi-a-r---   1.00g                          100.00
raid1_rimage_0(0),raid1_rimage_1(0)
  [raid1_rimage_0] VG         iwi-aor---   1.00g                                  /dev/sde1(7041)
  [raid1_rimage_1] VG         iwi-aor---   1.00g                                  /dev/sdf1(7041)
  [raid1_rmeta_0]  VG         ewi-aor---   4.00m                                   /dev/sde1(7040)
```

```
  [raid1_rmeta_1]  VG          ewi-aor---   4.00m                                    /dev/sdf1(7040)
  stripe1          VG          -wi-a-----  99.95g
/dev/sde1(0),/dev/sdf1(0)
  stripe1          VG          -wi-a-----  99.95g                                    /dev/sdd1(0)
  stripe1          VG          -wi-a-----  99.95g                                    /dev/sdc1(0)
  [lvol0_pmspare]  rhel_host-083 ewi-------   4.00m                                    /dev/vda2(0)
  pool00           rhel_host-083 twi-aotz--  <4.79g              72.90  54.69
pool00_tdata(0)
  [pool00_tdata]   rhel_host-083 Twi-ao----  <4.79g                                    /dev/vda2(1)
  [pool00_tmeta]   rhel_host-083 ewi-ao----   4.00m
/dev/vda2(1226)
  root             rhel_host-083 Vwi-aotz--  <4.79g pool00       72.90
  swap             rhel_host-083 -wi-ao---- 820.00m                                    /dev/vda2(1227)
```

lvs 명령과 함께 -v 인수를 사용하면 기본 디스플레이에 다음 필드를 추가합니다. seg_count,lv_major, lv_kernel_major ,lv_kernel_major,lv_kernel_minor,lv_uuid.

```
# lvs -v
    Finding all logical volumes
  LV        VG     #Seg Attr   LSize  Maj Min KMaj KMin Origin Snap%  Move Copy%  Log Convert
LV UUID
  lvol0     new_vg    1 owi-a- 52.00M  -1  -1 253  3                                LBy1Tz-sr23-Ojsl-
LT03-nHLC-y8XW-EhCl78
  newvgsnap1 new_vg    1 swi-a-  8.00M  -1  -1 253  5    lvol0    0.20                1ye1OU-1clu-
o79k-20h2-ZGF0-qCJm-CfbsIx
```

lvs 명령의 --segments 인수를 사용하여 세그먼트 정보를 강조하는 기본 열에 정보를 표시할 수 있습니다. segments 인수를 사용하면 seg 접두사는 선택 사항입니다. lvs --segments 명령은 기본적으로 다음 필드를 표시합니다. lv_name, KnativeServing_name ,lv_ attr,스트라이프,segtype,seg_size. 기본 디스플레이는tekton _name, 볼륨 그룹 내의lv_name, 논리 볼륨 내의 seg_start 에 따라 정렬됩니다. 논리 볼륨이 조각화되면 이 명령의 출력에 다음이 표시됩니다.

```
# lvs --segments
  LV      VG         Attr   #Str Type   SSize
  LogVol00 VolGroup00 -wi-ao    1 linear  36.62G
  LogVol01 VolGroup00 -wi-ao    1 linear 512.00M
  lv      vg         -wi-a-    1 linear 104.00M
  lv      vg         -wi-a-    1 linear 104.00M
  lv      vg         -wi-a-    1 linear 104.00M
  lv      vg         -wi-a-    1 linear  88.00M
```

lvs --segments 명령과 함께 -v 인수를 사용하면 기본 디스플레이에 다음 필드가 추가됩니다. seg_start,스트라이프 size ,chunksize.

```
# lvs -v --segments
    Finding all logical volumes
  LV        VG     Attr   Start SSize  #Str Type   Stripe Chunk
  lvol0     new_vg owi-a-    0  52.00M    1 linear     0     0
  newvgsnap1 new_vg swi-a-    0   8.00M    1 linear     0  8.00K
```

*다음 예제에서는 하나의 논리 볼륨이 구성된 시스템에서 **lvs** 명령의 기본 출력과 **lvs** 인수가 지정된 **lvs** 명령의 기본 출력 을 보여줍니다.*

```
# lvs
  LV    VG     Attr   LSize  Origin Snap%  Move Log Copy%
  lvol0 new_vg -wi-a- 52.00M
# lvs --segments
  LV    VG     Attr   #Str Type   SSize
  lvol0 new_vg -wi-a-    1 linear 52.00M
```

7.3. LVM 보고서 정렬

*일반적으로 **lvs,tektons** 또는 **pvs** 명령의 전체 출력은 올바르게 정렬되고 정렬되기 전에 내부적으로 생성되어 저장해야 합니다. **--unbuffered** 인수를 지정하여 정렬되지 않은 출력을 생성하는 즉시 표시할 수 있습니다.*

*정렬할 다른 정렬된 열 목록을 지정하려면 보고 명령의 **-O** 인수를 사용합니다. 이러한 필드를 출력 자체 내에 포함할 필요는 없습니다.*

*다음 예제에서는 물리 볼륨 이름, 크기 및 여유 공간을 표시하는 **pvs** 명령의 출력을 보여줍니다.*

```
# pvs -o pv_name,pv_size,pv_free
  PV         PSize  PFree
  /dev/sdb1  17.14G 17.14G
  /dev/sdc1  17.14G 17.09G
  /dev/sdd1  17.14G 17.14G
```

다음 예제에서는 여유 공간 필드를 기준으로 정렬된 동일한 출력을 보여줍니다.

```
# pvs -o pv_name,pv_size,pv_free -O pv_free
  PV         PSize  PFree
  /dev/sdc1  17.14G 17.09G
  /dev/sdd1  17.14G 17.14G
  /dev/sdb1  17.14G 17.14G
```

다음 예제에서는 정렬된 필드를 표시할 필요가 없음을 보여줍니다.

```
# pvs -o pv_name,pv_size -O pv_free
  PV         PSize
  /dev/sdc1  17.14G
  /dev/sdd1  17.14G
  /dev/sdb1  17.14G
```

역방향 정렬을 표시하려면 **-O** 인수 뒤에 **-** 문자를 사용하여 지정하는 필드 앞에 해당합니다.

```
# pvs -o pv_name,pv_size,pv_free -O -pv_free
  PV         PSize  PFree
  /dev/sdd1  17.14G 17.14G
  /dev/sdb1  17.14G 17.14G
  /dev/sdc1  17.14G 17.09G
```

7.4. LVM 보고서 디스플레이의 단위 지정

LVM 보고서 디스플레이의 단위를 지정하려면 **report** 명령의 **--units** 인수를 사용합니다. **(b)ytes, (k)ilobytes, (m)egabytes, (g)igabytes, (t)erabytes, (e)xabytes, (p)etabytes,** 및 **(h)uman-readable-readable)**를 지정할 수 있습니다. 기본 디스플레이는 사람이 읽을 수 있습니다. **/etc/lvm/lvm.conf** 파일의 **global** 섹션에서 **units** 매개변수를 설정하여 기본값을 재정의할 수 있습니다.

다음 예제에서는 기본 기가가 아닌 **pvs** 명령의 출력을 메가바이트로 지정합니다.

```
# pvs --units m
  PV         VG     Fmt  Attr PSize     PFree
  /dev/sda1         lvm2 --   17555.40M 17555.40M
  /dev/sdb1  new_vg lvm2 a-   17552.00M 17552.00M
  /dev/sdc1  new_vg lvm2 a-   17552.00M 17500.00M
  /dev/sdd1  new_vg lvm2 a-   17552.00M 17552.00M
```

기본적으로 단위는 **2**의 **powers(24** 개)로 표시됩니다. 단위 사양**(B, K, M, G, T, H)**을 대문자로 지정하여 장치를 **1000** 단위로 표시하도록 지정할 수 있습니다.

다음 명령은 출력을 기본 동작인 **1024**의 배수로 표시합니다.

```
# pvs
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
  /dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
  /dev/sdd1  new_vg lvm2 a-   17.14G 17.14G
```

다음 명령은 출력을 **1000**의 여러 개로 표시합니다.

```
#  pvs --units G
  PV         VG     Fmt  Attr PSize  PFree
  /dev/sdb1  new_vg lvm2 a-   18.40G 18.40G
```

```
  /dev/sdc1  new_vg lvm2 a-   18.40G 18.35G
  /dev/sdd1  new_vg lvm2 a-   18.40G 18.40G
```

(512바이트로 정의됨) 또는 사용자 지정 단위를 지정할 수도 있습니다.

다음 예제에서는 **pvs** 명령의 출력을 여러 섹터로 표시합니다.

```
# pvs --units s
  PV         VG     Fmt  Attr PSize     PFree
  /dev/sdb1  new_vg lvm2 a-   35946496S 35946496S
  /dev/sdc1  new_vg lvm2 a-   35946496S 35840000S
  /dev/sdd1  new_vg lvm2 a-   35946496S 35946496S
```

다음 예제는 **pvs** 명령의 출력을 **4MB** 단위로 표시합니다.

```
# pvs --units 4m
  PV         VG     Fmt  Attr PSize    PFree
  /dev/sdb1  new_vg lvm2 a-   4388.00U 4388.00U
  /dev/sdc1  new_vg lvm2 a-   4388.00U 4375.00U
  /dev/sdd1  new_vg lvm2 a-   4388.00U 4388.00U
```

7.5. LVM 명령 출력을 JSON 형식으로 표시

LVM 표시 명령의 **--reportformat** 옵션을 사용하여 출력을 **JSON** 형식으로 표시할 수 있습니다.

다음 예제에서는 표준 기본 형식의 **lvs** 출력을 보여줍니다.

```
# lvs
  LV      VG          Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  my_raid my_vg       Rwi-a-r---  12.00m                                    100.00
  root    rhel_host-075 -wi-ao----   6.67g
  swap    rhel_host-075 -wi-ao---- 820.00m
```

다음 명령은 **JSON** 형식을 지정하는 경우 동일한 **LVM** 구성의 출력을 보여줍니다.

```
# lvs --reportformat json
  {
      "report": [
          {
              "lv": [
                  {"lv_name":"my_raid", "vg_name":"my_vg", "lv_attr":"Rwi-a-r---",
 "lv_size":"12.00m", "pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"",
```

```
"move_pv":"", "mirror_log":"", "copy_percent":"100.00", "convert_lv":""},
                {"lv_name":"root", "vg_name":"rhel_host-075", "lv_attr":"-wi-ao----",
"lv_size":"6.67g", "pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"",
"move_pv":"", "mirror_log":"", "copy_percent":"", "convert_lv":""},
                {"lv_name":"swap", "vg_name":"rhel_host-075", "lv_attr":"-wi-ao----",
"lv_size":"820.00m", "pool_lv":"", "origin":"", "data_percent":"", "metadata_percent":"",
"move_pv":"", "mirror_log":"", "copy_percent":"", "convert_lv":""}
            ]
        }
    ]
}
```

output_format 설정을 사용하여 **/etc/lvm/lvm.conf** 파일에서 보고서 형식을 구성 옵션으로 설정할 수도 있습니다. 그러나 명령줄의 **--reportformat** 설정은 이 설정보다 우선합니다.

7.6. LVM 명령 로그 표시

보고서 지향 및 처리 지향 **LVM** 명령 모두 **log/report_command_log** 구성 설정으로 사용하도록 설정된 경우 명령 로그를 보고할 수 있습니다. 표시할 필드 집합을 결정하고 이 보고서에 대해 정렬할 수 있습니다.

다음 예제는 **LVM** 명령에 대한 전체 로그 보고서를 생성하도록 **LVM**을 구성합니다. 이 예에서는 볼륨이 포함된 볼륨 그룹 **VG** 와 같이 논리 볼륨 **lvol0** 및 **lvol1** 이 성공적으로 처리되었는지 확인할 수 있습니다.

```
# lvmconfig --type full log/command_log_selection
command_log_selection="all"

# lvs
  Logical Volume
  ==============
  LV    LSize Cpy%Sync
  lvol1 4.00m 100.00
  lvol0 4.00m

  Command Log
  ===========
  Seq LogType Context    ObjType ObjName ObjGrp  Msg     Errno RetCode
    1 status  processing lv      lvol0   vg      success     0      1
    2 status  processing lv      lvol1   vg      success     0      1
    3 status  processing vg      vg              success     0      1

# lvchange -an vg/lvol1
  Command Log
  ===========
  Seq LogType Context    ObjType ObjName ObjGrp  Msg     Errno RetCode
    1 status  processing lv      lvol1   vg      success     0      1
    2 status  processing vg      vg              success     0      1
```

***LVM** 보고서 및 명령 로그 구성에 대한 자세한 내용은 **lvmreport man** 페이지를 참조하십시오.*

8장. RAID 논리 볼륨 구성

LVM RAID 볼륨을 생성, 활성화, 변경, 제거, 표시 및 사용할 수 있습니다.

8.1. RAID 논리 볼륨

LVM은 RAID 수준 0, 1, 4, 5, 6, 10을 지원합니다.

LVM RAID 볼륨에는 다음과 같은 특징이 있습니다.

- LVM에서 생성 및 관리하는 RAID 논리 볼륨은 여러 장치(MD) 커널 드라이버를 활용합니다.
- RAID1 이미지를 배열에서 임시로 분할하고 나중에 다시 배열에 병합할 수 있습니다.
- LVM RAID 볼륨은 스냅샷을 지원합니다.

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 볼륨을 생성할 때 LVM은 배열의 모든 데이터 또는 패리티 하위 볼륨에 대해 크기가 1인 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(두 개의 암시적 패리티 장치 추가) RAID4를 생성하면 4개의 메타데이터 하위 볼륨이 생성됩니다 (lv_rmeta_0,lv_rmeta_1,lv_rmeta_2, 및 lv_rmeta_3) 및 4 데이터 하위 볼륨(lv_rimage_0, lv_rmeta_ 3) 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 손상을 완화하거나 방지하는 데 도움이 됩니다. 소프트 손상 및 RAID LV에 무결성을 추가하는 방법에 대한 자세한 내용은 8.6절. “RAID LV에서 DM 무결성 사용” 을 참조하십시오.

8.2. RAID 수준 및 선형 지원

RAID는 0, 1, 4, 5, 6, 10 및 선형 수준을 포함하여 다양한 구성을 지원합니다. 이러한 RAID 유형은 다음과 같이 정의됩니다.

수준 0

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

대부분의 RAID 수준 0 구현에서는 멤버 장치에서 배열에서 가장 작은 장치 크기까지만 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 용량은 하드웨어 RAID에서 가장 작은 멤버 디스크 용량 또는 어레이의 디스크 또는 파티션 수를 곱한 소프트웨어 RAID에서 가장 작은 멤버 파티션의 용량과 동일합니다.

수준 1

RAID 수준 1 또는 미러링 은 배열의 각 멤버 디스크에 동일한 데이터를 작성하여 각 디스크에 "미러" 사본을 남겨 두어 중복성을 제공합니다. 미러링은 단순성과 높은 수준의 데이터 가용성으로 인해 널리 사용됩니다. 레벨 1은 두 개 이상의 디스크로 작동하며 매우 우수한 데이터 안정성을 제공하고 읽기 집약적인 애플리케이션의 성능을 개선하지만 비교적 높은 비용으로 작동합니다.

RAID 수준 1은 배열의 모든 디스크에 동일한 정보를 쓰고, 데이터 안정성을 제공하지만 수준 5와 같은 패리티 기반 RAID 레벨보다 훨씬 적은 공간 효율적인 방식으로 높은 비용으로 제공됩니다. 그러나 이 공간 비효율성에는 성능 이점이 있습니다. 패리티 기반 RAID 수준은 패리티를 생성하기 위해 훨씬 더 많은 CPU 성능을 소비하지만 RAID 수준 1은 단순히 CPU 오버헤드가 거의 없는 여러 RAID 멤버에 동일한 데이터를 한 번 이상 쓰는 것입니다. 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.

레벨 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 트랜잭션에서 고유한 병목 현상을 나타내기 때문에 레벨 4는 쓰기 캐싱과 같은 기술이나 시스템 관리자가 이러한 병목 현상을 염두에 두고 소프트웨어 RAID 장치를 의도적으로 설계하지 않는 특정 상황에서 사용되지 않습니다(예: 어레이가 데이터로 채워지지 않으면 쓰기가 거의 없는 배열과 같은 배열). RAID 수준 4는 너무 드물게 사용되므로 Anaconda에서 옵션으로 사용할 수 없습니다. 그러나 실제로 필요한 경우 사용자가 수동으로 만들 수 있습니다.

하드웨어 RAID 레벨 4의 스토리지 용량은 가장 작은 멤버 파티션 수에 1 을 곱한 용량과 동일합니다. RAID 수준 4 배열의 성능은 항상 symmetrical이며, 즉, outperform 쓰기를 읽습니다. 이는 쓰기가 패리티를 생성할 때 추가 CPU 및 기본 메모리 대역폭을 소비한 다음 데이터뿐만 아니라 패리티를 쓰기 때문에 실제 데이터를 디스크에 작성할 때 추가 버스 대역폭을 소비하기 때문입니다. 읽기에는 배열이 저하된 상태에 있지 않으면 데이터를 읽고 패리티가 아닙니다.Reads need only the data and not the parity unless the array is in a degraded state. 결과적으로 읽기는 일반 운영 조건에서 동일한 양의 데이터 전송을 위해 드라이브 및 컴퓨터의 버스를 통해 적은 트래픽을 생성합니다.

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 CPU 및 소프트웨어 RAID를 사용하면 최신 CPU가 매우 빠르게 패리티를 생성할 수 있기 때문에 일반적으로 병목 현상이 발생하지 않습니다. 그러나 소프트웨어 RAID5 배열에 충분히 많은 멤버 장치가 있는 경우 모든 장치에서 집계 데이터 전송 속도가 충분히 높으면 이러한 병목 현상이 발생하기 시작할 수 있습니다.

수준 4와 마찬가지로 수준 5는 symmetrical 성능을 제공하며 성능이 크게 기록됩니다. 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 수준 6 배열의 총 용량은 RAID 수준 5 및 4와 유사하게 계산됩니다. 단, 추가 패리티 스토리지 공간에 대해 장치 수에서 2개의 장치를 뺀다.

수준 10

이 RAID 수준은 수준 0의 성능 이점과 수준 1의 중복성을 결합하려고 시도합니다. 또한 2 개 이상의 장치로 레벨 1 어레이에서 낭비되는 공간 중 일부를 완화하는 데 도움이 됩니다. 레벨 10에서는 예를 들어 각 데이터의 사본 2개만 저장하도록 구성된 3라이브 어레이를 생성할 수 있으며, 이 경우 전체 배열 크기가 가장 작은 장치보다 1.5배가 될 수 있습니다(예: 3device, 레벨 1 어레이와 동일). 이렇게 하면 CPU 프로세스 사용량이 RAID 수준 6과 같은 패리티를 계산하지 않지만 공간 효율은 줄어듭니다.

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 없습니다. 용량은 모든 멤버 디스크의 합계입니다.

8.3. LVM RAID 세그먼트 유형

RAID 논리 볼륨을 생성하려면 raid 유형을 lvcreate 명령의 --type 인수로 지정합니다. 다음 표에서는 가능한 RAID 세그먼트 유형을 설명합니다.

대부분의 사용자의 경우 사용 가능한 5개의 기본 유형(raid4,raid 5,raid6,raid10) 중 하나를 지정해야 합니다.

표 8.1. LVM RAID 세그먼트 유형

세그먼트 유형	설명
raid1	RAID1 미러링. 이는 **-m** 을 지정할 때 **lvcreate** 명령의 **--type** 인수의 기본 값이지만 striping을 지정하지 않습니다.
raid4	RAID4 전용 패리티 디스크
raid5	**raid5_ls**와 동일합니다.
raid5_la	• RAID5를 남겨 둡니다. • 데이터 연속으로 회전 패리티 0

세그먼트 유형	설명
raid5_ra	• RAID5 오른쪽metric. • 데이터 연속으로 순환 패리티 N
raid5_ls	• RAID5에서 대칭을 남겨 둡니다. • 데이터 재시작을 통해 패리티 0 회전
raid5_rs	• RAID5 올바른 대칭입니다. • 데이터 재시작으로 패터레이션 N 순환
raid6	**raid6_zr**와 동일합니다.
raid6_zr	• RAID6 0 재시작 • 데이터 재시작을 통해 패터리 0(left-to-right) 교체
raid6_nr	• RAID6 N 재시작 • 데이터 재시작으로 패터링 N(left-to-right) 교체
raid6_nc	• RAID6 N 계속 • 데이터 연속으로 회전 패터레이션 N(left-to-right)
raid10	• 스트라이핑된 미러입니다. **-m** 을 지정하고 1보다 큰 스트라이프 수를 지정하면 **lvcreate** 명령의 **--type** 인수의 기본값입니다. • 미러 세트 제거
raid0/raid0_meta	스트라이핑. RAID0은 스트라이프 크기 단위로 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 분산합니다. 이는 성능을 향상시키는 데 사용됩니다. 데이터 하위 볼륨이 실패하면 논리 볼륨 데이터가 손실됩니다.

8.4. RAID 논리 볼륨 생성

*이 섹션에서는 다양한 유형의 **RAID** 논리 볼륨을 생성하는 예제 명령을 제공합니다.*

-m 인수에 지정한 값에 따라 다양한 사본을 사용하여 RAID1 배열을 생성할 수 있습니다. 마찬가지로 -i 인수를 사용하여 RAID 4/5/6 논리 볼륨의 스트라이프 수를 지정합니다. -I 인수를 사용하여 스트라이프 크기를 지정할 수도 있습니다.

다음 명령은 1GB 크기의 볼륨 그룹 my_lv 에 my_ lv라는 2방향 RAID1 배열을 생성합니다.

```
# lvcreate --type raid1 -m 1 -L 1G -n my_lv my_vg
```

다음 명령은 1GB 크기의 볼륨 그룹 my_lv 에 my_lv라는 RAID5 배열(3 스트라이프 + 1개의 암시적 패리티 드라이브)을 생성합니다. LVM 스트라이핑 볼륨과 마찬가지로 스트라이프 수를 지정합니다. 올바른 패리티 드라이브 수는 자동으로 추가됩니다.

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

다음 명령은 1GB 크기의 볼륨 그룹 my_lv 에 my_lv라는 RAID6 배열(3 스트라이프 + 2개의 암시적 패리티 드라이브)을 생성합니다.

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

8.5. RAID0(STRIPED) 논리 볼륨 생성

RAID0 논리 볼륨은 여러 데이터 하위 볼륨에 논리 볼륨 데이터를 스트라이프 크기 단위로 분산합니다.

RAID0 볼륨을 생성하는 명령의 형식은 다음과 같습니다.

```
lvcreate --type raid0[_meta] --stripes Stripes --stripesize StripeSize VolumeGroup
[PhysicalVolumePath ...]
```

표 8.2. RAID0 명령 생성 매개변수

매개변수	설명

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 무해하기 때문에 미러링된 데이터 블록을 RAID1/10으로 저장하거나 패리티 블록을 RAID4/5/6으로 계산하고 저장할 필요가 없습니다. 따라서 미러링되거나 패리티 블록의 동기화 진행 상태를 유지하기 위해 메타데이터 볼륨이 필요하지 않습니다. 그러나 메타데이터 볼륨은 RAID0에서 RAID4/5/6/10으로 변환하고 각 할당 실패를 방지하기 위해 **raid0_meta** 를 미리 할당하도록 해당 메타데이터 볼륨을 미리 할당해야 합니다.
--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.
VolumeGroup	사용할 볼륨 그룹을 지정합니다.
PhysicalVolumePath ...	사용할 장치를 지정합니다. 이 값을 지정하지 않으면 LVM에서 *Stripes* 옵션에 지정된 장치 수를 선택합니다.

이 예제 절차에서는 /dev/sda1,/dev/sdb1, /dev/sdc1 의 디스크에서 데이터를 스트라이프하는 mylv 라는 LVM RAID0 논리 볼륨을 생성합니다.

1. ***볼륨 그룹에서 사용할 디스크에 pvcreate 명령을 사용하여 LVM 물리 볼륨으로 레이블을 지정합니다.***

주의

이 명령은 /dev/sda1,/dev/sdb1, /dev/sdc1 의 모든 데이터를 삭제합니다.

```
# pvcreate /dev/sda1 /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sda1" successfully created
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdc1" successfully created
```

2.

*볼륨 그룹 **myvg** 를 생성합니다. 다음 명령은 볼륨 그룹 **myvg** 를 생성합니다.*

```
# vgcreate myvg /dev/sda1 /dev/sdb1 /dev/sdc1
  Volume group "myvg" successfully created
```

***KnativeServing s** 명령을 사용하여 새 볼륨 그룹의 속성을 표시할 수 있습니다.*

```
# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  myvg   3   0   0 wz--n- 51.45G 51.45G
```

3.

*생성한 볼륨 그룹에서 **RAID0** 논리 볼륨을 생성합니다. 다음 명령은 볼륨 그룹 **myvg** 에서 **RAID0** 볼륨 **mylv** 를 생성합니다. 이 예에서는 스트라이프 **3**개와 스트라이프 크기가 **4**킬로바이트인 **2GB** 크기의 논리 볼륨을 생성합니다.*

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

4.

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

```
# mkfs.ext4 /dev/myvg/mylv
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 525312 4k blocks and 131376 inodes
Filesystem UUID: 9d4c0704-6028-450a-8b0a-8875358c0511
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

다음 명령은 논리 볼륨을 마운트하고 파일 시스템 디스크 사용량을 보고합니다.

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

8.6. RAID LV에서 DM 무결성 사용

시스템 관리자는 **RAID LV**와 함께 장치 매퍼(**DM**) 무결성을 사용하여 소프트 손상 또는 비트 로트로 인한 데이터 손실 위험을 최소화할 수 있습니다.

8.6.1. 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 **RAID(Redundant Array of Independent Disks) LV**는 장치가 실패할 때 데이터 손실을 방지합니다. **RAID** 배열을 포함하는 장치가 실패하면 해당 **RAID LV**의 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 무결성은 소프트 손상으로 인한 데이터 손실을 완화하거나 방지하기 위해 **RAID** 수준 **1, 4, 5, 6, 10**과 함께 사용됩니다. **RAID** 계층을 사용하면 데이터의 손상되지 않은 사본이 소프트 손상 오류를 수정할 수 있습니다. 무결성 계층은 각 **RAID** 이미지 위에 배치되고 추가 하위 **LV**는 각 **RAID** 이미지에 대한 무결성 메타데이터(데이터 체크섬)를 저장합니다. 무결성이 있는 **RAID LV**에서 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 무결성으로 새 **RAID LV**를 생성하거나 기존 **RAID LV**에 무결성을 추가하는 경우 다음을 고려하십시오.

- 무결성 메타데이터에는 추가 스토리지 공간이 필요합니다. 각 **RAID** 이미지에 대해 **500MB**의 데이터에는 데이터에 추가되는 체크섬으로 인해 **4MB**의 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.
- 비트맵 또는 저널의 두 가지 무결성 모드를 사용할 수 있습니다. 비트맵 무결성 모드는 일반적으로 저널 모드보다 더 나은 성능을 제공합니다.

작은 정보

*성능 문제가 발생하는 경우 무결성과 함께 **RAID1**을 사용하거나 특정 **RAID** 구성의 성능을 테스트하여 요구 사항을 충족하는지 확인하는 것이 좋습니다.*

8.6.2. DM 무결성으로 RAID LV 생성

RAID LV를 생성할 때 DM 무결성을 추가하면 소프트 손상으로 인한 데이터 손실 위험을 줄이는 데 도움이 됩니다.

사전 요구 사항

- *root 액세스 권한이 있어야 합니다.*

절차

- *DM 무결성으로 RAID LV를 생성하려면 다음을 수행합니다.*

```
# lvcreate --type <raid-level> --raidintegrity y -L <usable-size> -n <logical-volume> <volume-group>
```

다음과 같습니다.

<raid-level>

생성할 RAID LV의 RAID 수준을 지정합니다.

<usable-size>

사용 가능한 크기(MB)를 지정합니다.

<logical-volume>

생성할 LV의 이름을 지정합니다.

<volume-group>

에서 RAID LV를 생성할 볼륨 그룹의 이름을 지정합니다.

다음 예에서는 사용 가능한 크기가 **256M** 및 **RAID** 수준 **1**인 **test- lv** 라는 무결성이 있는 **RAID LV**를 만듭니다.

무결성이 있는 **RAID LV**의 예

```
# lvcreate --type raid1 --raidintegrity y -L256M -n test-lv test-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.
```

8.6.3. 기존 RAID LV에 DM 무결성 추가

소프트 손상으로 인한 데이터 손실 위험을 줄이기 위해 기존 **RAID LV**에 **DM** 무결성을 추가할 수 있습니다.

사전 요구 사항

- **root** 액세스 권한이 있어야 합니다.

절차

- 기존 **RAID LV**에 **DM** 무결성을 추가하려면 다음을 수행합니다.

```
# lvconvert --raidintegrity y <volume-group>/<logical-volume>
```

다음과 같습니다.

<volume-group>

에서 **RAID LV**를 생성할 볼륨 그룹의 이름을 지정합니다.

<logical-volume>

생성할 **LV**의 이름을 지정합니다.

8.6.4. RAID LV에서 무결성 제거

RAID LV에 무결성을 추가하면 해당 **RAID LV**에서 수행할 수 있는 작업 수가 제한됩니다. 따라서 특정 작업을 수행하기 전에 무결성을 제거해야 합니다.

사전 요구 사항

- **root** 액세스 권한이 있어야 합니다.

절차

- **RAID LV**에서 무결성을 제거하려면 다음을 수행합니다.

```
# lvconvert --raidintegrity n <volume-group>/<logical-volume>
```

다음과 같습니다.

<volume-group>

에서 **RAID LV**를 생성할 볼륨 그룹의 이름을 지정합니다.

<logical-volume>

생성할 **LV**의 이름을 지정합니다.

8.6.5. DM 무결성 정보 보기

무결성이 있거나 기존 **RAID LV**에 무결성을 추가할 때 다음 명령을 사용하여 무결성에 대한 정보를 확인합니다.

```
# lvs -a <volume-group>
```

여기서 **<volume-group>** 은 무결성이 있는 **RAID LV**가 포함된 볼륨 그룹의 이름입니다.

다음 예에서는 **test-vg** 볼륨 그룹에서 생성된 **test-lv RAID LV**에 대한 정보를 보여줍니다.

```
# lvs -a test-vg
  LV                VG     Attr       LSize   Origin              Cpy%Sync
```

```
test-lv                 test-vg rwi-a-r--- 256.00m                      2.10
[test-lv_rimage_0]      test-vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 93.75
[test-lv_rimage_0_imeta] test-vg ewi-ao----   8.00m
[test-lv_rimage_0_iorig] test-vg -wi-ao---- 256.00m
[test-lv_rimage_1]      test-vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 85.94
[test-lv_rimage_1_imeta] test-vg ewi-ao----   8.00m
[test-lv_rimage_1_iorig] test-vg -wi-ao---- 256.00m
[test-lv_rmeta_0]       test-vg ewi-aor---   4.00m
[test-lv_rmeta_1]       test-vg ewi-aor---   4.00m
```

동기화

*무결성이 있는 **RAID LV**를 생성하거나 기존 **RAID LV**에 무결성을 추가하는 경우, **LV**를 사용하기 전에 무결성 동기화 및 **RAID** 메타데이터가 완료될 때까지 기다리는 것이 좋습니다. 그러지 않으면 백그라운드 초기화가 **LV**의 성능에 영향을 미칠 수 있습니다. **Cpy%Sync** 열은 최상위 **RAID LV**와 각 **RAID** 이미지에 대한 동기화 진행 상황을 나타냅니다. **RAID** 이미지는 **LV** 열에 **raid_image_N** 으로 표시됩니다. 최상위 **RAID LV** 및 각 **RAID** 이미지에 대해 동기화 진행률이 **100%** 표시되는지 확인합니다.*

*무결성을 사용하는 **RAID** 이미지*

***Attr** 열에 나열된 속성의 **g** 속성은 **RAID** 이미지가 무결성을 사용하고 있음을 나타냅니다. 무결성 체크섬은 **_imeta RAID LV**에 저장됩니다.*

*각 **RAID LV**의 유형을 표시하려면 **lvs** 명령에 **-o+segtype** 옵션을 추가합니다.*

```
# lvs -a my-vg -o+segtype
  LV                  VG      Attr       LSize   Origin                Cpy%Sync Type
  test-lv             test-vg rwi-a-r--- 256.00m                        87.96    raid1
  [test-lv_rimage_0]      test-vg gwi-aor--- 256.00m [test-lv_rimage_0_iorig] 100.00   integrity
  [test-lv_rimage_0_imeta] test-vg ewi-ao----   8.00m                             linear
  [test-lv_rimage_0_iorig] test-vg -wi-ao---- 256.00m                             linear
  [test-lv_rimage_1]      test-vg gwi-aor--- 256.00m [test-lv_rimage_1_iorig] 100.00   integrity
  [test-lv_rimage_1_imeta] test-vg ewi-ao----   8.00m                             linear
  [test-lv_rimage_1_iorig] test-vg -wi-ao---- 256.00m                             linear
  [test-lv_rmeta_0]       test-vg ewi-aor---   4.00m                           linear
  [test-lv_rmeta_1]       test-vg ewi-aor---   4.00m                           linear
```

무결성 불일치

*각 **RAID** 이미지에서 탐지된 불일치 수를 계산하는 증분 카운터가 있습니다. 특정 **RAID** 이미지에서 무결성으로 감지된 데이터 불일치를 보려면 다음 명령을 실행합니다.*

 # lvs -o+integritymismatches <volume-group>/<logical-volume>_raid-image_<n>

다음과 같습니다.

<volume-group>

에서 **RAID LV**를 생성할 볼륨 그룹의 이름을 지정합니다.

<logical-volume>

생성할 **LV**의 이름을 지정합니다.

<n>

무결성 불일치 정보를 볼 **RAID** 이미지를 지정합니다.

보려는 각 **RAID** 이미지에 대해 명령을 실행해야 합니다. 다음 예에서는 **test-vg/test-lv** 에서 **rimage_0** 의 데이터 불일치를 확인합니다.

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

무결성이 데이터 불일치를 감지하지 못했으므로 **IntegMismatches** 카운터는 **0(0)**을 표시합니다.

커널 메시지 로그에서 무결성 불일치

다음 예와 같이 커널 메시지 로그에서 데이터 무결성 정보를 찾을 수도 있습니다.

커널 메시지 로그와의 **dm-integrity** 불일치의 예

```
device-mapper: integrity: dm-12: Checksum failed at sector 0x24e7
```

커널 메시지 로그의 **dm-integrity** 데이터 수정 예

```
md/raid1:mdX: read error corrected (8 sectors at 9448 on dm-16)
```

8.6.6. 추가 리소스

- 사용 가능한 모든 옵션에 대한 자세한 내용은 **lvmraid** 명령 도움말 페이지를 참조하십시오.

8.7. RAID 볼륨이 초기화되는 속도 제어

RAID10 논리 볼륨을 생성할 때 동기화 작업으로 논리 볼륨을 초기화하는 데 필요한 백그라운드 I/O는 볼륨 그룹 메타데이터 업데이트, 특히 많은 RAID 논리 볼륨을 생성할 때와 같은 다른 I/O 작업을 LVM 장치에 분산시킬 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다.

복구 제한을 구현하여 RAID 논리 볼륨이 초기화되는 속도를 제어할 수 있습니다. lvcreate 명령의 --minrecoveryrate 및 --maxrecoveryrate 옵션을 사용하여 해당 작업에 대해 최소 및 최대 I/O 비율을 설정하여 동기화 작업을 수행하는 속도를 제어합니다. 이러한 옵션을 다음과 같이 지정합니다.

- --maxrecoveryrate Rate[bBsSkKmMgG]

  nominal I/O 작업이 중복되지 않도록 RAID 논리 볼륨의 최대 복구 속도를 설정합니다. Rate 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 지정되지 않은 경우 kiB/sec/device 로 가정합니다. 복구 비율을 0으로 설정하면 바인딩 해제됩니다.

- --minrecoveryrate Rate[bBsSkKmMgG]

  RAID 논리 볼륨의 최소 복구 속도를 설정하여 동기화 작업에 대한 I/O가 높은 경우 I/O가 과도한 I/O가 있는 경우에도 최소 처리량을 달성하도록 합니다. Rate 는 배열의 각 장치에 대한 초당 양으로 지정됩니다. 접미사가 지정되지 않은 경우 kiB/sec/device로 가정합니다.

다음 명령은 최대 복구 속도 128 kiB/sec/device를 사용하여 크기가 10GB인 2방향 RAID10 배열을 생성합니다. 배열의 이름은 my_lv 이며 볼륨 그룹 my_vg 에 있습니다.

```
# lvcreate --type raid10 -i 2 -m 1 -L 10G --maxrecoveryrate 128 -n my_lv my_vg
```

RAID 스크럽 작업에 대해 최소 및 최대 복구 속도를 지정할 수도 있습니다.

8.8. 선형 장치를 RAID 장치로 변환

lvconvert 명령의 --type 인수를 사용하여 기존 선형 논리 볼륨을 RAID 장치로 변환할 수 있습니다.

다음 명령은 볼륨 그룹 my_vg 의 선형 논리 볼륨 my_lv 를 2방향 RAID1 배열로 변환합니다.

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
```

RAID 논리 볼륨은 메타데이터 및 데이터 하위 볼륨 쌍으로 구성되므로 선형 장치를 RAID1 배열로 변환할 때 새 메타데이터 하위 볼륨이 생성되고 선형 볼륨이 있는 동일한 물리 볼륨(하나)의 원래 논리 볼륨과 연결됩니다. 추가 이미지는 metadata/data 하위 볼륨 쌍에 추가됩니다. 예를 들어, 원본 장치가 다음과 같은 경우:

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

2방향 RAID1 배열로 변환한 후 장치에는 다음 데이터 및 메타데이터 하위 볼륨 쌍이 포함됩니다.

```
# lvconvert --type raid1 -m 1 my_vg/my_lv
# 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)
```

원래 논리 볼륨과 쌍을 사용하는 메타데이터 이미지를 동일한 물리 볼륨에 배치할 수 없는 경우 lvconvert 가 실패합니다.

8.9. LVM RAID1 논리 볼륨을 LVM 선형 논리 볼륨으로 변환

-m0 인수를 지정하여 기존 RAID1 LVM 논리 볼륨을 lvconvert 명령을 사용하여 LVM 선형 논리 볼륨으로 변환할 수 있습니다. 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.

다음 예제는 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)
```

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

```
# lvconvert -m0 my_vg/my_lv
# lvs -a -o name,copy_percent,devices my_vg
  LV     Copy%  Devices
  my_lv         /dev/sde1(1)
```

LVM RAID1 논리 볼륨을 LVM 선형 볼륨으로 변환할 때 제거할 물리 볼륨을 지정할 수 있습니다. 다음 예제에서는 /dev/sda1 및 /dev/sdb1 이라는 두 개의 이미지로 구성된 LVM RAID1 논리 볼륨의 레이아웃을 보여줍니다. 이 예에서 lvconvert 명령은 /dev/sda1 를 제거하여 선형 장치를 구성하는 물리 볼륨으로 /dev/sdb1 을 유지하도록 지정합니다.

```
# 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/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
# lvconvert -m0 my_vg/my_lv /dev/sda1
# lvs -a -o name,copy_percent,devices my_vg
  LV    Copy%  Devices
  my_lv       /dev/sdb1(1)
```

8.10. 미러링된 LVM 장치를 RAID1 장치로 변환

--type raid1 인수를 지정하여 lvconvert 명령을 사용하여 세그먼트 유형의 미러 를 사용하여 미러링된 기존 LVM 장치를 RAID1 LVM 장치로 변환할 수 있습니다. 이렇게 하면 mirror subvolumes(mimage)가 RAID 하위 볼륨(rimage)으로 변경됩니다. 또한 미러 로그가 제거되고 해당 데이터 하위 볼륨과 동일한 물리 볼륨의 데이터 하위 볼륨에 대해 메타데이터 하위 볼륨(rmeta)이 생성됩니다.

다음 예에서는 미러링된 논리 볼륨 my_vg/my_lv 의 레이아웃을 보여줍니다.

```
# 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)
```

다음 명령은 미러링된 논리 볼륨 my_vg/my_lv 를 RAID1 논리 볼륨으로 변환합니다.

```
# lvconvert --type raid1 my_vg/my_lv
# 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)
```

8.11. RAID 논리 볼륨 크기 조정

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. 스트라이핑된 **RAID** 논리 볼륨의 경우 스트라이핑된 **RAID** 논리 볼륨을 생성할 때와 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 **striped RAID** 논리 볼륨을 생성할 때와 동일한 스트라이프 반올림 제약 조건이 적용됩니다.

- **lvconvert** 명령의 **--stripes N** 매개 변수를 사용하여 스트라이핑된 **RAID** 논리 볼륨 **(raid4/5/6/10)**의 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 동일한 **RAID** 수준을 유지하면서 **RAID** 논리 볼륨의 속성을 변경할 수 있는 **RAID** 복구 기능의 일부입니다. **RAID** 논리 볼륨을 재작업하기 위해 **lvconvert** 명령을 사용하는 **RAID** 재**shaping** 및 예제에 대한 자세한 내용은 **lvmraid(7)** 매뉴얼 페이지를 참조하십시오.

8.12. 기존 RAID1 장치의 이미지 수 변경

LVM 미러링의 이전 구현에서 이미지 수를 변경할 수 있는 것처럼 기존 **RAID1** 배열의 이미지 수를 변경할 수 있습니다. **lvconvert** 명령을 사용하여 추가하거나 제거할 추가 **metadata**/**data** 하위 볼륨 쌍 수를 지정합니다.

lvconvert 명령을 사용하여 **RAID1** 장치에 이미지를 추가하는 경우 결과 장치의 총 이미지 수를 지정하거나 장치에 추가할 이미지 수를 지정할 수 있습니다. 선택적으로 새 메타데이터/데이터 이미지 쌍이 상주하는 물리 볼륨을 지정할 수도 있습니다.

metadata 하위 볼륨(이름 **rmeta**)은 항상 데이터 하위 볼륨 **rimage**와 동일한 물리 장치에 존재합니다. **metadata**/**data** 하위 볼륨 쌍은 **RAID** 배열의 다른 **metadata**/**data** 하위 볼륨 쌍의 다른 메타데이터/데이터 하위 볼륨 쌍과 동일한 물리 볼륨에 생성되지 않습니다(**--alloc**를 지정하지 않는 경우).

RAID1 볼륨에 이미지를 추가하는 명령의 형식은 다음과 같습니다.

```
lvconvert -m new_absolute_count vg/lv [removable_PVs]
lvconvert -m +num_additional_images vg/lv [removable_PVs]
```

예를 들어 다음 명령은 2방향 RAID1 배열인 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)
```

다음 명령은 2방향 RAID1 장치 my_vg/my_lv 를 3-way RAID1 장치로 변환합니다.

```
# lvconvert -m 2 my_vg/my_lv
# 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_2(0)
  [my_lv_rimage_0]       /dev/sde1(0)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rimage_2]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sde1(256)
  [my_lv_rmeta_1]        /dev/sdf1(0)
  [my_lv_rmeta_2]        /dev/sdg1(0)
```

RAID1 배열에 이미지를 추가할 때 이미지에 사용할 물리 볼륨을 지정할 수 있습니다. 다음 명령은 2방향 RAID1 장치 my_vg/my_lv 를 3방향 RAID1 장치로 변환하여 배열에 물리 볼륨 /dev/sdd1 을 사용하도록 지정합니다.

```
# lvs -a -o name,copy_percent,devices my_vg
  LV              Copy%  Devices
  my_lv           56.00 my_lv_rimage_0(0),my_lv_rimage_1(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
# lvconvert -m 2 my_vg/my_lv /dev/sdd1
# 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_2(0)
  [my_lv_rimage_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rimage_2]       /dev/sdd1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
  [my_lv_rmeta_2]        /dev/sdd1(0)
```

RAID1 배열에서 이미지를 제거하려면 다음 명령을 사용합니다. lvconvert 명령을 사용하여 RAID1 장치에서 이미지를 제거하는 경우 결과 장치에 대한 총 이미지 수를 지정하거나 장치에서 제거할 이미지 수를 지정할 수 있습니다. 장치를 제거할 물리 볼륨을 선택적으로 지정할 수도 있습니다.

```
lvconvert -m new_absolute_count vg/lv [removable_PVs]
lvconvert -m -num_fewer_images vg/lv [removable_PVs]
```

또한 이미지 및 관련 메타데이터 하위 볼륨이 제거되면 더 많은 수의 이미지가 슬롯을 채우기 위해 축소됩니다. lv_rimage_0 , lv_rimage_1, lv_rimage_1 , lv_rimage_0 및 lv_rimage_0 및 lv_rimage_0 및 lv_rimage_ 0으로 구성된 3way RAID1 배열에서lv_rimage_1 을 제거하면 lv_rimage_0 및 lv_rimage_0 및 lv_rimage_ 0으로 구성된 RAID1 배열이 생성됩니다. 하위 볼륨의 lv_rimage_2 는 이름이 되고 빈 슬롯을 초과하여 lv_rimage_1 이 됩니다.

다음 예제에서는 3방향 RAID1 논리 볼륨 my_vg/my_lv 의 레이아웃을 보여줍니다.

```
# 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/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rimage_2]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
  [my_lv_rmeta_2]        /dev/sdg1(0)
```

다음 명령은 3방향 RAID1 논리 볼륨을 2방향 RAID1 논리 볼륨으로 변환합니다.

```
# lvconvert -m1 my_vg/my_lv
# 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)
```

다음 명령은 3방향 RAID1 논리 볼륨을 2방향 RAID1 논리 볼륨으로 변환하여 제거할 이미지가 포함된 물리 볼륨을 /dev/sde1 로 지정합니다.

```
# lvconvert -m1 my_vg/my_lv /dev/sde1
# 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/sdf1(1)
```

```
  [my_lv_rimage_1]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sdf1(0)
  [my_lv_rmeta_1]        /dev/sdg1(0)
```

8.13. RAID 이미지를 별도의 논리 볼륨으로 분할

RAID 논리 볼륨의 이미지를 분할하여 새 논리 볼륨을 구성할 수 있습니다.

RAID 이미지를 분할하는 명령 형식은 다음과 같습니다.

```
lvconvert --splitmirrors count -n splitname vg/lv [removable_PVs]
```

기존 RAID1 논리 볼륨에서 RAID 이미지를 제거할 때와 마찬가지로 장치 중간에서 RAID 데이터 하위 볼륨(및 관련 메타데이터 하위 볼륨)을 제거하면 더 많은 번호가 지정된 이미지가 슬롯을 채우기 위해 전환됩니다. 따라서 RAID 배열을 구성하는 논리 볼륨의 인덱스 번호는 손상된 정수 시퀀스입니다.

참고

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

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

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

다음 예제에서는 3방향 RAID1 논리 볼륨인 my_lv 를 2방향 RAID1 논리 볼륨, my_lv, 선형 논리 볼륨, new로 분할합니다.

```
# 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/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rimage_2]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
  [my_lv_rmeta_2]        /dev/sdg1(0)
# lvconvert --splitmirror 1 -n new my_vg/my_lv
# 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)
  new                    /dev/sdg1(1)
```

8.14. RAID 이미지 분할 및 병합

lvconvert 명령의 **--splitmirrors** 인수와 함께 **--trackchanges** 인수를 사용하여 변경 사항을 추적하는 동안 읽기 전용용으로 **RAID1** 배열 이미지를 일시적으로 분할할 수 있습니다. 이렇게 하면 이미지가 분할된 이후 변경된 배열의 해당 부분만 다시 동기화하면서 나중에 이미지를 배열로 병합할 수 있습니다.

RAID 이미지를 분할하는 **lvconvert** 명령의 형식은 다음과 같습니다.

```
lvconvert --splitmirrors count --trackchanges vg/lv [removable_PVs]
```

track changes 인수를 사용하여 **RAID** 이미지를 분할하면 분할할 이미지를 지정할 수 있지만 분할할 볼륨 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.

이후 lvconvert 명령을 --merge 인수와 함께 실행하여 지정된 --trackchanges 인수와 함께 분리된 이미지를 병합할 수 있습니다. 이미지를 병합할 때 이미지가 분할된 이후 변경된 배열의 부분만 다시 동기화됩니다.

RAID 이미지를 병합하기 위한 lvconvert 명령의 형식은 다음과 같습니다.

```
lvconvert --merge raid_image
```

다음 예제에서는 RAID1 논리 볼륨을 생성한 다음 나머지 배열로 변경 사항을 추적하는 동안 해당 볼륨에서 이미지를 분할합니다.

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

다음 예제에서는 나머지 배열의 변경 사항을 추적하는 동안 RAID1 볼륨에서 이미지를 분할한 다음 볼륨을 다시 배열로 병합합니다.

```
# lvconvert --splitmirrors 1 --trackchanges my_vg/my_lv
  lv_rimage_1 split from my_lv for read-only purposes.
  Use 'lvconvert --merge my_vg/my_lv_rimage_1' to merge back into my_lv
# 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)
# lvconvert --merge my_vg/my_lv_rimage_1
  my_vg/my_lv_rimage_1 successfully merged back into my_vg/my_lv
# 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)
```

8.15. RAID 오류 정책 설정

LVM RAID는 lvm.conf 파일의 raid_fault_policy 필드에 의해 정의된 기본 설정에 따라 자동 방식으로 장치 오류를 처리합니다.

- raid_fault_policy 필드가 allocate 로 설정된 경우 시스템은 볼륨 그룹의 예비 장치로 오류가 발생한 장치를 대체하려고 합니다. 사용 가능한 예비 장치가 없으면 시스템 로그에 보고됩니다.

- raid_fault_policy 필드가 warn 으로 설정되면 시스템에서 경고가 생성되고 로그는 장치가 실패했음을 나타냅니다. 이를 통해 사용자는 수행할 작업을 결정할 수 있습니다.

사용 편의성을 지원하기에 충분한 장치가 남아 있는 경우 RAID 논리 볼륨이 계속 작동합니다.

8.15.1. 할당 RAID 오류 정책

다음 예에서 raid_fault_policy 필드는 lvm.conf 파일에 할당 하도록 설정되어 있습니다. RAID 논리 볼륨은 다음과 같이 배치됩니다.

```
# 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/sde1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
  [my_lv_rimage_2]       /dev/sdg1(1)
  [my_lv_rmeta_0]        /dev/sde1(0)
  [my_lv_rmeta_1]        /dev/sdf1(0)
  [my_lv_rmeta_2]        /dev/sdg1(0)
```

/dev/sde 장치가 실패하면 시스템 로그에 오류 메시지가 표시됩니다.

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

```
Jan 17 15:57:18 bp-01 lvm[8599]: Device #0 of raid1 array, my_vg-my_lv, has failed.
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994294784: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
250994376704: Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at 0:
Input/output error
Jan 17 15:57:18 bp-01 lvm[8599]: /dev/sde1: read failed after 0 of 2048 at
4096: Input/output error
Jan 17 15:57:19 bp-01 lvm[8599]: Couldn't find device with uuid
3lugiV-3eSP-AFAR-sdrP-H20O-wM2M-qdMANy.
Jan 17 15:57:27 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is not in-sync.
Jan 17 15:57:36 bp-01 lvm[8599]: raid1 array, my_vg-my_lv, is now in-sync.
```

raid_fault_policy 필드가 할당 되도록 설정되어 있으므로 실패한 장치는 볼륨 그룹의 새 장치로 교체됩니다.

```
# lvs -a -o name,copy_percent,devices 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/sdf1(1)
  [lv_rimage_2]        /dev/sdg1(1)
  [lv_rmeta_0]         /dev/sdh1(0)
  [lv_rmeta_1]         /dev/sdf1(0)
  [lv_rmeta_2]         /dev/sdg1(0)
```

실패한 장치가 교체되었지만 디스플레이는 여전히 **LVM**에 실패한 장치를 찾을 수 없음을 나타냅니다. 이는 실패한 장치가 **RAID** 논리 볼륨에서 제거되었지만 볼륨 그룹에서 실패한 장치가 아직 제거되지 않았기 때문입니다. 볼륨 그룹에서 실패한 장치를 제거하려면 **first reduce --removemissing VG** 를 실행할 수 있습니다.

raid_fault_policy 가 **allocate** 로 설정되었지만 예비 장치가 없는 경우 할당이 실패하고 논리 볼륨이 그대로 유지됩니다. 할당이 실패하면 드라이브를 수정한 다음 **lvresh** 명령의 **--refresh** 옵션을 사용하여 실패한 장치를 복구할 수 있습니다. 또는 오류가 발생한 장치를 교체할 수 있습니다.

8.15.2. 경고 RAID 오류 정책

다음 예에서 **raid_fault_policy** 필드는 **lvm.conf** 파일에서 **warn** 으로 설정되어 있습니다. **RAID** 논리 볼륨은 다음과 같이 배치됩니다.

```
# 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/sdh1(1)
  [my_lv_rimage_1]       /dev/sdf1(1)
```

```
[my_lv_rimage_2]      /dev/sdg1(1)
[my_lv_rmeta_0]       /dev/sdh1(0)
[my_lv_rmeta_1]       /dev/sdf1(0)
[my_lv_rmeta_2]       /dev/sdg1(0)
```

/dev/sdh 장치가 실패하면 시스템 로그에 오류 메시지가 표시됩니다. 그러나 이 경우 LVM은 이미지 중 하나를 교체하여 RAID 장치를 자동으로 복구하지 않습니다. 대신 장치에 오류가 발생하면 lvconvert 명령의 --repair 인수로 장치를 교체할 수 있습니다.

8.16. 논리 볼륨에서 RAID 장치 교체

논리 볼륨에서 RAID 장치를 교체할 수 있습니다.

- *RAID 장치에 오류가 없는 경우 8.16.1절. "실패하지 않은 RAID 장치 교체" 를 따르십시오.*
- *RAID 장치가 실패한 경우 8.16.4절. "논리 볼륨에서 실패한 RAID 장치 교체" 를 따르십시오.*

8.16.1. 실패하지 않은 RAID 장치 교체

논리 볼륨에서 RAID 장치를 교체하려면 lvconvert 명령의 --replace 인수를 사용합니다.

사전 요구 사항

- *RAID 장치가 실패하지 않았습니다. RAID 장치에 실패한 경우 다음 명령이 작동하지 않습니다.*

절차

- *RAID 장치를 교체합니다.*

```
# lvconvert --replace dev_to_remove vg/lv possible_replacements
```

  - *dev_to_remove 를 교체할 물리 볼륨 경로로 바꿉니다.*
  - *Knative Serving/lv 를 볼륨 그룹 및 RAID 배열의 논리 볼륨 이름으로 바꿉니다.*

- *possible_replacements* 를 교체로 사용하려는 물리 볼륨의 경로로 바꿉니다.

예 8.1. RAID1 장치 교체

다음 예제에서는 **RAID1** 논리 볼륨을 생성한 다음 해당 볼륨에서 장치를 교체합니다.

1. **RAID1** 배열을 생성합니다.

```
# lvcreate --type raid1 -m 2 -L 1G -n my_lv my_vg

  Logical volume "my_lv" created
```

2. **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. **/dev/sdb2** 물리 볼륨을 교체합니다.

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

4. 교체를 사용하여 **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)
```

예 8.2. 대체 물리 볼륨 지정

다음 예제에서는 **RAID1** 논리 볼륨을 생성한 다음 해당 볼륨에서 장치를 교체하여 교체에 사용할 물리 볼륨을 지정합니다.

1. **RAID1** 배열을 생성합니다.

```
# lvcreate --type raid1 -m 1 -L 100 -n my_lv my_vg

  Logical volume "my_lv" created
```

2. **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_0]       /dev/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
```

3. 물리 볼륨을 검사합니다.

```
# pvs

  PV         VG      Fmt  Attr PSize    PFree
  /dev/sda1  my_vg   lvm2 a--  1020.00m  916.00m
  /dev/sdb1  my_vg   lvm2 a--  1020.00m  916.00m
  /dev/sdc1  my_vg   lvm2 a--  1020.00m 1020.00m
  /dev/sdd1  my_vg   lvm2 a--  1020.00m 1020.00m
```

4. **/dev/sdb1** 물리 볼륨을 **/dev/sdd1** 로 바꿉니다.

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

5. 교체를 사용하여 **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)
```

예 8.3. 여러 RAID 장치 교체

다음 예와 같이 여러 대체 인수를 지정하여 한 번에 두 개 이상의 **RAID** 장치를 교체할 수 있습니다.

1. **RAID1** 배열을 생성합니다.

```
# lvcreate --type raid1 -m 2 -L 100 -n my_lv my_vg

  Logical volume "my_lv" created
```

2. **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/sda1(1)
  [my_lv_rimage_1]       /dev/sdb1(1)
  [my_lv_rimage_2]       /dev/sdc1(1)
  [my_lv_rmeta_0]        /dev/sda1(0)
  [my_lv_rmeta_1]        /dev/sdb1(0)
  [my_lv_rmeta_2]        /dev/sdc1(0)
```

3. **/dev/sdb1** 및 **/dev/sdc1** 물리 볼륨을 교체합니다.

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

4. 교체를 사용하여 **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)
```

8.16.2. LVM RAID에서 실패한 장치

RAID는 기존 LVM 미러링과 같지 않습니다. LVM 미러링에 실패한 장치를 제거해야 하거나 미러링된 논리 볼륨이 중단되었습니다. RAID 배열은 실패한 장치로 계속 실행될 수 있습니다. 실제로 RAID1 이외의 RAID 유형의 경우 장치를 제거하면 더 낮은 수준 RAID(예: RAID6에서 RAID5로 변환하거나 RAID4 또는 RAID5에서 RAID0)로 변환하는 것을 의미합니다.

따라서 실패한 장치를 무조건 제거하고 교체를 할당하는 대신 LVM을 사용하면 lvconvert 명령의 --repair 인수를 사용하여 1단계 솔루션에서 RAID 볼륨에서 실패한 장치를 교체할 수 있습니다.

8.16.3. 논리 볼륨에서 실패한 RAID 장치 복구

LVM RAID 장치 오류가 일시적인 오류이거나 오류가 발생한 장치를 복구할 수 있는 경우 실패한 장치의 복구를 시작할 수 있습니다.

사전 요구 사항

- 이전에 실패한 장치가 이제 작동 중입니다.

절차

- RAID 장치가 포함된 논리 볼륨을 새로 고칩니다.

```
# lvchange --refresh my_vg/my_lv
```

검증 단계

- 복구된 장치로 논리 볼륨을 검사합니다.

```
# lvs --all --options name,devices,lv_attr,lv_health_status my_vg
```

8.16.4. 논리 볼륨에서 실패한 RAID 장치 교체

*이 절차에서는 **LVM RAID** 논리 볼륨에서 물리 볼륨 역할을 하는 실패한 장치를 대체합니다.*

사전 요구 사항

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

  *볼륨 그룹에서 사용 가능한 물리 볼륨이 충분한 물리 볼륨이 없는 경우, **KnativeServingextend** 유틸리티를 사용하여 충분히 큰 새 물리 볼륨을 추가합니다.*

절차

1. *다음 예에서 **RAID** 논리 볼륨은 다음과 같이 배치됩니다.*

```
# 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 실패하면 **lvs** 명령의 출력은 다음과 같습니다.*

```
# 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 교체하고 논리 볼륨을 표시합니다.

```
# lvconvert --repair my_vg/my_lv

  /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.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
  Faulty devices in my_vg/my_lv successfully replaced.
```

선택 사항: 오류가 발생한 장치를 대체하는 물리 볼륨을 수동으로 지정하려면 명령 끝에 물리 볼륨을 추가합니다.

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

4.

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-vIzA-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.*

5.

볼륨 그룹에서 실패한 장치를 제거합니다.

```
# vgreduce --removemissing VG
```

8.17. RAID 논리 볼륨(RAID 스크럽링)에서 데이터 일관성 확인

LVM은 RAID 논리 볼륨에 대한 스크럽 지원을 제공합니다. RAID 스크럽은 배열의 모든 데이터 및 패리티 블록을 읽고 일치 여부를 확인하는 프로세스입니다.

절차

1. *선택 사항: 스크럽 프로세스에서 사용하는 I/O 대역폭을 제한합니다.*

   RAID 스크럽 작업을 수행할 때 동기화 작업에 필요한 백그라운드 I/O는 볼륨 그룹 메타데이터 업데이트 등 다른 I/O를 LVM 장치에 분산시킬 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다. 복구 제한을 구현하여 스크럽 작업의 비율을 제어할 수 있습니다.

   다음 단계에서 lvchange --syncaction 명령에 다음 옵션을 추가합니다.

   --maxrecoveryrate Rate[bBsSkKmMgG]

   작업이 고등 I/O 작업을 수행할 수 있도록 최대 복구 비율을 설정합니다. 복구 비율을 0으로 설정하면 작업이 바인딩되지 않습니다.

   --minrecoveryrate Rate[bBsSkKmMgG]

   중복 I/O가 과도한 I/O가 있는 경우에도 동기화 작업의 I/O가 최소 처리량을 유지하도록 최소 복구 비율을 설정합니다.

   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 vg/raid_lv
```

3. *배열의 불일치를 수정하십시오.*

```
# lvchange --syncaction repair vg/raid_lv
```

참고

lvchange --syncaction repair 작업은 **lvconvert --repair** 작업과 동일한 기능을 수행하지 않습니다.

- **lvchange --syncaction repair** 작업은 배열에서 백그라운드 동기화 작업을 시작합니다.
- **lvconvert --repair** 작업 복구 또는 미러 또는 **RAID** 논리 볼륨에서 실패한 장치를 교체하십시오.

4. 선택 사항: 스크럽 작업에 대한 정보를 표시합니다.

```
# lvs -o +raid_sync_action,raid_mismatch_count vg/lv
```

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

  idle

  모든 동기화 작업이 완료됨(없음 실행)

  resync

  어레이 초기화 또는 머신 장애 후 복구

  recover

  배열에서 장치 교체

  Check

  array inconsistencies를 찾습니다.

  복구

  불일치 검색 및 수정

- **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 9는 논리 볼륨의 상태를 표시하고 다음 지표를 지원합니다.*

  - *m (mismatches)은 RAID 논리 볼륨에 불일치가 있음을 나타냅니다. 이 문자는 평가 작업이 RAID의 일부가 일치하지 않는 것을 감지한 후에 표시됩니다.*

  - *R (refresh)은 RAID 배열의 장치에 오류가 발생했으며 LVM이 장치 레이블을 읽고 작동하는 장치를 고려하더라도 커널이 실패한 것으로 간주했음을 나타냅니다. 논리 볼륨을 새로 고쳐 장치를 사용할 수 있음을 커널에 알리거나 실패한 것으로 의심되는 경우 장치를 교체합니다.*

추가 리소스

- *자세한 내용은 lvchange(8) 및 lvmraid(7) 도움말 페이지를 참조하십시오.*

8.18. RAID 수준 변환(RAID 사용)

LVM은 Raid takeover 를 지원하므로 RAID 논리 볼륨을 하나의 RAID 수준에서 다른 RAID 레벨로 변환합니다(예: RAID 5에서 RAID 6). RAID 수준을 변경하는 것은 일반적으로 장치 장애에 대한 탄력성을 늘리거나 줄이기 위해 수행하거나 논리 볼륨을 정지하는 것입니다. RAID 사용에는 lvconvert 를 사용합니다. RAID에 대한 자세한 내용은 lvconvert 를 사용하여 RAID 논리 볼륨을 변환하는 예제를 보려면 lvmraid(7) 도움말 페이지를 참조하십시오.

8.19. RAID 볼륨의 속성 변경 (RAID RESHAPE)

RAID 재sha ping은 동일한 RAID 수준을 유지하면서 RAID 논리 볼륨의 속성을 변경하는 것을 의미합니다. 변경할 수 있는 일부 속성에는 RAID 레이아웃, 스트라이프 크기 및 스트라이프 수가 포함됩니다. RAID 논리 볼륨을 재작업하기 위해 lvconvert 명령을 사용하는 RAID 재shaping 및 예제에 대한 자세한 내용은 lvmraid(7) 매뉴얼 페이지를 참조하십시오.

8.20. RAID1 논리 볼륨에서 I/O 작업 제어

lvchange 명령의 --write behind 매개변수를 사용하여 RAID1 논리 볼륨에서 장치의 I/O 작업을 제어할 수 있습니다. 이러한 매개 변수를 사용하는 형식은 다음과 같습니다.

- *--[raid]writemostly PhysicalVolume[:{t|y|n}]*

  RAID1 논리 볼륨의 장치를 대부분 쓰기로 표시합니다. 이러한 드라이브에 대한 모든 읽기는 필요하지 않은 한 피해야합니다. 이 매개변수를 설정하면 I/O 작업 수를 최소로 유지합니다. 기본적으로 논리 볼륨에 지정된 물리 볼륨에 대해 write-mostly 속성은 yes로 설정됩니다. 물리적 볼륨에 :n 을 추가하거나 :t 를 지정하여 값을 토글하여 쓰기-대부분 플래그를 제거할 수 있습니다. --write most 인수는 단일 명령으로 두 번 이상 지정할 수 있으므로 논리 볼륨에 있는 모든 물리 볼륨에 대한 쓰기-대부분 속성을 한 번에 변경할 수 있습니다.

- *--[RAID]writebehind IOCount*

  가장 많이 쓰기로 표시된 RAID1 논리 볼륨의 장치에 허용되는 적용되지 않은 쓰기의 최대 수를 지정합니다. 이 값을 초과하면 쓰기가 동기적으로 되어 배열이 쓰기가 완료되었다는 신호를 보내기 전에 모든 보조 장치에 대한 쓰기가 완료됩니다. 값을 0으로 설정하면 기본 설정이 지워지고 시스템에서 임의로 값을 선택할 수 있습니다.

8.21. RAID 논리 볼륨에서 영역 크기 변경

RAID 논리 볼륨을 생성할 때 논리 볼륨의 리전 크기는 /etc/lvm/lvm.conf 파일의 raid_region_size 매개변수 값이 됩니다. lvcreate 명령의 -R 옵션을 사용하여 이 기본값을 재정의할 수 있습니다.

RAID 논리 볼륨을 생성한 후 lvconvert 명령의 -R 옵션을 사용하여 볼륨의 영역 크기를 변경할 수 있습니다. 다음 예제에서는 논리 볼륨 KnativeServing /raidlv 의 지역 크기를 4096K로 변경합니다. 지역 크기를 변경하려면 RAID 볼륨을 동기화해야 합니다.

```
# lvconvert -R 4096K vg/raid1
Do you really want to change the region_size 512.00 KiB of LV vg/raid1 to 4.00 MiB? [y/n]: y
  Changed region size on RAID LV vg/raid1 to 4.00 MiB.
```

9장. 논리 볼륨의 스냅샷

LVM 스냅샷 기능을 사용하면 서비스 중단 없이 특정 시점에 볼륨의 가상 이미지(예: **/dev/sda**)를 생성할 수 있습니다.

9.1. 스냅샷 볼륨 개요

스냅샷을 가져온 후 원본 볼륨(원본)을 수정하면 스냅샷 기능은 볼륨 상태를 재구성할 수 있도록 변경 전과 마찬가지로 수정된 데이터 영역의 사본을 만듭니다. 스냅샷을 만들면 원본에 대한 전체 읽기 및 쓰기 액세스 권한은 그대로 유지됩니다.

스냅샷은 스냅샷이 생성된 후 변경되는 데이터 영역만 복사하므로 스냅샷 기능에는 최소한의 스토리지가 필요합니다. 예를 들어 원본의 용량이 거의 업데이트된 원본의 경우 스냅샷을 유지하기에 충분한 용량의 **3~5~5~4%**이면 됩니다. 백업 프로시저를 대체하는 것은 제공하지 않습니다. 스냅샷 복사본은 가상 복사본이며 실제 미디어 백업이 아닙니다.

스냅샷 크기는 원본 볼륨에 대한 변경 사항을 저장하기 위해 별도로 설정된 공간을 제어합니다. 예를 들어 스냅샷을 만든 다음 원본을 완전히 덮어쓰면 스냅샷은 변경 사항을 보유할 원본 볼륨만큼 최소한 커야 합니다. 스냅샷 크기를 정기적으로 모니터링해야 합니다. 예를 들어 **/usr** 와 같은 읽기**-**대부분 볼륨의 수명이 짧은 스냅샷에는 **/home** 과 같은 많은 쓰기가 포함되어 있기 때문에 볼륨의 장기 스냅샷보다 적은 공간이 필요합니다.

스냅샷이 가득 차면 원본 볼륨에서 변경 사항을 더 이상 추적할 수 없기 때문에 스냅샷이 무효화됩니다. 그러나 스냅샷이 유효하지 않게 하려면 사용량이 **snapshot_autoextend_threshold** 값을 초과할 때마다 스냅샷을 자동으로 확장하도록 **LVM**을 구성할 수 있습니다. 스냅샷은 완전히 복구할 수 있으며 다음 작업을 수행할 수 있습니다.

- 스토리지 용량이 있는 경우 스냅샷 볼륨의 크기를 늘려 해당 볼륨이 삭제되지 않도록 할 수 있습니다.
- 스냅샷 볼륨이 필요한 것보다 큰 경우 볼륨의 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 시스템을 복구해야 하는지 확인할 수 있습니다.

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

- **Red Hat Virtualization**에서 사용할 **LVM** 볼륨을 만들 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.

9.2. 원래 볼륨의 스냅샷 생성

lvcreate 명령을 **-s** 또는 **--size** 인수와 함께 사용한 다음 필요한 크기를 사용하여 원래 볼륨(원본)의 스냅샷을 만듭니다. 볼륨의 스냅샷은 쓰기 가능합니다. 기본적으로 스냅샷은 씬 프로비저닝된 스냅샷과 비교하여 일반 활성화 명령 중에 **origin**으로 활성화됩니다. **LVM**에서는 원본 볼륨의 크기 및 볼륨에 필요한 메타데이터 크기의 합계보다 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** 라는 원래 볼륨의 스냅샷 볼륨을 생성합니다.

사전 요구 사항

- **VDDK 001**이 볼륨 그룹을 생성 했습니다. 자세한 내용은 *LVM 볼륨 그룹 생성을* 참조하십시오.

절차

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

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

2.

*크기가 **100MB** 인 **/dev/vg001/origin snap** 이라는 스냅샷 논리 볼륨을 생성합니다.*

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

원래 논리 볼륨에 파일 시스템이 포함된 경우 원래 파일 시스템이 계속 업데이트되는 동안 파일 시스템의 콘텐츠에 액세스하기 위해 임의 디렉터리에 스냅샷 논리 볼륨을 마운트할 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 해당 상태와 같은 논리 볼륨 /dev/vg001/origin상태를 표시할 수도 있습니다.

주의

*스냅샷은 원본 볼륨이 변경될 때 크기가 증가하므로 **lvs** 명령으로 정기적으로 스냅샷 볼륨의 백분율을 모니터링하여 전체 상태가 되지 않도록 하는 것이 중요합니다. **100%** 가득 차 있는 스냅샷은 완전히 손실되므로 원본 부분에 대한 쓰기는 스냅샷을 손상시키지 않고 성공할 수 없습니다.*

4.

*사용량이 **snapshot_autoextend_threshold** 값을 초과하면 스냅샷이 **100 %** 가득 차면 유효하지 않도록 스냅샷을 자동으로 확장하도록 **LVM**을 구성할 수 있습니다. **/etc/lvm.conf** 파일에서 **snapshot_autoextend_threshold** 및 **snapshot_autoextend_percent** 옵션의 기존 값을 보고 요구 사항에 따라 편집합니다.*

*다음 예제에서는 **snapshot_autoextend_threshold** 옵션을 **100**보다 작은 값으로 설정하고 **snapshot_autoextend_percent** 옵션을 스냅샷 볼륨을 확장하려면 요구 사항에 따라 값으로 설정합니다.*

```
# vi /etc/lvm.conf
snapshot_autoextend_threshold = 70
snapshot_autoextend_percent = 20
```

다음 명령을 실행하여 이 스냅샷을 수동으로 확장할 수도 있습니다.

```
# lvextend -L+100M /dev/vg001/snap
```

참고

이 기능을 사용하려면 볼륨 그룹에 할당되지 않은 공간이 필요합니다. 스냅샷의 자동 확장에서는 스냅샷에 필요한 최대 계산된 크기 이상으로 스냅샷의 크기를 늘리지 않습니다. 스냅샷이 원본을 처리할 수 있을 만큼 크게 확장되면 더 이상 자동 확장을 모니터링하지 않습니다.

추가 리소스

- **lvcreate(8), lvextend(8)** 및 **lvs(8)** 매뉴얼 페이지
- **/etc/lvm/lvm.conf file**

9.3. 원래 볼륨에 스냅샷 병합

lvconvert 명령을 **--merge** 옵션과 함께 사용하여 스냅샷을 원래(원본) 볼륨으로 병합합니다. 데이터 또는 파일이 손실된 경우 시스템 롤백을 수행할 수 있습니다. 그렇지 않으면 시스템을 이전 상태로 복원해야 합니다. 스냅샷 볼륨을 병합한 후 결과 논리 볼륨에는 원본 볼륨의 이름, 마이너 번호, **UUID**가 있습니다. 병합이 진행 중인 동안 병합되는 스냅샷에 지시된 것처럼 원본을 읽거나 씁니다. 병합이 완료되면 병합된 스냅샷이 제거됩니다.

origin 및 스냅샷 볼륨이 모두 열려 있지 않고 활성 상태가 되면 병합이 즉시 시작됩니다. 그렇지 않으면 병합은 원본 또는 스냅샷이 활성화되고 둘 다 종료된 후에 시작됩니다. 원본 볼륨이 활성화된 후 루트 파일 시스템과 같이 닫을 수 없는 원본으로 스냅샷을 병합할 수 있습니다.

절차

1. 스냅샷 볼륨을 병합합니다. 다음 명령은 스냅샷 볼륨 **Replication 001/snap** 을 원본으로 병합합니다.

```
# lvconvert --merge vg001/snap
Merging of volume vg001/snap started.
  vg001/origin: Merged: 100.00%
```

2. 원본 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) 매뉴얼 페이지

10장. 씬 프로비저닝된 볼륨(볼륨 내) 생성 및 관리

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 사용하면 사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다.
- 씬 프로비저닝된 스냅샷 볼륨을 사용하면 동일한 데이터 볼륨에 더 많은 가상 장치를 저장할 수 있습니다.

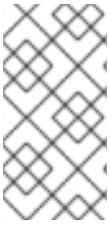
10.1. 씬 프로비저닝 개요

현재 많은 최신 스토리지 스택은 빠른 프로비저닝과 씬 프로비저닝 중에서 선택할 수 있는 기능을 제공합니다.

- 간격 프로비저닝은 실제 사용과 관계없이 블록이 할당된 블록 스토리지의 기존 동작을 제공합니다.
- 씬 프로비저닝은 데이터를 저장하는 물리적 장치보다 크기가 클 수 있는 더 큰 블록 스토리지 풀을 프로비저닝할 수 있는 기능을 부여하여 프로비저닝을 초과합니다. 개별 블록이 실제로 사용될 때까지 할당되지 않기 때문에 과도한 프로비저닝이 가능합니다. 동일한 풀을 공유하는 씬 프로비저닝된 장치가 여러 개 있는 경우 이러한 장치를 과도하게 프로비저닝할 수 있습니다.

씬 프로비저닝을 사용하면 물리적 스토리지를 과도하게 커밋하고 대신 씬 풀이라는 여유 공간 풀을 관리할 수 있습니다. 애플리케이션에 필요한 경우 이 씬 풀을 임의의 수의 장치에 할당할 수 있습니다. 스토리지 공간 할당을 위해 필요한 경우 씬 풀을 동적으로 확장할 수 있습니다.

예를 들어 10명의 사용자가 애플리케이션에 대해 100GB 파일 시스템을 요청하는 경우 각 사용자에 대해 100GB 파일 시스템으로 표시되는 내용을 생성할 수 있지만 필요한 경우에만 사용되는 실제 스토리지가 아닌 실제 스토리지에서 지원하는 항목을 생성할 수 있습니다.



참고

씬 프로비저닝을 사용하는 경우 스토리지 풀을 모니터링하고 사용 가능한 물리 공간이 부족해질 때 용량을 더 추가하는 것이 중요합니다.

다음은 씬 프로비저닝 장치를 사용할 때의 몇 가지 이점입니다.

- *사용 가능한 물리 스토리지보다 큰 논리 볼륨을 생성할 수 있습니다.*
- *더 많은 가상 장치를 동일한 데이터 볼륨에 저장할 수 있습니다.*
- *데이터 요구 사항을 지원하기 위해 논리적으로 자동으로 증가할 수 있는 파일 시스템을 생성하고 사용하지 않은 블록은 풀의 모든 파일 시스템에서 사용하기 위해 풀로 반환됩니다.*

다음은 씬 프로비저닝된 장치 사용의 잠재적인 단점입니다.

- *씬 프로비저닝된 볼륨은 사용 가능한 물리 스토리지가 부족할 위험이 있습니다. 기본 스토리지가 과도하게 프로비저닝되면 사용 가능한 물리적 스토리지가 부족하여 중단될 수 있습니다. 예를 들어 지원을 위해 **1T** 물리적 스토리지만 사용하여 씬 프로비저닝된 스토리지 **10T**를 생성하면 **1T**가 소모된 후 볼륨을 사용할 수 없게 됩니다.*
- *씬 프로비저닝된 장치 후 볼륨이 삭제됨을 계층으로 보내지 않으면 사용량 계산이 정확하지 않습니다. 예를 들어, **-o discard** 마운트 옵션 없이 파일 시스템을 배치하고 씬 프로비저닝된 장치 상단에 **fstrim** 을 주기적으로 실행하지 않으면 이전에 사용된 스토리지를 할당 해제하지 않습니다. 이러한 경우 실제로 사용하지 않더라도 시간이 지남에 따라 프로비저닝된 전체 용량을 사용하게 됩니다.*
- *사용 가능한 물리적 공간이 부족하지 않도록 논리 및 물리적 사용량을 모니터링해야 합니다.*
- *스냅샷이 있는 파일 시스템에서 **COW**(기록 시 복사) 작업이 느려질 수 있습니다.*
- *데이터 블록은 여러 파일 시스템 간에 격리될 수 있으며, 최종 사용자에게 표시되지 않는 경우에도 기본 스토리지의 임의의 액세스 제한이 발생합니다.*

10.2. 씬 프로비저닝된 논리 볼륨 생성

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

lvcreate 명령의 **-T** 또는 **--thin** 옵션을 사용하면 씬 풀 또는 씬 볼륨을 만들 수 있습니다. **lvcreate** 명령의 **-T** 옵션을 사용하여 단일 명령으로 씬 풀과 씬 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 만들기*를 참조하십시오.

절차

1. **thin** 풀을 생성합니다.

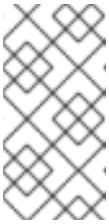
```
# lvcreate -L 100M -T vg001/mythinpool
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  Logical volume "mythinpool" created.
```

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 만들 수도 있습니다.

- **lvcreate** 명령의 **--thinpool** 매개변수를 사용하여 씬 풀을 생성할 수도 있습니다. **-T** 옵션과 달리 **--thinpool** 매개변수를 사용하려면 생성 중인 씬 풀 논리 볼륨의 이름을 지정해야 합니다. 다음 예제에서는 **--thinpool** 매개 변수를 사용하여 크기가 **100M** 인 **volumes** 그룹의 **mythinpool** 이라는 씬 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 대한 스트라이핑이 지원되면, 다음 명령은 **-i** 및 **-I** 옵션을 사용하여 스트라이프를 생성할 수 있습니다. 다음 명령은 두 개의 **64 kB** 스트라이프와 청크 크기가 **256 kB** 로 설정된 **volumes** 그룹에 **thinpool** 로 이름이 지정된 **100M** 씬 풀을 생성합니다. 또한 **rfc 001/thinvolume**이라는 **1T** 씬 볼륨 도 생성합니다.



참고

볼륨 그룹에 충분한 여유 공간이 있는 물리 볼륨이 두 개 있는지 확인하거나 씬 풀을 만들 수 없습니다.

```
# 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 포함된 풀보다 큰 볼륨의 가상 크기를 지정합니다. 다음 예와 같이 추가 매개변수를 사용하여 씬 볼륨을 생성할 수도 있습니다.

- *씬 볼륨과 씬 풀을 모두 생성하려면 **lvcreate** 명령의 **-T** 옵션을 사용하고 크기 및 가상 크기 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 씬 볼륨과 씬 풀을 만들려면 **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. 기존 논리 볼륨을 씬 풀 볼륨의 메타데이터 볼륨으로 변환하려면 **--thinpool** 매개변수와 함께 **--poolmetadata** 매개변수를 사용해야 합니다.*

*다음 예제에서는 볼륨 그룹의 기존 논리 볼륨 **lv1** 을 **thin pool** 볼륨으로 변환하고 기존 논리 볼륨 **lv2** 를 볼륨 그룹 **overcome 001** 의 메타데이터 볼륨으로 해당 **thin pool** 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 유지하지 않고 콘텐츠를 덮어쓰므로 논리 볼륨의 콘텐츠가 제거됩니다.*

- *기본적으로 **lvcreate** 명령은 다음 공식에 따라 **thin** 풀의 메타데이터 논리 볼륨의 크기를 설정합니다.*

```
Pool_LV_size / Pool_LV_chunk_size * 64
```

*스냅샷 수가 많거나 씬 풀에 대한 작은 청크 크기가 있는 경우 나중에 **lvcreate** 명령의 **--poolmetadatasize** 매개변수를 사용하여 **thin pool**의 메타데이터 볼륨의 기본값을 늘려야 할 수 있습니다. 씬 풀의 메타데이터 논리 볼륨에 지원되는 값은 **2MiB**에서 **16GiB** 사이의 범위입니다.*

*다음 예제에서는 **thin** 풀의 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 씬 풀 및 씬 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 풀 크기를 확장합니다. 그러나 씬 풀의 크기를 줄일 수는 없습니다.*

참고

*씬 풀과 씬 볼륨을 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** 풀과 **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 풀과 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.*

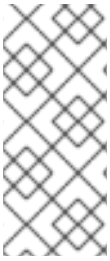
```
# lvremove -f vg001/mythinpool1
  Logical volume "thinvolume1" successfully removed.
  Logical volume "mythinpool1" successfully removed.
```

추가 리소스

- *lvcreate(8), lvrename(8), lvs(8) 및 lvconvert(8) 도움말 페이지*

10.3. 씬 프로비저닝된 스냅샷 볼륨

Red Hat Enterprise Linux는 씬 프로비저닝된 스냅샷 볼륨을 지원합니다. 씬 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

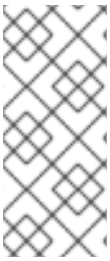
모든 LVM 스냅샷 볼륨 및 모든 씬 볼륨과 마찬가지로 씬 스냅샷 볼륨은 클러스터의 노드에서 지원되지 않습니다. 스냅샷 볼륨은 하나의 클러스터 노드에서만 활성화해야 합니다.

기존 스냅샷은 원본이 변경될 때 데이터가 보존되는 각 스냅샷에 대해 새 공간을 할당해야 합니다. 그러나 씬 프로비저닝 스냅샷은 동일한 공간을 원본과 공유합니다. thin LV의 스냅샷은 thin LV에 공통된 데이터 블록과 해당 스냅샷 중 하나가 공유되기 때문에 효율적입니다. thin LV의 스냅샷을 생성하거나 다른 씬 스냅샷에서 생성할 수 있습니다. 재귀 스냅샷에 공통된 블록도 thin 풀에서 공유됩니다.

씬 스냅샷 볼륨은 다음과 같은 이점을 제공합니다.

- 원본의 스냅샷 수를 늘리면 성능에 부정적인 영향을 미칩니다.
- 씬 스냅샷 볼륨은 새 데이터가 작성되어 각 스냅샷에 복사되지 않기 때문에 디스크 사용량을 줄일 수 있습니다.
- 기존 스냅샷의 요구 사항인 원본을 사용하여 씬 스냅샷 볼륨을 동시에 활성화할 필요가 없습니다.
- 스냅샷에서 원본을 복원할 때는 씬 스냅샷을 병합할 필요가 없습니다. 원본을 제거하고 대신 스냅샷을 사용할 수 있습니다. 기존 스냅샷에는 변경 사항을 저장하는 별도의 볼륨이 있으며, 이는 원본과 병합되어 재설정됩니다.
- 기존 스냅샷과 비교하여 허용되는 스냅샷 수에 대한 제한이 크게 높습니다.

씬 스냅샷 볼륨을 사용할 때는 많은 이점이 있지만 기존 **LVM** 스냅샷 볼륨 기능이 요구 사항에 더 적합할 수 있는 몇 가지 사용 사례가 있습니다. 모든 유형의 볼륨에서 기존 스냅샷을 사용할 수 있습니다. 그러나 **thin-snapshots**를 사용하려면 **thin-provisioning**을 사용해야 합니다.



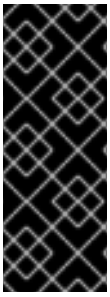
참고

씬 스냅샷 볼륨의 크기는 제한할 수 없습니다. 스냅샷은 필요한 경우 씬 풀의 모든 공간을 사용합니다. 일반적으로 사용할 스냅샷 형식을 결정할 때 사이트의 특정 요구 사항을 고려해야 합니다.

기본적으로 일반 활성화 명령 중에 **thin** 스냅샷 볼륨을 건너뜁니다.

10.4. 씬 프로비저닝된 스냅샷 볼륨 생성

씬 프로비저닝된 스냅샷 볼륨을 사용하면 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 볼륨이 생성되어 있습니다. 자세한 내용은 *씬 프로비저닝된 논리 볼륨 생성을* 참조하십시오.

절차

- 씬 프로비저닝된 스냅샷 볼륨을 생성합니다. 다음 명령은 씬 프로비저닝된 논리 볼륨 **8200 1/thinvolume**의 **mysnapshot1** 로 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. 씬 볼륨의 크기를 확장하는 방법에 대한 자세한 내용은 *씬 프로비저닝된 논리 볼륨 생성을* 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 생

성합니다. thin 스냅샷 볼륨의 이름은 mythinsnap 입니다. 그런 다음 논리 볼륨 origin_volume 은 기존 씬 풀 rfc001을 사용하는 볼륨 그룹sssd 001 의 thin snapshot volume 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 프로비저닝 스냅샷 볼륨을 생성하려면 다음 명령을 사용합니다.

```
# lvcreate -s vg001/mysnapshot2 --name mysnapshot3
Logical volume "mysnapshot3" created.
```

검증

- *씬 스냅샷 논리 볼륨의 모든 mostly 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
```

여기,

- *thinvolume 은 volume group Replication 001 의 origin 볼륨입니다.*
- *mysnapshot1 은 thinvolume의 스냅샷입니다.*
- *mysnapshot2 는 mysnapshot1의 스냅샷입니다.*

- ***mysnapshot3** 은 **mysnapshot2**의 스냅샷입니다.*

참고

***lv_ancestors** 및 **lv_descendants** 필드에 기존 종속성이 표시됩니다. 그러나 해당 항목이 체인 중간에서 제거된 경우 종속성 체인을 중단할 수 있는 제거된 항목을 추적하지 않습니다.*

추가 리소스

- ***lvcreate(8)** 매뉴얼 페이지*

11장. 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.

11.1. 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

- 캐시 모드 및 기타 자세한 내용은 lvmcache(7) 도움말 페이지를 참조하십시오.

11.2. LVM 캐싱 구성 요소

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 **LVM**은 모든 캐싱 구성 요소를 함께 그룹화하는 단일 결과 장치를 노출합니다. 결과 장치의 이름은 느린 원래 논리 볼륨과 동일합니다.

11.3. 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 장치 경로(예: /dev/sdf1)

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.

검증 단계

- *새로 생성된 장치를 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

- 이 절차 및 관리 예제를 포함한 기타 세부 정보는 **lvmcache(7)** 도움말 페이지를 참조하십시오.

11.4. 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

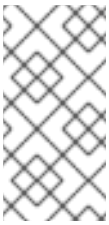
cachepool 볼륨의 이름

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.

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.

검증 단계

- 새로 생성된 장치를 확인합니다.

```
# 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)** 도움말 페이지.

- ***lvmcache(7)** 도움말 페이지.*

- ***lvconvert(8)** 도움말 페이지.*

11.5. 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.*

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 장치 경로(예: /dev/sdf1)

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.

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

- 관리 예제를 포함한 자세한 내용은 **lvmcache(7)** 도움말 페이지를 참조하십시오.

11.6. 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
```

다음 값을 바꿉니다.

vg

볼륨 그룹 이름

main-lv

캐싱이 활성화된 논리 볼륨의 이름입니다.

2. **cachevol** 또는 **cachepool** 볼륨을 분리합니다.

```
# lvconvert --splitcache vg/main-lv
```

다음 값을 바꿉니다.

vg

볼륨 그룹 이름

main-lv

캐싱이 활성화된 논리 볼륨의 이름입니다.

검증 단계

- 논리 볼륨이 더 이상 함께 연결되지 않았는지 확인합니다.

```
# lvs --all --options +devices [replaceable]_vg_

LV      Attr       Type   Devices
fastvol -wi------- linear /dev/fast-pv
main-lv -wi------- linear /dev/slow-pv
```

추가 리소스

- **lvmcache(7)** 도움말 페이지

12장. 논리 볼륨 활성화

활성 상태인 논리 볼륨은 블록 장치를 통해 사용할 수 있습니다. 활성화되는 논리 볼륨은 액세스할 수 있으며 변경될 수 있습니다. 논리 볼륨을 생성하면 기본적으로 활성화됩니다.

개별 논리 볼륨을 비활성화하여 커널에서 알 수 없는 다양한 상황이 있습니다. **lvchange** 명령의 **-a** 옵션을 사용하여 개별 논리 볼륨을 활성화하거나 비활성화할 수 있습니다.

개별 논리 볼륨을 비활성화하는 명령의 형식은 다음과 같습니다.

```
lvchange -an vg/lv
```

개별 논리 볼륨을 활성화하는 명령의 형식은 다음과 같습니다.

```
lvchange -ay vg/lv
```

KnativeServing change 명령의 **-a** 옵션을 사용하여 볼륨 그룹의 모든 논리 볼륨을 활성화 및 활성화 또는 비활성화할 수 있습니다. 이는 볼륨 그룹의 개별 논리 볼륨에서 **lvchange -a** 명령을 실행하는 것과 동일합니다.

볼륨 그룹의 모든 논리 볼륨을 비활성화하는 명령 형식은 다음과 같습니다.

```
vgchange -an vg
```

볼륨 그룹의 모든 논리 볼륨을 활성화하는 명령 형식은 다음과 같습니다.

```
vgchange -ay vg
```

12.1. 논리 볼륨의 자동 비활성화 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.

/etc/lvm/lvm.conf 구성 파일에서 다음 구성 옵션을 사용하여 논리 볼륨의 자동 비활성화를 제어할 수 있습니다.

- 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.

이러한 옵션 설정에 대한 자세한 내용은 /etc/lvm/lvm.conf 구성 파일을 참조하십시오.

12.2. 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 옵션을 사용하는 방법에 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.

다음과 같은 방법으로 논리 볼륨에서 활성화 건너뛰기를 설정할 수 있습니다.

- lvcreate 명령의 -kn 또는 --setactivationskip n 옵션을 지정하여 논리 볼륨을 생성할 때 활성화 건너뛰기를 비활성화할 수 있습니다.

- lvchange 명령의 -kn 또는 --setactivationskip n 옵션을 지정하여 기존 논리 볼륨의 활성화 건너뛰기를 비활성화할 수 있습니다.

- *lvchange 명령의 -ky 또는 --setactivationskip y 옵션을 사용하여 전원이 꺼진 볼륨에 대해 활성화 건너뛰기를 다시 활성화할 수 있습니다.*

논리 볼륨에 대한 활성화 건너뛰기 플래그가 설정되었는지 확인하려면 다음 예제와 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 설정된 k 속성으로 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 활성화 건너뛰기 플래그가 설정된 씬 스냅샷 논리 볼륨을 활성화합니다.

```
# lvchange -ay -K VG/SnapLV
```

다음 명령은 활성화 건너뛰기 플래그를 사용하지 않고 씬 스냅샷을 생성합니다.

```
# lvcreate --type thin -n SnapLV -kn -s ThinLV --thinpool VG/ThinPoolLV
```

다음 명령은 스냅샷 논리 볼륨에서 활성화 건너뛰기를 제거합니다.

```
# lvchange -kn VG/SnapLV
```

12.3. 공유 논리 볼륨 활성화

다음과 같이 lvchange 및virtualization change 명령의 -a 옵션을 사용하여 공유 논리 볼륨의 논리 볼륨 활성화를 제어할 수 있습니다.

명령	활성화
lvchange -ay|e	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.

12.4. 누락된 장치가 있는 논리 볼륨 활성화

***lvchange** 명령을 사용하여 다음 값 중 하나로 **activation_mode** 매개변수를 설정하여 누락된 장치가 있는 논리 볼륨을 구성할 수 있습니다.*

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.

***activation_mode** 의 기본값은 **/etc/lvm/lvm.conf** 파일의 **activation_mode** 설정에 따라 결정됩니다. 자세한 내용은 **lvmraid(7)** 도움말 페이지를 참조하십시오.*

13장. LVM 장치 스캔 제어

/etc/lvm/lvm.conf 파일에서 필터를 구성하여 LVM 장치 검사를 제어할 수 있습니다. lvm.conf 파일의 필터는 /dev 디렉토리의 장치 이름에 적용할 수 있는 일련의 간단한 정규 표현식으로 구성되며, 발견된 각 블록 장치를 수락할지 여부를 결정합니다.

13.1. LVM 장치 필터

LVM 툴은 /dev 디렉토리에서 장치를 스캔하고 LVM 메타데이터에 대한 모든 장치를 확인합니다. /etc/lvm/lvm.conf 파일의 필터는 LVM을 스캔하는 장치를 제어합니다.

필터는 /dev 디렉토리 또는 /etc/lvm/lvm.conf 파일의 dir 키워드로 지정된 디렉터리에 LVM이 적용되는 패턴 목록입니다. 패턴은 임의의 문자로 구분된 정규식이며, 앞에 허용 또는 r 을 거부하기 위해 붙습니다. LVM이 장치를 수락하거나 거부(ignore)하는지 여부를 확인하는 목록의 첫 번째 정규식입니다. LVM은 패턴과 일치하지 않는 장치를 허용합니다.

다음은 모든 장치를 검사하는 필터의 기본 구성입니다.

```
filter = [ "a/.*/" ]
```

13.2. LVM 장치 필터 구성의 예

다음 예제에서는 필터를 사용하여 LVM 스캔을 제어하는 방법을 보여줍니다.

주의

여기에 제시된 일부 예제는 시스템의 추가 장치와 의도하지 않게 일치할 수 있으며 시스템에 권장되는 관행을 나타내지 않을 수 있습니다. 예를 들어 a/loop/ 는 a/.*loop.*/ 와 동일하며 /dev/solooperation/lvol1 과 일치합니다.

- 다음 필터는 구성 파일에서 필터가 구성되지 않기 때문에 기본 동작인 모든 검색된 장치를 추가합니다.

```
filter = [ "a/.*/" ]
```

- 다음 필터는 드라이브에 미디어 가 없는 경우 지연을 방지하기 위해 **cdrom** 장치를 제거합니다.

```
filter = [ "r|^/dev/cdrom$|" ]
```

- 다음 필터는 모든 루프 장치를 추가하고 다른 모든 블록 장치를 제거합니다.

```
filter = [ "a/loop/", "r/.*/" ]
```

- 다음 필터는 모든 루프 및 **IDE** 장치를 추가하고 다른 모든 블록 장치를 제거합니다.

```
filter = [ "a|loop|", "a|/dev/hd.*|", "r|.*|" ]
```

- 다음 필터는 첫 번째 **IDE** 드라이브에서 파티션 **8**을 추가하고 다른 모든 블록 장치를 제거합니다.

```
filter = [ "a|^/dev/hda8$|", "r/.*/" ]
```

13.3. LVM 장치 필터 구성 적용

이 절차에서는 **LVM** 스캔 장치를 제어하는 **LVM** 장치 필터의 구성을 변경합니다.

사전 요구 사항

- 사용할 장치 필터 패턴을 준비합니다.

절차

1. **/etc/lvm/lvm.conf** 파일을 수정하지 않고 장치 필터 패턴을 테스트합니다.

   --config 'devices{ filter = [장치 필터 패턴] }' 옵션과 함께 **LVM** 명령을 사용합니다. 예를 들면 다음과 같습니다.

```
# lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|.*|" ] }'
```

2. 새 장치 필터 패턴을 사용하도록 **/etc/lvm/lvm.conf** 구성 파일의 **filter** 옵션을 편집합니다.

3. *사용하려는 물리 볼륨 또는 볼륨 그룹이 없는 경우 새 구성과 함께 누락되어 있는지 확인합니다.*

```
# pvscan
```

```
# vgscan
```

4. *재부팅 시 **LVM**이 필요한 장치만 검사하도록 **initramfs** 파일 시스템을 다시 빌드합니다.*

```
# dracut --force --verbose
```

14장. 논리 볼륨 상단에 LVM 물리 볼륨 계층화

논리 볼륨 상단에 물리 볼륨을 생성할 수 있도록 LVM을 구성할 수 있습니다.

기본적으로 LVM 명령은 시스템의 논리 볼륨을 스캔하지 않습니다. 이 기본 동작은 다음과 같은 이점을 제공합니다.

- 시스템에 활성 논리 볼륨이 많은 경우 모든 LVM 명령에 추가 시간이 필요하므로 성능에 부정적인 영향을 미치고 원치 않는 지연 또는 시간 초과가 발생할 수 있습니다.
- 논리 볼륨에 게스트 VM 이미지의 물리 볼륨이 포함된 경우 호스트는 일반적으로 게스트에 속하는 계층적 물리 볼륨을 검사하거나 사용하지 않도록 합니다. 그러나 호스트의 LVM이 이러한 물리 볼륨에 액세스하는 것을 방지하기 위해 13장. LVM 장치 스캔 제어 에 게스트 VM의 물리 볼륨이 직접 존재하는 경우 에 설명된 대로 필터를 구성해야 합니다.

논리 볼륨 상단에 물리 볼륨을 계층화할 때 논리 볼륨을 스캔해야 할 수 있습니다. 그러면 pvcreate 명령을 논리 볼륨에서 실행할 수 있습니다. 모든 논리 볼륨을 검사하도록 LVM을 구성하려면 /etc/lvm/lvm.conf 파일의 scan_lvs 구성 옵션을 scan_lvs=1. 어떤 논리 볼륨 LVM 명령 검사를 제한하려면 13장. LVM 장치 스캔 제어 에 설명된 대로 /etc/lvm/lvm.conf 구성 파일에 장치 필터를 설정할 수 있습니다.

15장. LVM 할당 제어

기본적으로 볼륨 그룹은 동일한 물리 볼륨에 병렬 스트라이프를 배치하지 않는 것과 같은 공통의 규칙에 따라 물리 확장 영역을 할당합니다. 이는 일반 할당 정책입니다. **pxecreate** 명령의 **--alloc** 인수를 사용하여 연속 된 **,anywhere** 또는 **cling** 의 할당 정책을 지정할 수 있습니다. 일반적으로 일반 이외의 할당 정책은 비정상적인 범위 또는 비표준 할당을 지정해야 하는 특수한 경우에만 필요합니다.

15.1. LVM 할당 정책

LVM 작업에서 하나 이상의 논리 볼륨에 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 물리 볼륨의 해당 범위 내에서 할당되지 않은 물리 확장 영역만 고려됩니다.

- 각 할당 정책은 가장 엄격한 정책(연속 연속)부터 시작하여 **--alloc** 옵션을 사용하여 지정된 할당 정책으로 종료하거나 특정 논리 볼륨 또는 볼륨 그룹의 기본값으로 설정됩니다. 각 정책의 경우 할당 정책에 따라 가능한 한 많은 공간을 할당해야 하는 빈 논리 볼륨 공간의 가장 낮은 논리 범위부터 작동합니다. 공간이 더 필요한 경우 **LVM**은 다음 정책으로 이동합니다.

할당 정책 제한은 다음과 같습니다.

- 연속 된 할당 정책에서는 논리 볼륨의 첫 번째 논리 범위가 아닌 논리 범위의 물리적 위치가 즉시 논리 범위 앞에 인접하여 있어야 합니다.

  논리 볼륨을 스트라이핑하거나 미러링하면 연속 할당 제한이 공간이 필요한 각 스트라이프 또는 미러 이미지(**leg**)에 독립적으로 적용됩니다.

- **cling** 의 할당 정책을 사용하려면 해당 논리 볼륨의 앞부분에서 이미 사용 중인 논리 볼륨에 사용된 물리 볼륨을 기존 논리 볼륨에 추가해야 합니다. 구성 매개변수 할당/**cling_tag_list** 가 정의된 경우 나열된 태그가 두 물리 볼륨 모두에 있는 경우 두 개의 물리 볼륨이 일치하도록 간주됩니다. 이를 통해 유사한 속성(예: 물리 위치)을 포함한 물리적 볼륨 그룹에 태그를 지정하고 할당 목적으로 동일하게 취급할 수 있습니다.

  논리 볼륨을 제거 또는 미러링할 때는 공간이 필요한 각 스트라이프 또는 미러 이미지(**leg**)에 독립적으로 연결 할당 제한이 적용됩니다.

-

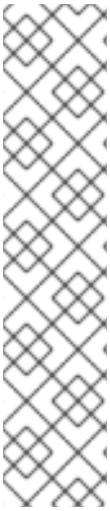
일반 할당 정책은 해당 병렬 논리 볼륨 내의 동일한 오프셋에서 병렬 논리 볼륨(즉, 다른 스트라이프 또는 미러 이미지/leg)에 이미 할당된 논리 범위와 동일한 물리 볼륨을 공유하는 물리 범위를 선택하지 않습니다.

미러 데이터를 유지하기 위해 논리 볼륨과 동시에 미러 로그를 할당할 때 일반 의 할당 정책은 먼저 로그 및 데이터에 대한 다른 물리 볼륨을 선택합니다. 이것이 가능하지 않고 **allocation/mirror_logs_require_separate_pvs** 구성 매개변수가 **0**으로 설정된 경우 로그가 데이터의 일부로 물리 볼륨을 공유할 수 있습니다.

마찬가지로, 씬 풀 메타데이터를 할당할 때, 일반 의 할당 정책은 **allocation/thin_pool_metadata_require_separate_pvs** 구성 매개변수 값에 따라 미러 로그 할당과 동일한 고려 사항을 따릅니다.

- 할당 요청을 충족할 수 있는 사용 가능한 범위가 충분하지만 일반 할당 정책에서 사용하지 않는 경우 동일한 물리 볼륨에 두 개의 스트라이프를 배치하여 성능이 저하되는 경우에도 모든 할당 정책이 사용됩니다.

할당 정책은 **KnativeServing change** 명령을 사용하여 변경할 수 있습니다.



참고

정의된 할당 정책에 따라 이 섹션에 설명된 레이아웃 동작 이상의 레이아웃 동작에 의존하는 경우 코드의 향후 버전에서 변경될 수 있습니다.**If you rely on any layout behavior beyond that documented in this section according to the defined allocation policies, you should note that this might change in future versions of the code.** 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 논리 볼륨에 대한 특정 레이아웃을 얻는 것이 중요한 경우 **lvcreate** 및 **lvconvert** 단계의 시퀀스를 통해 빌드해야 합니다. 이렇게 하면 각 단계에 적용되는 할당 정책이 레이아웃에 따라 결정되지 않도록 **LVM**을 남겨 둡니다.

할당 프로세스가 특정 사례에서 현재 작동하는 방식을 보려면 명령에 **-vvvv** 옵션을 추가하여 디버그 로깅 출력을 읽을 수 있습니다.

15.2. 물리 볼륨에서 할당 방지

pvchange 명령을 사용하여 하나 이상의 물리 볼륨의 사용 가능한 공간에 물리 확장 영역의 할당을 방지할 수 있습니다. 디스크 오류가 있거나 물리 볼륨을 제거하는 경우 이 작업이 필요할 수 있습니다.

다음 명령은 /dev/sdk1 의 물리 확장 영역 할당을 허용하지 않습니다.

```
# pvchange -x n /dev/sdk1
```

pvchange 명령의 -xy 인수를 사용하여 이전에 허용하지 않은 위치를 할당할 수도 있습니다.

15.3. CLING 할당 정책을 사용하여 논리 볼륨 확장

LVM 볼륨을 확장하는 경우 lvextend 명령의 --alloc cling 옵션을 사용하여 연결 할당 정책을 지정할 수 있습니다. 이 정책은 기존 논리 볼륨의 마지막 세그먼트와 동일한 물리 볼륨의 공간을 선택합니다. 물리 볼륨에 공간이 부족하고 태그 목록이 /etc/lvm/lvm.conf 파일에 정의되어 있는 경우 LVM에서 태그가 물리 볼륨에 연결되어 있는지 확인하고 기존 Extent와 새 Extent 간에 해당 물리 볼륨 태그를 찾습니다.

예를 들어 단일 볼륨 그룹 내의 두 사이트 간에 미러링된 논리 볼륨이 있는 경우 물리적 볼륨을 @site1 및 @site2 태그로 태깅하여 배치된 위치에 따라 물리 볼륨을 태그할 수 있습니다. 그러면 lvm.conf 파일에서 다음 행을 지정할 수 있습니다.

```
cling_tag_list = [ "@site1", "@site2" ]
```

다음 예에서 lvm.conf 파일은 다음 행을 포함하도록 수정되었습니다.

```
cling_tag_list = [ "@A", "@B" ]
```

또한 이 예에서 물리 볼륨 /dev/sdb1,/dev/sdc1,/dev/sdd1,/dev/sde1,/dev/sdf1, /dev/sdg1 , /dev/sdg1 , /dev/sdg1 , /dev/sdg1 , /dev/sdg1 ,/dev/sdg1, /dev/sdg1 로 구성된 볼륨 그룹 taft 가 생성되었습니다. 이러한 물리 볼륨에는 A,B 및 C 태그가 지정되어 있습니다. 예제에서는 C 태그를 사용하지 않지만 LVM에서 태그를 사용하여 미러에 사용할 물리 볼륨을 선택한다는 내용이 표시됩니다.

```
# pvs -a -o +pv_tags /dev/sd[bcdefgh]
  PV         VG   Fmt  Attr PSize  PFree  PV Tags
  /dev/sdb1  taft lvm2 a--  15.00g 15.00g A
  /dev/sdc1  taft lvm2 a--  15.00g 15.00g B
  /dev/sdd1  taft lvm2 a--  15.00g 15.00g B
  /dev/sde1  taft lvm2 a--  15.00g 15.00g C
  /dev/sdf1  taft lvm2 a--  15.00g 15.00g C
  /dev/sdg1  taft lvm2 a--  15.00g 15.00g A
  /dev/sdh1  taft lvm2 a--  15.00g 15.00g A
```

다음 명령은 볼륨 그룹 taft 에서 10기가바이트 미러링된 볼륨을 생성합니다.

```
# lvcreate --type raid1 -m 1 -n mirror --nosync -L 10G taft
  WARNING: New raid1 won't be synchronised. Don't read what you didn't write!
  Logical volume "mirror" created
```

다음 명령은 미러 위치 및 **RAID** 메타데이터 하위 볼륨에 사용되는 장치를 보여줍니다.

```
# lvs -a -o +devices
  LV              VG   Attr       LSize  Log Cpy%Sync Devices
  mirror          taft Rwi-a-r--- 10.00g       100.00 mirror_rimage_0(0),mirror_rimage_1(0)
  [mirror_rimage_0] taft iwi-aor--- 10.00g              /dev/sdb1(1)
  [mirror_rimage_1] taft iwi-aor--- 10.00g              /dev/sdc1(1)
  [mirror_rmeta_0]  taft ewi-aor---  4.00m              /dev/sdb1(0)
  [mirror_rmeta_1]  taft ewi-aor---  4.00m              /dev/sdc1(0)
```

다음 명령은 미러링된 볼륨의 크기를 확장하며, **cling** 할당 정책을 사용하여 동일한 태그가 있는 물리 볼륨을 사용하여 미러 브리지를 확장해야 함을 나타냅니다.

```
# lvextend --alloc cling -L +10G taft/mirror
  Extending 2 mirror images.
  Extending logical volume mirror to 20.00 GiB
  Logical volume mirror successfully resized
```

다음 디스플레이 명령은 대문자와 동일한 태그가 있는 물리 볼륨을 사용하여 미러 가 확장되었음을 보여줍니다. **C** 태그가 있는 물리 볼륨은 무시됩니다.

```
# lvs -a -o +devices
  LV              VG   Attr       LSize  Log Cpy%Sync Devices
  mirror          taft Rwi-a-r--- 20.00g       100.00 mirror_rimage_0(0),mirror_rimage_1(0)
  [mirror_rimage_0] taft iwi-aor--- 20.00g              /dev/sdb1(1)
  [mirror_rimage_0] taft iwi-aor--- 20.00g              /dev/sdg1(0)
  [mirror_rimage_1] taft iwi-aor--- 20.00g              /dev/sdc1(1)
  [mirror_rimage_1] taft iwi-aor--- 20.00g              /dev/sdd1(0)
  [mirror_rmeta_0]  taft ewi-aor---  4.00m              /dev/sdb1(0)
  [mirror_rmeta_1]  taft ewi-aor---  4.00m              /dev/sdc1(0)
```

15.4. 태그를 사용하여 LVM RAID 오브젝트 간 차별화

LVM RAID 개체에 태그를 할당하여 그룹별로 **LVM RAID** 동작 제어를 자동화할 수 있습니다.

PV(물리 볼륨**)** 태그는 논리 볼륨**(LV)** 또는 볼륨 그룹**(VG)** 태그와 달리, 할당 정책에 따라 **PV** 수준에서 수행되므로 **LVM raid**의 할당 제어를 담당합니다. 스토리지 유형을 다른 속성으로 구분하려면 적절하게 태그를 지정합니다**(**예**: NVMe, SSD, HDD). VG**에 추가한 후 각 새 **PV**에 태그를 지정하는 것이 좋습니다.

*이 절차에서는 **/dev/sda** 가 **SSD**라고 가정하고 **/dev/sd[b-f]** 가 하나의 파티션이 있는 **HDD**라고 가정하여 논리 볼륨에 오브젝트 태그를 추가합니다.*

사전 요구 사항

- ***lvm2** 패키지가 설치되어 있습니다.*
- ***PV**로 사용할 스토리지 장치를 사용할 수 있습니다.*

절차

1. *볼륨 그룹을 만듭니다.*

```
# vgcreate MyVG /dev/sd[a-f]1
```

2. *물리 볼륨에 태그를 추가합니다.*

```
# pvchange --addtag ssds /dev/sda1

# pvchange --addtag hdds /dev/sd[b-f]1
```

3. ***RAID6** 논리 볼륨을 생성합니다.*

```
# lvcreate --type raid6 --stripes 3 -L1G -nr6 MyVG @hdds
```

4. *선형 캐시 풀 볼륨을 만듭니다.*

```
# lvcreate -nr6pool -L512m MyVG @ssds
```

5. ***RAID6** 볼륨을 캐시하도록 변환합니다.*

```
# lvconvert --type cache --cachevol MyVG/r6pool MyVG/r6
```

추가 리소스

- ***lvcreate(8), lvconvert(8), lvmraid(7)** 및 **lvmcache(7)** 도움말 페이지.*

16장. 태그가 있는 LVM 오브젝트 그룹화

시스템 관리자는 활성화와 같은 LVM 동작 제어를 그룹별로 자동화할 수 있도록 LVM 개체에 태그를 그룹화할 수 있습니다.

16.1. LVM 오브젝트 태그

LVM 태그는 동일한 유형의 LVM2 개체를 그룹화하는 데 사용되는 용어입니다. 태그는 물리 볼륨, 볼륨 그룹, 논리 볼륨과 같은 오브젝트와 클러스터 구성의 호스트에 연결됩니다.

태그는 PV, VG 또는 LV 인수 대신 명령줄에서 제공됩니다. 모호성을 방지하려면 태그 앞에 @를 추가해야 합니다. 각 태그는 명령줄의 위치에 따라 예상되는 유형의 태그를 포함하는 모든 오브젝트로 대체하여 확장됩니다.

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.

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 오브젝트 태그 추가

이 절차에서는 **LVM** 오브젝트 태그를 추가하는 방법을 설명합니다.

사전 요구 사항

- **lvm2** 패키지가 설치되어 있습니다.
- 하나 이상의 물리 볼륨, 볼륨 그룹 또는 논리 볼륨이 생성됩니다.

절차

- 오브젝트 태그를 생성하려면 **LVM** 명령에 **--addtag** 옵션을 추가합니다.
  - 물리 볼륨에서 태그를 만들려면 옵션을 **pvchange** 명령에 추가합니다.
  - 볼륨 그룹에서 태그를 만들려면 옵션을 **KnativeServing change** 또는 **KnativeServing create** 명령에 추가합니다.
  - 논리 볼륨에서 태그를 생성하려면 옵션을 **lvchange** 또는 **lvcreate** 명령에 추가합니다.

16.4. LVM 오브젝트 태그 제거

이 절차에서는 **LVM** 오브젝트 태그를 제거하는 방법을 설명합니다.

사전 요구 사항

- **lvm2** 패키지가 설치되어 있습니다.
- 물리 볼륨, 볼륨 그룹 또는 논리 볼륨의 오브젝트 태그가 생성됩니다.

절차

- 오브젝트 태그를 삭제하려면 **LVM** 명령에 **--deltag** 옵션을 추가합니다.
  - 물리 볼륨에서 태그를 삭제하려면 **pvchange** 명령에 옵션을 추가합니다.
  - 볼륨 그룹에서 태그를 삭제하려면 **net change** 또는 **KnativeServingcreate** 명령에 옵션을 추가합니다.
  - 논리 볼륨에서 태그를 삭제하려면 옵션을 **lvchange** 또는 **lvcreate** 명령에 추가합니다.

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.

사전 요구 사항

- *사용자가 이 어셈블리를 따르기 전에 충족해야 하는 총알 조건 목록입니다.*
- *이 어셈블리를 시작하기 전에 사용자가 따라야 하는 다른 모듈 또는 어셈블리에 연결할 수도 있습니다.**You can also link to other modules or assembly the user must follow before starting this assembly.***
- *어셈블리에 사전 요구 사항이 없는 경우 섹션 제목과 총알을 삭제합니다.*

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 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.
  - 명령 출력의 상세 수준을 높이기 위해 **-v** 인수를 LVM 명령에 추가합니다. **vs s**를 추가하여 세부 정보 표시를 늘릴 수 있습니다. 예를 들어, 이러한 **v**의 최대 **4**개가 허용됩니다(예: **-vvv**).
  - **/etc/lvm/lvm.conf** 구성 파일의 로그 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. **-vvvv** 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/** 디렉토리에서 **LVM** 메타데이터의 마지막 백업을 검사하고 **/etc/lvm/archive/** 디렉터리에 보관된 버전을 검사합니다.

- 현재 구성 정보를 확인합니다.

```
# lvmconfig
```

- **/run/lvm/hints** 캐시 파일에서 물리 볼륨이 있는 장치의 레코드를 확인합니다.

추가 리소스

- **lvmdump(8) man page**

17.2. 실패한 LVM 장치에 대한 정보 표시

볼륨이 실패한 이유를 결정하는 데 도움이 되는 **LVM** 볼륨에 대한 정보를 표시할 수 있습니다.

절차

- **vgs** 또는 **lvs** 유틸리티를 사용하여 실패한 볼륨을 표시합니다.

예 17.1. 실패한 볼륨 그룹

이 예에서 볼륨 그룹 **myvg** 를 구성하는 장치 중 하나가 실패했습니다. 볼륨 그룹을 사용할 수 없지만 실패한 장치에 대한 정보를 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. 논리 볼륨 실패

이 예에서는 볼륨 그룹의 논리 볼륨이 실패하여 장치 중 하나가 실패했습니다. 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. 미러링된 논리 볼륨의 암호에 실패했습니다.

다음 예제에서는 미러링된 논리 볼륨의 가을에 실패한 경우 **KnativeServing s** 및 **lvs** 유틸리티의 명령 출력을 보여줍니다.

```
# vgs --all --options +devices

  VG    #PV #LV #SN Attr   VSize VFree Devices
  corey 4 4 0 rz-pnc 1.58T 1.34T my_mirror_mimage_0(0),my_mirror_mimage_1(0)
```

```
  corey 4 4 0 rz-pnc 1.58T 1.34T /dev/sdd1(0)
  corey 4 4 0 rz-pnc 1.58T 1.34T unknown device(0)
  corey 4 4 0 rz-pnc 1.58T 1.34T /dev/sdb1(0)
```

```
# lvs --all --options +devices

  LV                VG    Attr   LSize   Origin Snap%  Move Log          Copy%  Devices
  my_mirror corey mwi-a- 120.00G my_mirror_mlog 1.95
my_mirror_mimage_0(0),my_mirror_mimage_1(0)
  [my_mirror_mimage_0] corey iwi-ao 120.00G unknown device(0)
  [my_mirror_mimage_1] corey iwi-ao 120.00G /dev/sdb1(0)
  [my_mirror_mlog] corey lwi-ao 4.00M /dev/sdd1(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

씬 풀을 제거하면 **LVM**에서 작업을 취소할 수 없습니다.

17.4. 누락된 LVM 물리 볼륨의 메타데이터 찾기

물리 볼륨의 볼륨 그룹의 메타데이터 영역에 실수로 덮어쓰거나 삭제되는 경우 메타데이터 영역이 올바르지 않거나 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.

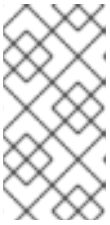
사전 요구 사항

- 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** 장치의 레이블을 물리 볼륨으로 지정합니다.

- **FmGRh3-zhok-iVI8-7qTD-S5BI-MAEN-NYM5Sk**의 **UUID**
- **VG_00050.vg** 에 포함된 메타데이터 정보는 볼륨 그룹의 가장 최근 아카이브 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 재동기화합니다.

```
# 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 그룹의 공간 사용량을 보고한 **LVM** 명령은 보고된 수를 **2** 자리로 반올림하여 사람이 읽을 수 있는 출력을 제공합니다. 여기에는 **KnativeServingdisplay** 및 **tekton 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 확장 영역 수를 지정하여 반올림 오류를 방지할 수 있습

니다.

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 그룹의 나머지 여유 공간의 백분율을 사용하도록 논리 볼륨을 확장할 수도 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 RAID 문제 해결

LVM RAID 장치의 다양한 문제를 해결하여 데이터 오류를 수정하거나 장치를 복구하거나, 실패한 장치를 교체할 수 있습니다.

17.8.1. RAID 논리 볼륨(RAID 스크럽링)에서 데이터 일관성 확인

LVM은 RAID 논리 볼륨에 대한 스크럽 지원을 제공합니다. RAID 스크럽은 배열의 모든 데이터 및 패리티 블록을 읽고 일치 여부를 확인하는 프로세스입니다.

절차

1. 선택 사항: 스크럽 프로세스에서 사용하는 I/O 대역폭을 제한합니다.

   RAID 스크럽 작업을 수행할 때 동기화 작업에 필요한 백그라운드 I/O는 볼륨 그룹 메타데이터 업데이트 등 다른 I/O를 LVM 장치에 분산시킬 수 있습니다. 이로 인해 다른 LVM 작업이 느려질 수 있습니다. 복구 제한을 구현하여 스크럽 작업의 비율을 제어할 수 있습니다.

   다음 단계에서 lvchange --syncaction 명령에 다음 옵션을 추가합니다.

   --maxrecoveryrate Rate[bBsSkKmMgG]

   작업이 고등 I/O 작업을 수행할 수 있도록 최대 복구 비율을 설정합니다. 복구 비율을 0으로 설정하면 작업이 바인딩되지 않습니다.

   --minrecoveryrate Rate[bBsSkKmMgG]

   중복 I/O가 과도한 I/O가 있는 경우에도 동기화 작업의 I/O가 최소 처리량을 유지하도록 최소 복구 비율을 설정합니다.

   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 vg/raid_lv
```

3. *배열의 불일치를 수정하십시오.*

```
# lvchange --syncaction repair vg/raid_lv
```

참고

***lvchange --syncaction repair** 작업은 **lvconvert --repair** 작업과 동일한 기능을 수행하지 않습니다.*

- ***lvchange --syncaction repair** 작업은 배열에서 백그라운드 동기화 작업을 시작합니다.*
- ***lvconvert --repair** 작업 복구 또는 미러 또는 **RAID** 논리 볼륨에서 실패한 장치를 교체하십시오.*

4. *선택 사항: 스크럽 작업에 대한 정보를 표시합니다.*

```
# lvs -o +raid_sync_action,raid_mismatch_count vg/lv
```

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

  idle

  모든 동기화 작업이 완료됨(없음 실행)

  resync

  어레이 초기화 또는 머신 장애 후 복구

  recover

  배열에서 장치 교체

  Check

  ***array inconsistencies**를 찾습니다.*

복구

불일치 검색 및 수정

- 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 9는 논리 볼륨의 상태를 표시하고 다음 지표를 지원합니다.
  - m (mismatches)은 RAID 논리 볼륨에 불일치가 있음을 나타냅니다. 이 문자는 평가 작업이 RAID의 일부가 일치하지 않는 것을 감지한 후에 표시됩니다.
  - R (refresh)은 RAID 배열의 장치에 오류가 발생했으며 LVM이 장치 레이블을 읽고 작동하는 장치를 고려하더라도 커널이 실패한 것으로 간주했음을 나타냅니다. 논리 볼륨을 새로 고쳐 장치를 사용할 수 있음을 커널에 알리거나 실패한 것으로 의심되는 경우 장치를 교체합니다.

추가 리소스

- 자세한 내용은 lvchange(8) 및 lvmraid(7) 도움말 페이지를 참조하십시오.

17.8.2. LVM RAID에서 실패한 장치

RAID는 기존 LVM 미러링과 같지 않습니다. LVM 미러링에 실패한 장치를 제거해야 하거나 미러링된 논리 볼륨이 중단되었습니다. RAID 배열은 실패한 장치로 계속 실행될 수 있습니다. 실제로 RAID1 이외의 RAID 유형의 경우 장치를 제거하면 더 낮은 수준 RAID(예: RAID6에서 RAID5로 변환하거나 RAID4 또는 RAID5에서 RAID0)로 변환하는 것을 의미합니다.

따라서 실패한 장치를 무조건 제거하고 교체를 할당하는 대신 LVM을 사용하면 lvconvert 명령의 --repair 인수를 사용하여 1단계 솔루션에서 RAID 볼륨에서 실패한 장치를 교체할 수 있습니다.

17.8.3. 논리 볼륨에서 실패한 RAID 장치 복구

LVM RAID 장치 오류가 일시적인 오류이거나 오류가 발생한 장치를 복구할 수 있는 경우 실패한 장치의 복구를 시작할 수 있습니다.

사전 요구 사항

- *이전에 실패한 장치가 이제 작동 중입니다.*

절차

- ***RAID** 장치가 포함된 논리 볼륨을 새로 고칩니다.*

```
# lvchange --refresh my_vg/my_lv
```

검증 단계

- *복구된 장치로 논리 볼륨을 검사합니다.*

```
# lvs --all --options name,devices,lv_attr,lv_health_status my_vg
```

*17.8.4. 논리 볼륨에서 실패한 **RAID** 장치 교체*

*이 절차에서는 **LVM RAID** 논리 볼륨에서 물리 볼륨 역할을 하는 실패한 장치를 대체합니다.*

사전 요구 사항

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

  *볼륨 그룹에서 사용 가능한 물리 볼륨이 충분한 물리 볼륨이 없는 경우, **KnativeServingextend** 유틸리티를 사용하여 충분히 큰 새 물리 볼륨을 추가합니다.*

절차

1. *다음 예에서 **RAID** 논리 볼륨은 다음과 같이 배치됩니다.*

```
# 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 실패하면 lvs 명령의 출력은 다음과 같습니다.*

```
# 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 교체하고 논리 볼륨을 표시합니다.*

```
# lvconvert --repair my_vg/my_lv

  /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.
Attempt to replace failed RAID images (requires full device resync)? [y/n]: y
  Faulty devices in my_vg/my_lv successfully replaced.
```

선택 사항: 오류가 발생한 장치를 대체하는 물리 볼륨을 수동으로 지정하려면 명령 끝에 물리 볼륨을 추가합니다.

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

4. *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-vIzA-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.*

5. *볼륨 그룹에서 실패한 장치를 제거합니다.*

```
# vgreduce --removemissing VG
```

*17.9. 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.*

*17.9.1. **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** 디렉토리에 여러 개의 장치 노드가 있으므로 모든 장치 노드에 동일한 **LVM** 메타데이터가 포함됩니다.*

*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.*

17.9.2. 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.

17.9.3. LVM 장치 필터

LVM 툴은 /dev 디렉토리에서 장치를 스캔하고 LVM 메타데이터에 대한 모든 장치를 확인합니다. /etc/lvm/lvm.conf 파일의 필터는 LVM을 스캔하는 장치를 제어합니다.

필터는 /dev 디렉토리 또는 /etc/lvm/lvm.conf 파일의 dir 키워드로 지정된 디렉터리에 LVM이 적용되는 패턴 목록입니다. 패턴은 임의의 문자로 구분된 정규식이며, 앞에 허용 또는 r 을 거부하기 위해 붙습니다. LVM이 장치를 수락하거나 거부(ignore)하는지 여부를 확인하는 목록의 첫 번째 정규식입니다. LVM은 패턴과 일치하지 않는 장치를 허용합니다.

다음은 모든 장치를 검사하는 필터의 기본 구성입니다.

```
filter = [ "a/.*/" ]
```

17.9.4. 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.

구성하는 필터는 LVM에서 root 볼륨 그룹과 다중 경로 장치가 있는 로컬 하드 드라이브와 같은 메타데이터를 확인하는 데 필요한 모든 장치를 포함해야 합니다. 다중 경로 장치의 기본 경로(예: /dev/sdb,/dev/sdd 등)를 거부하면 LVM에서 다중 경로 장치 자체에서 각 고유 메타데이터 영역을 찾으므로 이러한 중복 PV 경고를 방지할 수 있습니다.

-

*이 필터는 첫 번째 하드 드라이브와 **DM Multipath** 장치의 두 번째 파티션을 허용하지만 다른 모든 파티션은 거부됩니다.*

```
filter = [ "a|/dev/sda2$|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

- *이 필터는 모든 **HP SmartRegistryLogin** 컨트롤러 및 모든 **EMC PowerPath** 장치를 허용합니다.*

```
filter = [ "a|/dev/cciss/.*|", "a|/dev/emcpower.*|", "r|.*|" ]
```

- *이 필터는 첫 번째 **IDE** 드라이브 및 다중 경로 장치의 모든 파티션을 허용합니다.*

```
filter = [ "a|/dev/hda.*|", "a|/dev/mapper/mpath.*|", "r|.*|" ]
```

17.9.5. LVM 장치 필터 구성 적용

*이 절차에서는 **LVM** 스캔 장치를 제어하는 **LVM** 장치 필터의 구성을 변경합니다.*

사전 요구 사항

- *사용할 장치 필터 패턴을 준비합니다.*

절차

1. */etc/lvm/lvm.conf 파일을 수정하지 않고 장치 필터 패턴을 테스트합니다.*

   *--config 'devices{ filter = [장치 필터 패턴] }' 옵션과 함께 **LVM** 명령을 사용합니다. 예를 들면 다음과 같습니다.*

   ```
   # lvs --config 'devices{ filter = [ "a|/dev/emcpower.*|", "r|.*|" ] }'
   ```

2. *새 장치 필터 패턴을 사용하도록 /etc/lvm/lvm.conf 구성 파일의 filter 옵션을 편집합니다.*

3. *사용하려는 물리 볼륨 또는 볼륨 그룹이 없는 경우 새 구성과 함께 누락되어 있는지 확인합니다.*

   ```
   # pvscan
   ```

```
# vgscan
```

4. *재부팅 시 **LVM**이 필요한 장치만 검사하도록 **initramfs** 파일 시스템을 다시 빌드합니다.*

```
# dracut --force --verbose
```

17.9.6. 추가 리소스

- *13장. LVM 장치 스캔 제어*