Red Hat Enterprise Linux 8

RHEL에서 논리 볼륨 중복 제거 및 압축

LVM에 VDO를 배포하여 스토리지 용량 증가

Last Updated: 2024-08-03

Red Hat Enterprise Linux 8 RHEL에서 논리 볼륨 중복 제거 및 압축

LVM에 VDO를 배포하여 스토리지 용량 증가

법적 공지

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

LVM(Logical Volume Manager)의 VDO(Virtual Data Optimizer) 기능을 사용하여 중복 및 압축된 논리 볼륨을 관리합니다. VDO는 LVM 씬 프로비저닝 볼륨과 유사하게 LVM의 논리 볼륨(LV) 유형으로 관리할 수 있습니다. LVM(LVM-VDO)에 VDO를 배포하여 블록 액세스, 파일 액세스, 로컬 스토리지 및 원격 스토리지에 중복된 스토리지를 제공할 수 있습니다. 100%가 사용되는 VDO 볼륨의 물리적 공간을 피하기 위해 씬 프로비저닝된 VDO 볼륨을 구성할 수도 있습니다. VDO 볼륨을 LVM으로 가져온 후 LVM 도구를 사용하여 VDO 볼륨을 관리할 수 있습니다.

차례

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

Jira를 통해 피드백 제출 (등록 필요)

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

1장. LVM의 VDO 소개

VDO(가상 데이터 최적화 도구) 기능은 스토리지를 위한 인라인 블록 수준 중복 제거, 압축 및 씬 프로비저닝을 제공합니다. LVM 씬 프로비저닝 볼륨과 유사하게 VDO를 LVM(Logical Volume Manager) 논리 볼륨(LV) 유형으로 관리할 수 있습니다.

LVM(LVM-VDO)의 VDO 볼륨에는 다음 구성 요소가 포함됩니다.

VDO pool LV

- 이는 VDO LV의 데이터를 저장, 중복 제거 및 압축하는 백업 물리적 장치입니다. VDO 풀 LV는 VDO 볼륨의 물리적 크기를 설정합니다. 이 볼륨은 VDO 디스크에 저장할 수 있는 데이터 양입니다.
- 현재 각 VDO pool LV는 하나의 VDO LV만 보유할 수 있습니다. 결과적으로 VDO는 각 VDO LV를 별도로 중복하고 압축합니다. 별도의 LV에 저장된 중복 데이터는 동일한 VDO 볼륨의 데이터 최적화의 이점을 누릴 수 없습니다.

VDO LV

- VDO 풀 LV 상단에 프로비저닝된 가상 장치입니다. VDO LV는 VDO 볼륨의 프로비저닝된 논리 크기를 설정합니다. 이 볼륨은 중복 제거 및 압축이 발생하기 전에 애플리케이션에 쓸 수 있는 데이터 양입니다.

LVM 씬 프로비저닝 구현의 구조에 이미 익숙한 경우 Table 1.1을 참조하여 VDO의 다양한 측면이 시스템에 어떻게 제공되는지 이해할 수 있습니다.

표 1.1. LVM 및 LVM 씬 프로비저닝의 VDO의 구성 요소 비교

	물리적 장치	프로비저닝된 장치
LVM의 VDO	VDO pool LV	VDO LV
LVM 씬 프로비저닝	씬 풀	씬 볼륨

VDO는 씬 프로비저닝되므로 파일 시스템과 애플리케이션은 실제 사용 가능한 물리적 공간이 아니라 사용 중인 논리 공간만 볼 수 있습니다. 스크립팅을 사용하여 사용 가능한 물리적 공간을 모니터링하고 사용량이 임계값을 초과하는 경우 경고를 생성합니다. 사용 가능한 VDO 공간 모니터링에 대한 자세한 내용은 Monitoring VDO 섹션을 참조하십시오.

추가 리소스

- 스토리지 중복 제거 및 압축
- 씬 프로비저닝 볼륨 생성 및 관리(thin volumes)

2장. LVM-VDO 요구 사항

LVM의 VDO에는 배치 및 시스템 리소스에 대한 특정 요구 사항이 있습니다.

2.1. VDO 메모리 요구 사항

각 VDO 볼륨에는 두 개의 고유한 메모리 요구 사항이 있습니다.

VDO 모듈

VDO에는 고정된 38MB의 RAM과 몇 가지 변수 양이 필요합니다.

- 구성된 블록 맵 캐시 크기 1MB당 1.15MB의 RAM. 블록 맵 캐시에는 최소 150MB의 RAM이 필요합니다.
- 논리 공간 1TB당 1.6MB의 RAM.
- 볼륨에서 관리하는 물리적 스토리지의 1TB당 268MB의 RAM.

UDS 인덱스

UDS(Universal Deduplication Service)에는 최소 250MB의 RAM이 필요합니다. 이는 중복 제거에서 사용하는 기본 양이기도 합니다. 이 값은 인덱스에 필요한 스토리지 크기에도 영향을 미치므로 VDO 볼륨을 포맷할 때 값을 구성할 수 있습니다.
UDS 인덱스에 필요한 메모리는 인덱스 유형 및 중복 제거 창의 필수 크기에 따라 결정됩니다.

인덱스 유형	중복 제거 창	참고
밀도	1GB RAM당 1TB	1GB 밀도 지수는 일반적으로 최대 4TB의 물리적 스토리지에 충분합니다.
스파스	1GB RAM당 10TB	일반적으로 1GB 스파스 인덱스는 최대 40TB의 물리적 스토리지에 충분합니다.

참고

기본 설정 2GB slab 크기와 0.25 밀도 인덱스를 사용하는 VDO 볼륨의 최소 디스크 사용량에는 approx 4.7GB가 필요합니다. 이는 0% 중복 제거 또는 압축에서 쓸 수 있는 2GB의 물리적 데이터보다 약간 적습니다.

여기에서 최소 디스크 사용은 기본 slab 크기 및 dense 인덱스의 합계입니다.

UDS Sparse Indexing 기능은 VDO에 권장되는 모드입니다. 데이터의 시간적 로컬성에 의존하고 메모리에서 가장 관련성이 높은 인덱스 항목만 유지하려고 합니다. 스파스 인덱스를 사용하면 UDS는 동일한 양의 메모리를 사용하는 동시에 밀도가 있는 것보다 10배 큰 중복 제거 창을 유지할 수 있습니다.

스파스 인덱스가 가장 큰 범위를 제공하지만, 밀도가 더 많은 중복 제거 조언을 제공합니다. 동일한 양의 메모리에 대해 대부분의 워크로드에서 밀도 및 스파스 인덱스 간의 중복 제거 비율의 차이는 무시할 수 있습니다.

추가 리소스

- 물리적 크기 VDO 요구 사항의 예

2.2. VDO 스토리지 공간 요구 사항

최대 256TB의 물리 스토리지를 사용하도록 VDO 볼륨을 구성할 수 있습니다. 물리적 스토리지의 특정 부분만 데이터를 저장하는 데 사용할 수 있습니다. 이 섹션에서는 VDO 관리 볼륨의 사용 가능한 크기를 결정하는 계산 방법을 제공합니다.

VDO는 두 가지 유형의 VDO 메타데이터 및 UDS 인덱스에 대한 스토리지가 필요합니다.

- 첫 번째 유형의 VDO 메타데이터는 약 1MB의 *물리적 스토리지와* slab당 1MB를 추가로 사용합니다.
- 두 번째 유형의 VDO 메타데이터는 1GB의 *논리 스토리지* 마다 약 1.25MB를 사용하고 가장 가까운 슬랩으로 반올림합니다.
- UDS 인덱스에 필요한 스토리지 양은 인덱스 유형 및 인덱스에 할당된 RAM 크기에 따라 달라집니다. 1GB RAM마다 밀도가 높은 UDS 인덱스는 17GB의 스토리지를 사용하며 스파스 UDS 인덱스는 170GB의 스토리지를 사용합니다.

추가 리소스

- 물리적 크기 VDO 요구 사항의 예
- VDO의 Slab 크기

2.3. 물리적 크기 VDO 요구 사항의 예

다음 표에서는 기본 볼륨의 물리적 크기에 따라 VDO의 대략적인 시스템 요구 사항을 제공합니다. 각 테이블에는 기본 스토리지 또는 백업 스토리지와 같이 의도한 배포에 적합한 요구 사항이 나열되어 있습니다.

정확한 숫자는 VDO 볼륨 구성에 따라 다릅니다.

기본 스토리지 배포

1차 스토리지의 경우 UDS 인덱스는 물리적 크기의 크기에서 25% 사이입니다.

표 2.1. 기본 스토리지에 대한 스토리지 및 메모리 요구 사항

물리 크기	RAM 사용량: UDS	RAM 사용량: VDO	디스크 사용량	인덱스 유형
10GB–1TB	250MB	472MB	2.5GB	밀도
2–10TB	1GB	3GB	10GB	밀도
	250MB		22GB	스파스
11–50TB	2GB	14GB	170GB	스파스
51–100TB	3GB	27GB	255GB	스파스
101–256TB	12GB	69GB	1020GB	스파스

백업 스토리지 배포

백업 스토리지의 경우 UDS 인덱스는 백업 세트의 크기를 처리하지만 실제 크기보다 크지는 않습니다. 향후 백업 세트 또는 물리적 크기가 증가할 것으로 예상되는 경우 인덱스 크기로 고려합니다.

표 2.2. 백업 스토리지에 대한 스토리지 및 메모리 요구 사항

물리 크기	RAM 사용량: UDS	RAM 사용량: VDO	디스크 사용량	인덱스 유형
10GB–1TB	250MB	472MB	2.5GB	밀도
2–10TB	2GB	3GB	170GB	스파스
11–50TB	10GB	14GB	850GB	스파스
51–100TB	20GB	27GB	1700GB	스파스
101–256TB	26GB	69GB	3400GB	스파스

2.4. 스토리지 스택에 LVM-VDO 배치

VDO 논리 볼륨 아래에 특정 스토리지 계층을 배치하고 그 위의 다른 스토리지 계층을 배치해야 합니다.

VDO 상단에 씩 프로비저닝된 계층을 배치할 수 있지만 이 경우 씩 프로비저닝 보장을 사용할 수 없습니다. VDO 계층은 씬 프로비저닝되므로 씬 프로비저닝의 영향은 위의 모든 계층에 적용됩니다. VDO 볼륨을 모니터링하지 않으면 VDO보다 씩 프로비저닝된 볼륨의 물리 공간이 부족할 수 있습니다.

다음 계층에서 지원되는 배치는 VDO 아래에 있습니다. VDO 위에 배치하지 마십시오.

- DM Multipath
- DM 암호화
- 소프트웨어 RAID(LVM 또는 MD RAID)

다음 구성은 **지원되지 않습니다**.

- 루프백 장치 상단에 있는 VDO
- VDO 상단에 암호화된 볼륨
- VDO 볼륨의 파티션
- RAID(예: LVM RAID, MD RAID 또는 VDO 볼륨 상단에 있는 기타 유형)
- LVM-VDO에 Ceph Storage 배포

추가 리소스

- LVM 볼륨 지식 베이스 스택

3장. 중복 제거 및 압축된 논리 볼륨 생성

VDO 기능을 사용하여 데이터를 삭제하고 압축하는 LVM 논리 볼륨을 생성할 수 있습니다.

3.1. LVM-VDO 배포 시나리오

다음과 같은 다양한 방법으로 LVM(LVM-VDO)에 VDO를 배포할 수 있습니다.

- 블록 액세스
- 파일 액세스
- 로컬 스토리지
- 원격 스토리지

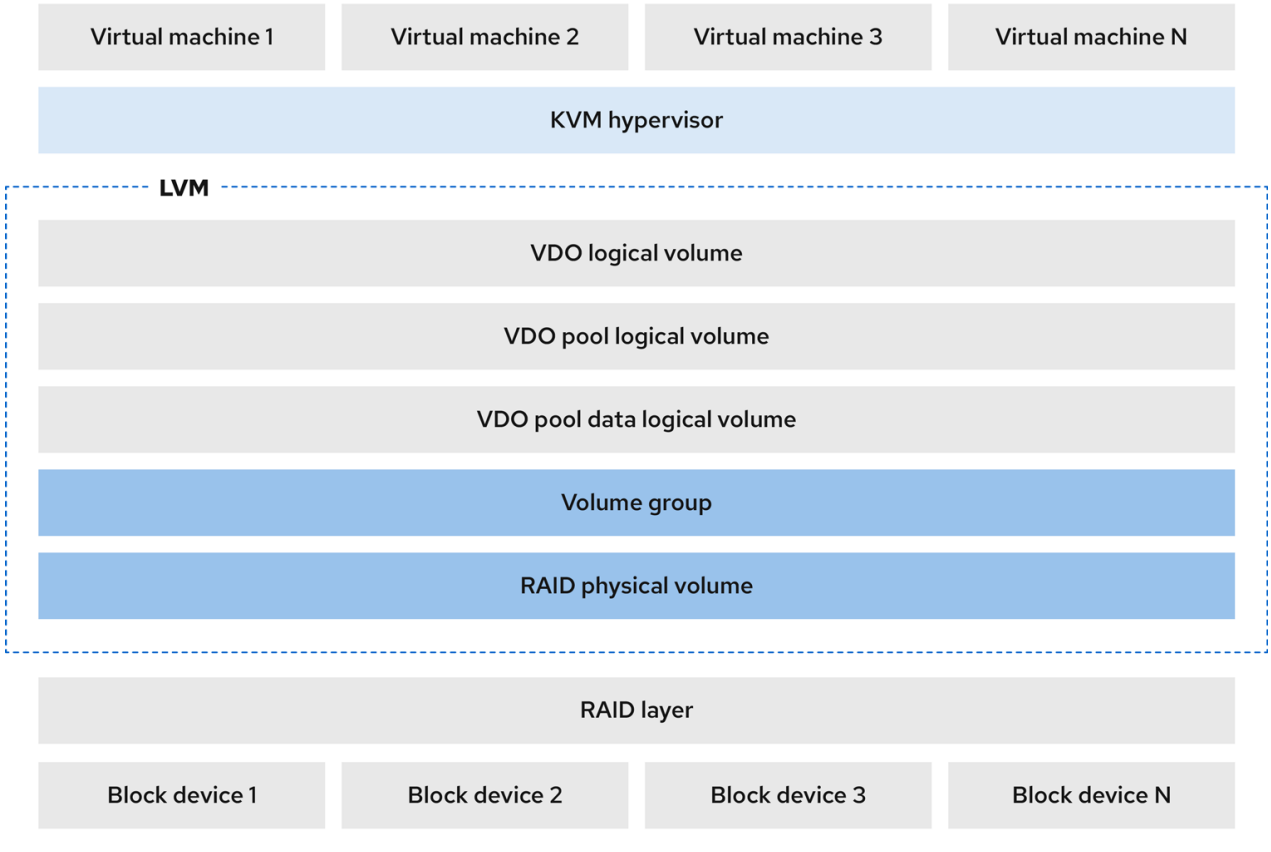
LVM-VDO는 중복 제거 스토리지를 일반 논리 볼륨(LV)으로 노출하므로 표준 파일 시스템, iSCSI 및 FC 대상 드라이버 또는 통합 스토리지와 함께 사용할 수 있습니다.

참고

현재 LVM-VDO에 Ceph Storage를 배포하는 것은 지원되지 않습니다.

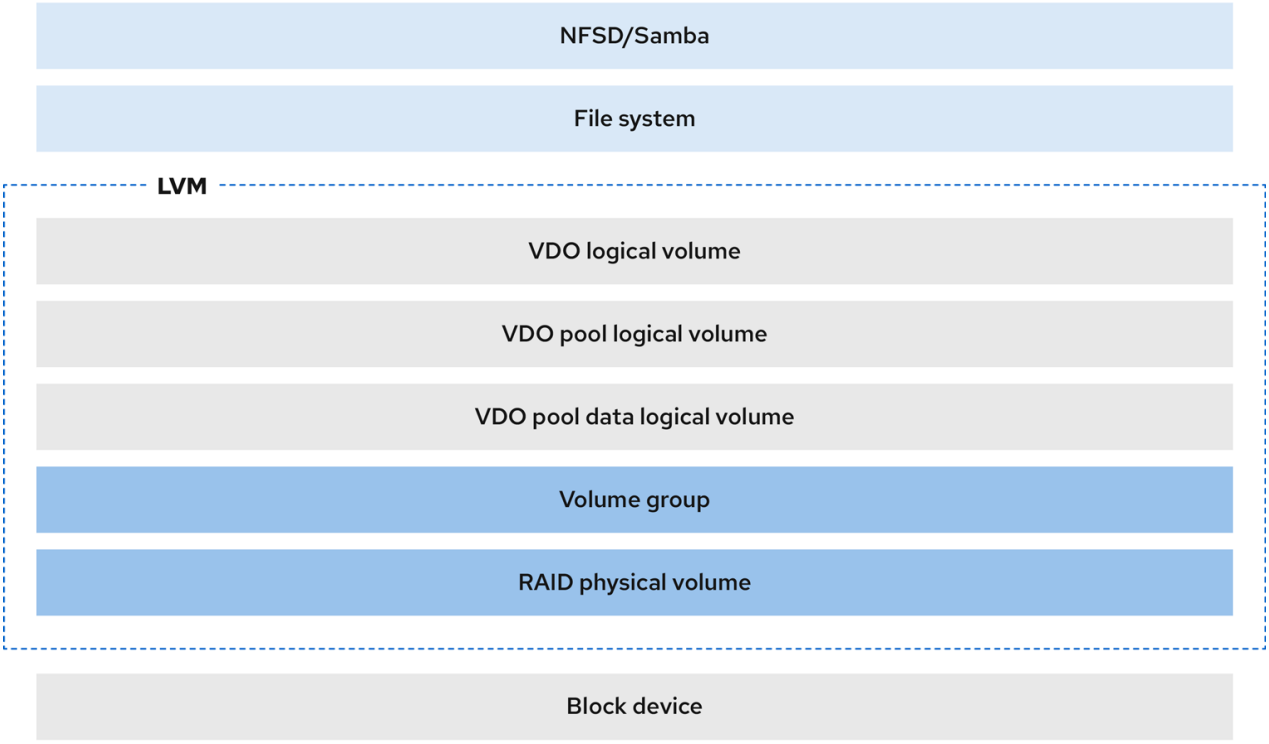
KVM

직접 연결된 스토리지로 구성된 KVM 서버에 LVM-VDO를 배포할 수 있습니다.



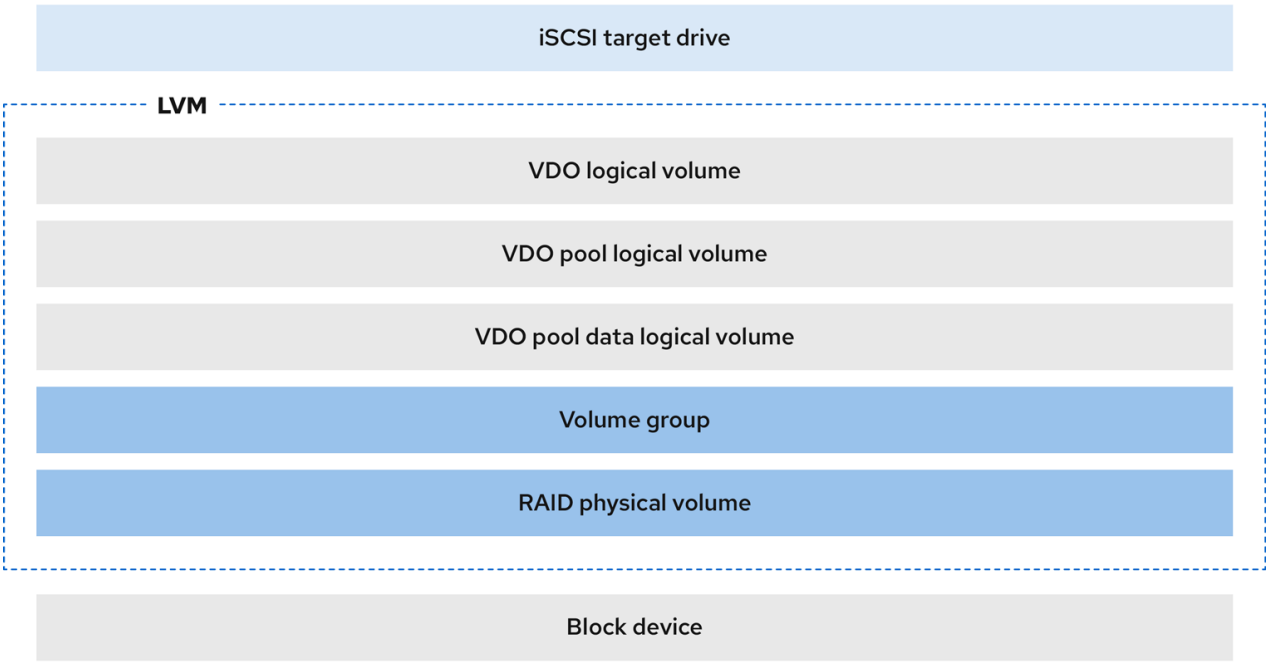
파일 시스템

VDO LV 상단에 파일 시스템을 생성하고 NFS 서버 또는 Samba를 사용하여 NFS 또는 CIFS 사용자에게 노출할 수 있습니다.



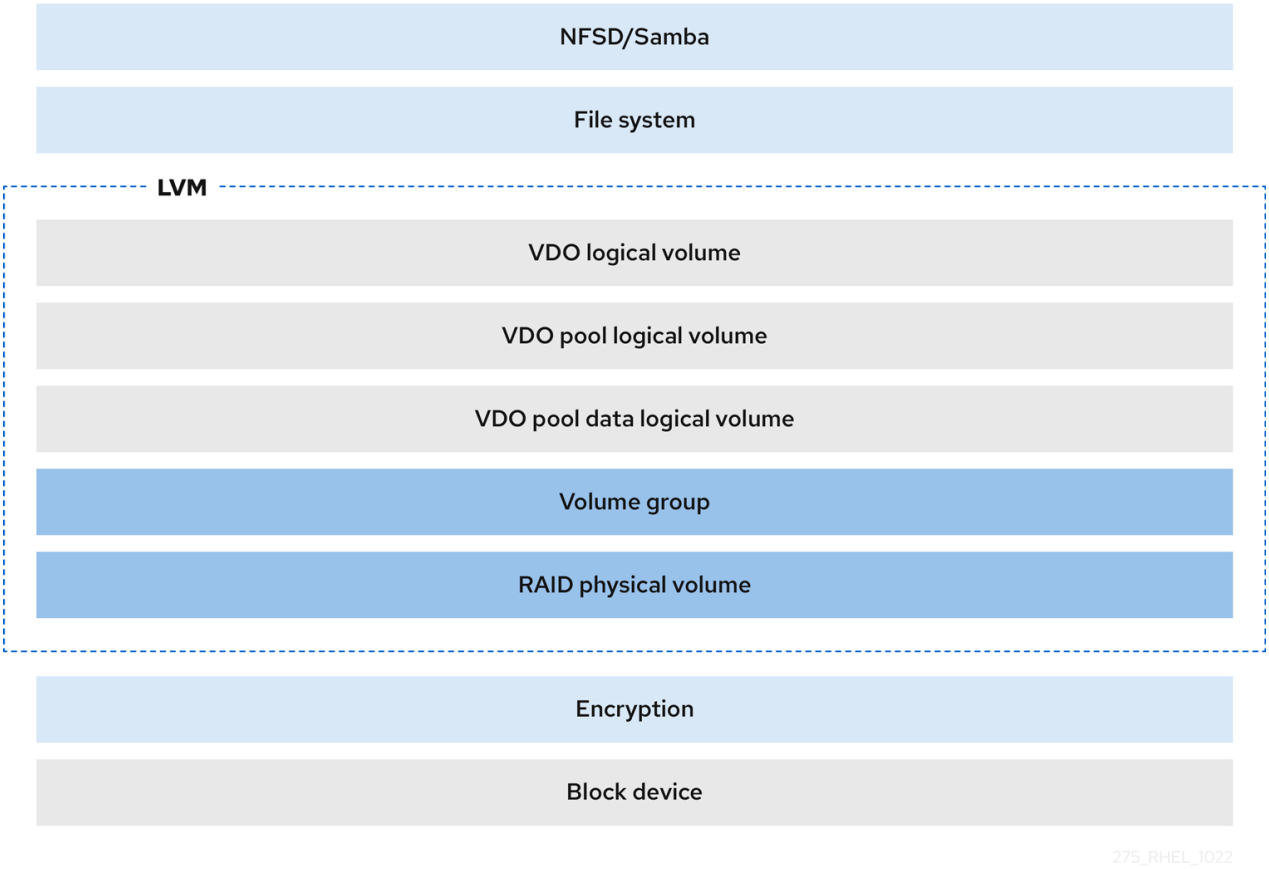
iSCSI 대상

VDO LV 전체를 iSCSI 대상으로 원격 iSCSI 이니시에이터로 내보낼 수 있습니다.



Encryption

DM Crypt와 같은 DM(Device Mapper) 메커니즘은 VDO와 호환됩니다. VDO LV 볼륨을 암호화하면 데이터 보안이 보장되고 VDO LV 위의 파일 시스템은 여전히 중복 제거됩니다.



중요

VDO LV 위의 암호화 계층을 적용하면 데이터 중복 제거가 거의 없습니다. VDO에서 중복을 제거하기 전에 암호화를 사용하면 중복 블록이 서로 다릅니다.

항상 VDO LV 아래에 암호화 계층을 배치합니다.

3.2. LVM-VDO 볼륨의 물리 및 논리적 크기

이 섹션에서는 VDO가 활용할 수 있는 물리적 크기, 사용 가능한 물리적 크기 및 VDO에 대해 설명합니다.

물리 크기

이 크기는 VDO 풀 LV에 할당된 물리 확장 영역과 동일합니다. VDO는 다음을 위해 이 스토리지를 사용합니다.

- 중복 제거 및 압축 가능한 사용자 데이터
- UDS 인덱스와 같은 VDO 메타데이터

사용 가능한 물리 크기

이것은 VDO가 사용자 데이터에 사용할 수 있는 물리적 크기의 부분입니다.
실제 크기와 동일한 값으로 반올림되며 slab 크기의 배수로 반올림됩니다.

논리 크기

이는 VDO LV가 애플리케이션에 제공하는 프로비저닝된 크기입니다. 일반적으로 사용 가능한 실제 크기보다 큽니다. VDO는 현재 절대 최대 논리 크기가 4 PB인 물리 볼륨의 최대 254배의 논리 크기를 지원합니다.
VDO LV(논리 볼륨)를 설정할 때 VDO LV에서 제공하는 논리 스토리지 크기를 지정합니다. 활성 VM 또는 컨테이너를 호스팅할 때, Red Hat은 10:1 논리적 비율과 물리적인 비율로 스토리지를 프로비저닝하는 것이 좋습니다. 즉, 1TB의 물리적 스토리지를 사용하는 경우 10TB의 논리적 스토리지가 됩니다.

virtual **size** 옵션을 지정하지 않으면 VDO는 볼륨을 **1:1** 비율로 프로비저닝합니다. 예를 들어, 20GB VDO 풀 LV 상단에 VDO LV를 배치하면 기본 인덱스 크기가 사용되는 경우 VDO는 UDS 인덱스용으로 2.5GB를 예약합니다. 나머지 17.5GB는 VDO 메타데이터 및 사용자 데이터에 대해 제공됩니다. 따라서 사용할 수 있는 스토리지가 17.5GB를 넘지 않으며 실제 VDO 볼륨을 구성하는 메타데이터로 인해 더 적을 수 있습니다.

추가 리소스

- 물리적 크기 VDO 요구 사항의 예

3.3. VDO의 슬랩 크기

VDO 볼륨의 물리 스토리지는 여러 슬랩으로 나뉩니다. 각 조각은 물리적 공간의 인접 지역입니다. 지정된 볼륨에 대한 모든 슬래브는 크기가 같으며 2의 배수 128MB 최대 32GB의 거듭제곱이 될 수 있습니다.

기본 slab 크기는 작은 테스트 시스템에서 VDO를 쉽게 평가할 수 있도록 2GB입니다. 단일 VDO 볼륨은 최대 8192 개의 슬래브를 보유할 수 있습니다. 따라서 2GB의 slab이 있는 기본 구성에서 허용되는 최대 물리적 스토리지는 16TB입니다. 32GB slab을 사용하는 경우 허용되는 최대 물리 스토리지는 256TB입니다. VDO는 항상 메타데이터를 위해 최소 하나의 slab을 예약하므로 예약된 슬랩을 사용자 데이터를 저장하는 데 사용할 수 없습니다.

Slab 크기는 VDO 볼륨의 성능에 영향을 미치지 않습니다.

표 3.1. 물리 볼륨 크기 별 권장 VDO 슬래브 크기

물리 볼륨 크기	권장 슬랩 크기
10~99GB	1GB
100GB - 1TB	2GB
2-256 TB	32GB

참고

기본 설정 2GB slab 크기와 0.25 밀도 인덱스를 사용하는 VDO 볼륨의 최소 디스크 사용량에는 approx 4.7GB가 필요합니다. 이는 0% 중복 제거 또는 압축에서 쓸 수 있는 2GB의 물리적 데이터보다 약간 적습니다.

여기에서 최소 디스크 사용은 기본 slab 크기 및 dense 인덱스의 합계입니다.

lvcreate 명령에 **--config 'allocation/vdo_slab_size_mb=*size-in-megabytes*'** 옵션을 제공하여 slab 크기를 제어할 수 있습니다.

3.4. VDO 설치

이 절차에서는 VDO 볼륨을 생성, 마운트 및 관리하는 데 필요한 소프트웨어를 설치합니다.

절차

- VDO 소프트웨어를 설치합니다.

```
# yum install lvm2 kmod-kvdo vdo
```

3.5. LVM-VDO 볼륨 생성

이 절차에서는 VDO 풀 LV에 VDO 논리 볼륨(LV)을 생성합니다.

사전 요구 사항

- VDO 소프트웨어를 설치합니다. 자세한 내용은 VDO 설치를 참조하십시오.
- 사용 가능한 스토리지 용량이 있는 LVM 볼륨 그룹이 시스템에 있습니다.

절차

1. VDO LV의 이름을 선택합니다(예: **vdo1).** 시스템에서 각 VDO LV에 다른 이름 및 장치를 사용해야 합니다.
   다음 모든 단계에서 *vdo-name* 을 이름으로 바꿉니다.
2. VDO LV를 생성합니다.

```
# lvcreate --type vdo \
        --name vdo-name
        --size physical-size
        --virtualsize logical-size \
        vg-name
```

   - *vg-name* 을 VDO LV를 배치하려는 기존 LVM 볼륨 그룹의 이름으로 바꿉니다.
   - *logical-size* 를 VDO LV가 제공하는 논리 스토리지 양으로 바꿉니다.
   - 물리적 크기가 16TiB보다 크면 다음 옵션을 추가하여 볼륨의 slab 크기를 32GiB로 늘립니다.

```
--config 'allocation/vdo_slab_size_mb=32768'
```

     16TiB보다 큰 실제 크기에서 2GiB의 기본 slab 크기를 사용하는 경우 **lvcreate** 명령이 실패하고 다음 오류가 발생합니다.

```
ERROR - vdoformat: formatVDO failed on '/dev/device': VDO Status: Exceeds maximum number of slabs supported
```

예 3.1. 컨테이너 스토리지를 위한 VDO LV 생성

예를 들어 1TB VDO 풀 LV에서 컨테이너 스토리지용 VDO LV를 생성하려면 다음을 사용합니다.

```
# lvcreate --type vdo \
        --name vdo1
        --size 1T
        --virtualsize 10T \
        vg-name
```

중요

VDO 볼륨을 생성할 때 오류가 발생하면 볼륨을 제거하여 정리합니다.

3. VDO LV에 파일 시스템을 생성합니다.
   - XFS 파일 시스템의 경우:

     ```
     # mkfs.xfs -K /dev/vg-name/vdo-name
     ```

   - ext4 파일 시스템의 경우:

     ```
     # mkfs.ext4 -E nodiscard /dev/vg-name/vdo-name
     ```

추가 리소스

- **lvmvdo(7)** 도움말 페이지

3.6. 웹 콘솔에서 VDO 볼륨 생성

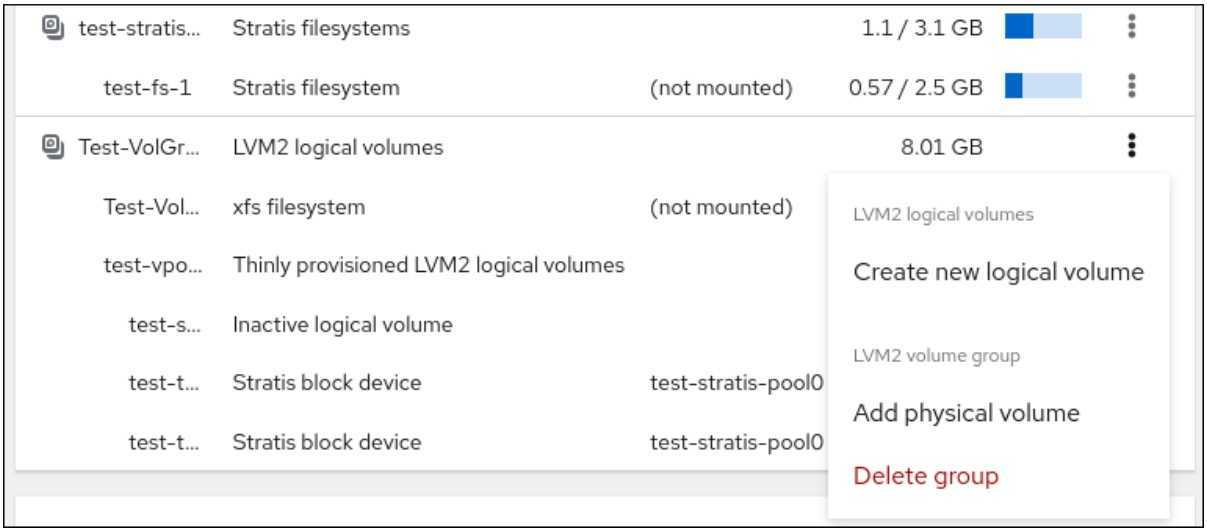
RHEL 웹 콘솔에서 VDO 볼륨을 생성합니다.

사전 요구 사항

- RHEL 8 웹 콘솔이 설치되고 액세스할 수 있습니다.
  자세한 내용은 웹 콘솔 설치를 참조하십시오.
- **cockpit-storaged** 패키지가 시스템에 설치됩니다.
- VDO를 생성할 LVM2 그룹입니다.

절차

1. RHEL 8 웹 콘솔에 로그인합니다.
   자세한 내용은 웹 콘솔에 로그인 을 참조하십시오.
2. 스토리지를 **클릭합니다**.
3. VDO 볼륨을 생성할 LVM2 그룹 옆에 있는 Cryostat 메뉴 버튼을 클릭합니다.



4. **Purpose** 필드 옆에 있는 드롭다운 메뉴에서 **VDO 파일 시스템 볼륨** 을 선택합니다.

5. **이름** 필드에 공백 없이 VDO 볼륨의 이름을 입력합니다.

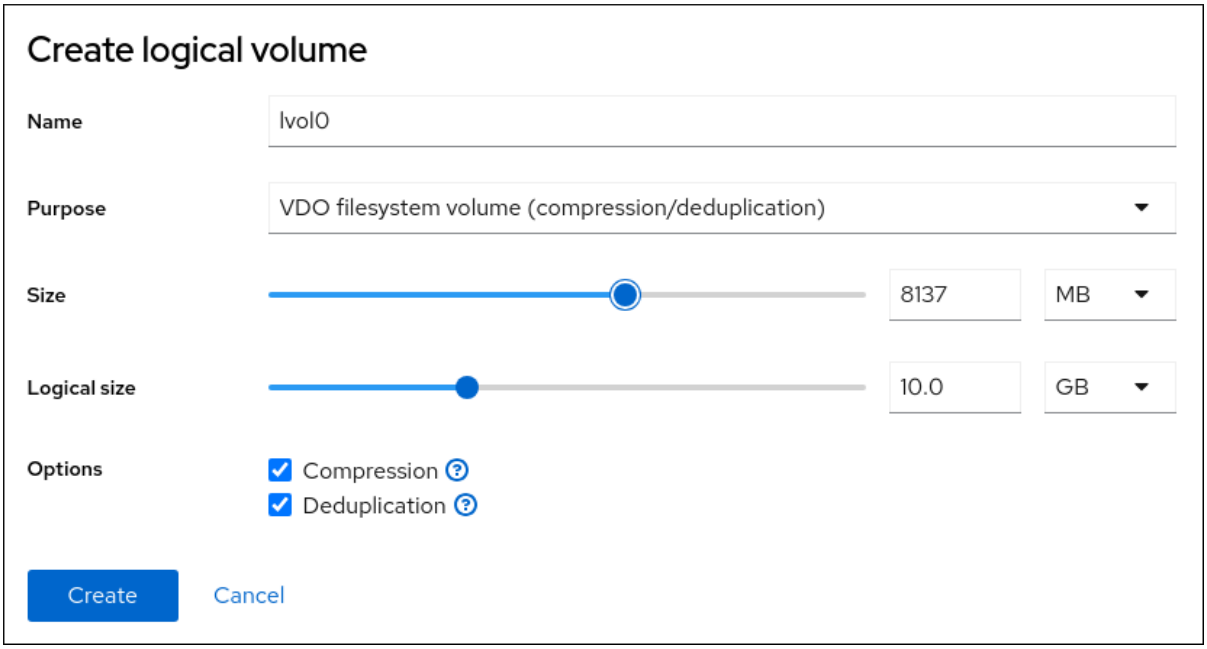
6. **논리 크기** 표시줄에서 VDO 볼륨의 크기를 설정합니다. 10회 이상 확장할 수 있지만 VDO 볼륨을 생성하는 목적을 고려하십시오.

   - 활성 VM 또는 컨테이너 스토리지의 경우 볼륨의 물리 크기보다 10배인 논리 크기를 사용합니다.
   - 오브젝트 스토리지의 경우 볼륨의 물리 크기보다 3배인 논리 크기를 사용합니다.

   자세한 내용은 Deploying VDO 를 참조하십시오.

7. **압축** 옵션을 선택합니다. 이 옵션은 다양한 파일 형식을 효율적으로 줄일 수 있습니다.
   자세한 내용은 LVM-VDO 볼륨에서 압축 설정 변경을 참조하십시오.

8. **중복 제거** 옵션을 선택합니다.
   이 옵션은 중복 블록의 여러 복사본을 제거하여 스토리지 리소스의 사용을 줄입니다. 자세한 내용은 LVM-VDO 볼륨에서 중복 제거 설정 변경을 참조하십시오.



검증 단계

- **Storage** 섹션에 새 VDO 볼륨이 표시되는지 확인합니다. 그런 다음 파일 시스템으로 포맷할 수 있습니다.

3.7. 웹 콘솔에서 VDO 볼륨 포맷

VDO 볼륨은 물리 드라이브 역할을 합니다. 이를 사용하려면 파일 시스템으로 포맷해야 합니다.

주의

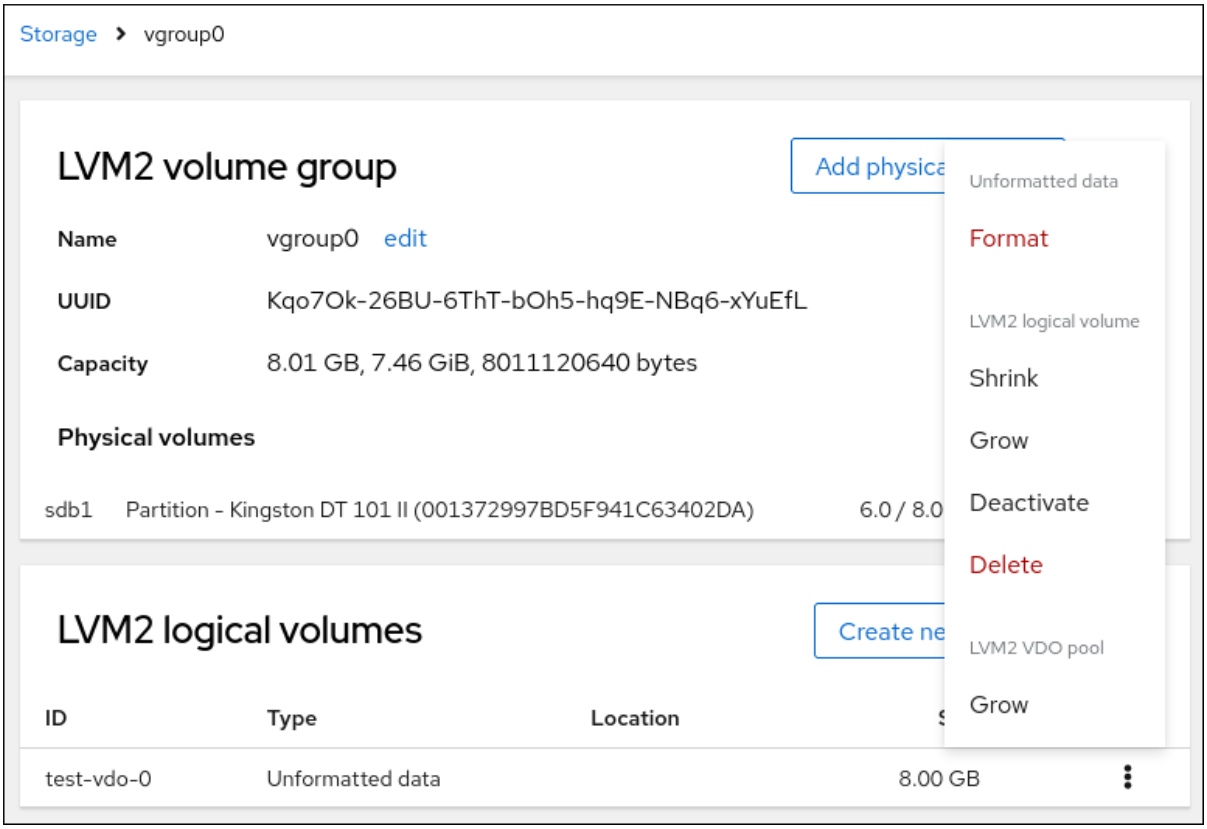
포맷은 볼륨의 모든 데이터를 지웁니다.

사전 요구 사항

- RHEL 8 웹 콘솔이 설치되고 액세스할 수 있습니다.
  자세한 내용은 웹 콘솔 설치를 참조하십시오.
- **cockpit-storaged** 패키지가 시스템에 설치됩니다.
- VDO 볼륨이 생성됩니다.

절차

1. RHEL 8 웹 콘솔에 로그인합니다. 자세한 내용은 웹 콘솔에 로그인 을 참조하십시오.
2. 스토리지를 **클릭합니다**.
3. 포맷할 VDO 볼륨이 포함된 LVM2 볼륨 그룹을 클릭합니다.
4. 포맷할 VDO **볼륨이** 포함된 줄 끝에 있는 메뉴 버튼 Cryostat를 클릭합니다.
5. **형식** 을 클릭합니다.



6. **이름** 필드에 논리 볼륨 이름을 입력합니다.

7. **마운트 지점** 필드에서 마운트 경로를 추가합니다.

8. 기본적으로 웹 콘솔은 이 대화 상자를 완료한 후 디스크 헤더만 다시 작성합니다. 이 옵션의 장점은 포맷 속도입니다. **0으로 기존 데이터를 덮어쓰**는 경우 웹 콘솔은 0으로 전체 디스크를 다시 작성합니다. 이 옵션은 프로그램이 전체 디스크를 통과해야하기 때문에 느려집니다. 디스크에 중요한 데이터가 포함되어 있고 다시 작성하려는 경우 이 옵션을 사용합니다.

9. **유형** 드롭다운 메뉴에서 파일 시스템을 선택합니다.

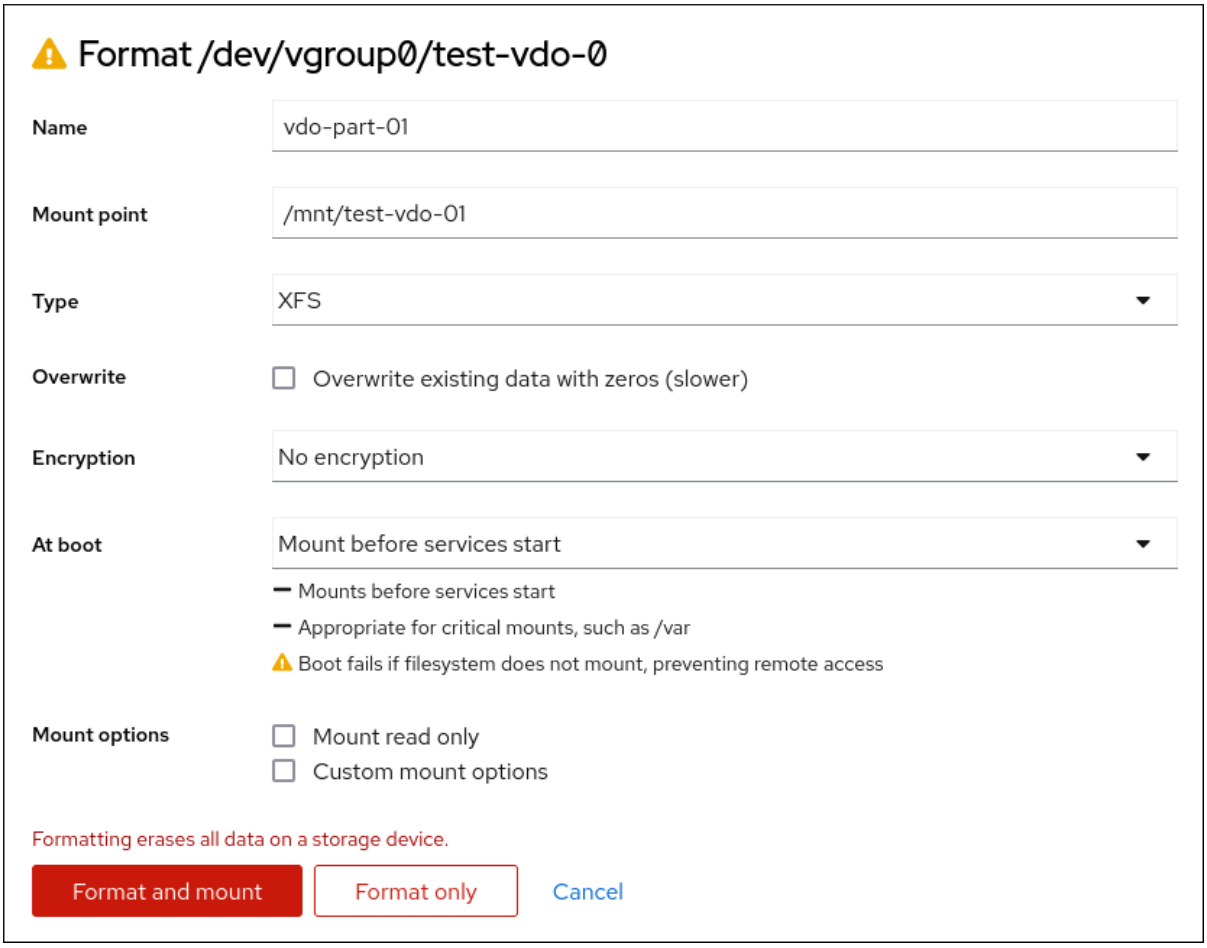
   - 기본 옵션인 **XFS** 파일 시스템은 대규모 논리 볼륨을 지원하며, 중단 없이 물리적 드라이브를 온라인 상태로 전환하며 증가합니다.
     XFS는 볼륨 축소를 지원하지 않습니다. 따라서 XFS로 포맷된 볼륨의 크기를 줄일 수 없습니다.

   - **ext4** 파일 시스템은 논리 볼륨을 지원하며, 중단 없이 물리적 드라이브를 온라인 상태로 전환, 증가 및 축소합니다.

   LUKS(Linux Unified Key Setup) 암호화가 있는 버전을 선택하여 암호를 사용하여 볼륨을 암호화할 수도 있습니다.

10. **부팅 시** 드롭다운 메뉴에서 볼륨을 마운트할 시기를 선택합니다.

11. **형식 및 마운트** 또는 **형식만** 클릭합니다.
    포맷은 사용된 포맷 옵션 및 볼륨 크기에 따라 몇 분이 걸릴 수 있습니다.



검증

- 성공적으로 완료되면 **스토리지** 탭과 LVM2 볼륨 그룹 탭에서 포맷된 VDO 볼륨의 세부 정보를 확인할 수 있습니다.

3.8. 웹 콘솔에서 VDO 볼륨 확장

RHEL 8 웹 콘솔에서 VDO 볼륨을 확장합니다.

사전 요구 사항

- RHEL 8 웹 콘솔이 설치되고 액세스할 수 있습니다.
  자세한 내용은 웹 콘솔 설치를 참조하십시오.
- **cockpit-storaged** 패키지가 시스템에 설치됩니다.
- VDO 볼륨이 생성됩니다.

절차

1. RHEL 8 웹 콘솔에 로그인합니다.
   자세한 내용은 웹 콘솔에 로그인 을 참조하십시오.
2. 스토리지를 **클릭합니다**.
3. VDO **장치** 상자에서 VDO 볼륨을 클릭합니다.
4. VDO 볼륨 세부 정보에서 **Grow** 버튼을 클릭합니다.

5. VDO **대화 상자의 논리 크기 확장 대화 상자에서 VDO**볼륨의 논리 크기를 확장합니다.
6. 확장 을 **클릭합니다**.

검증 단계

- 새 크기에 대한 VDO 볼륨 세부 정보를 확인하여 변경 사항이 성공했는지 확인합니다.

3.9. LVM-VDO 볼륨 마운트

이 절차에서는 LVM-VDO 볼륨에 파일 시스템을 수동으로 또는 영구적으로 마운트합니다.

사전 요구 사항

- LVM-VDO 볼륨이 시스템에 있습니다. 자세한 내용은 LVM-VDO 볼륨 생성을 참조하십시오.

절차

- LVM-VDO 볼륨에 파일 시스템을 수동으로 마운트하려면 다음을 사용합니다.

```
# mount /dev/vg-name/vdo-name mount-point
```

- 부팅 시 자동으로 마운트되도록 파일 시스템을 구성하려면 **/etc/fstab** 파일에 행을 추가합니다.
  - XFS 파일 시스템의 경우:

```
/dev/vg-name/vdo-name mount-point xfs defaults 0 0
```

  - ext4 파일 시스템의 경우:

```
/dev/vg-name/vdo-name mount-point ext4 defaults 0 0
```

  LVM-VDO 볼륨이 iSCSI와 같은 네트워크가 필요한 블록 장치에 있는 경우 **_netdev** 마운트 옵션을 추가합니다. 네트워크가 필요한 iSCSI 및 기타 블록 장치의 경우 **_netdev** 마운트 옵션에 대한 정보는 **systemd.mount(5)** 도움말 페이지를 참조하십시오.

추가 리소스

- **systemd.mount(5)** 도움말 페이지

3.10. LVM-VDO 볼륨에서 압축 설정 변경

기본적으로 VDO 풀 논리 볼륨(LV)의 압축이 활성화됩니다. CPU 사용량을 저장하려면 이를 비활성화할 수 있습니다. **lvchange** 명령을 사용하여 압축을 활성화하거나 비활성화합니다.

사전 요구 사항

- LVM-VDO 볼륨이 시스템에 있습니다.

절차

1. 논리 볼륨의 압축 상태를 확인합니다.

```
# lvs -o+vdo_compression,vdo_compression_state
  LV         VG         Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync
Convert VDOCompression VDOCompressionState
  vdo_name vg_name vwi-a-v---   1.00t vpool0        0.00                                     enabled
online
  vpool0   vg_name dwi------- <15.00g                20.03                                 enabled
online
```

2. VDOPoolLV의 압축을 비활성화합니다.

```
# lvchange --compression n vg-name/vdopoolname
```

압축을 활성화하려면 **n** 대신 **y** 옵션을 사용합니다.

검증

- 압축의 현재 상태를 확인합니다.

```
# lvs -o+vdo_compression,vdo_compression_state
  LV         VG         Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync
Convert VDOCompression VDOCompressionState
  vdo_name vg_name vwi-a-v---   1.00t vpool0        0.00
offline
  vpool0   vg_name dwi------- <15.00g                20.03                                 offline
```

추가 리소스

- **lvmvdo(7)** 도움말 페이지
- **lvcreate(8)** 도움말 페이지

3.11. LVM-VDO 볼륨에서 중복 제거 설정 변경

기본적으로 VDO 풀 논리 볼륨(LV)의 중복 제거가 활성화됩니다. 메모리를 저장하려면 중복 제거를 비활성화할 수 있습니다. **lvchange** 명령을 사용하여 중복 제거를 활성화하거나 비활성화합니다.

참고

VDO가 지속적인 병렬 I/O 작업을 처리하는 방식으로 인해 VDO 볼륨은 해당 작업 내에서 중복 데이터를 계속 식별합니다. 예를 들어 VM 복제 작업이 진행 중이고 VDO 볼륨에 여러 개의 중복 블록이 가까운 경우 해당 볼륨은 여전히 중복 제거를 사용하여 일부 공간을 절약할 수 있습니다. 볼륨의 인덱스 상태는 프로세스에 영향을 미치지 않습니다.

사전 요구 사항

- LVM-VDO 볼륨이 시스템에 있습니다.

절차

1. 논리 볼륨의 중복 제거 상태를 확인합니다.

```
# lvs -o+vdo_deduplication,vdo_index_state
  LV         VG         Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync
```

```
Convert VDODeduplication VDOIndexState
  vdo_name vg_name vwi-a-v---   1.00t vpool0       0.00                                    enabled
online
  vpool0   vg_name dwi------- <15.00g               20.03                                   enabled
online
```

2. VDOPoolLV의 중복 제거를 비활성화합니다.

```
# lvchange --deduplication n vg-name/vdopoolname
```

중복 제거를 활성화하려면 **n** 대신 **y** 옵션을 사용합니다.

검증

- 중복 제거의 현재 상태를 확인합니다.

```
# lvs -o+vdo_deduplication,vdo_index_state
  LV       VG       Attr       LSize   Pool   Origin Data%  Meta%  Move Log Cpy%Sync
Convert VDODeduplication VDOIndexState
  vdo_name vg_name vwi-a-v---   1.00t vpool0       0.00
closed
  vpool0   vg_name dwi------- <15.00g               20.03
closed
```

추가 리소스

- **lvmvdo(7)** 도움말 페이지
- **lvcreate(8)** 도움말 페이지

3.12. 가상 데이터 최적화 도구를 사용하여 씬 프로비저닝 관리

VDO 볼륨의 물리적 공간 사용량이 100%에 적용되는 조건을 해결하기 위해 씬 프로비저닝된 VDO 볼륨을 구성하여 물리 공간의 향후 확장을 준비할 수 있습니다. 예를 들어, **lvcreate** 작업에서 **-l 100%FREE** 를 사용하는 대신 '95%FREE'를 사용하여 나중에 복구할 수 있도록 예약된 공간이 있는지 확인합니다. 이 절차에서는 다음 문제를 해결하는 방법을 설명합니다.

- 볼륨이 공간이 부족합니다.
- 파일 시스템이 읽기 전용 모드로 전환
- 볼륨에서 보고한 ENOSPC

참고

VDO 볼륨에서 높은 물리적 공간 사용을 처리하는 가장 좋은 방법은 사용하지 않는 파일을 삭제하고 온라인 삭제 또는 **fstrim** 프로그램을 사용하여 사용되지 않은 파일에서 사용하는 블록을 삭제하는 것입니다. VDO 볼륨의 물리 공간은 기본 slab 크기가 2GB인 VDO 볼륨의 경우 16TB인 8192 slabs로, 최대 32GB 크기의 VDO 볼륨의 경우 256TB로 증가할 수 있습니다.

다음 모든 단계에서 *myvg* 및 *myvdo* 를 각각 볼륨 그룹 및 VDO 이름으로 교체합니다.

사전 요구 사항

1. VDO 소프트웨어를 설치합니다. 자세한 내용은 VDO 설치를 참조하십시오.
2. 사용 가능한 스토리지 용량이 있는 LVM 볼륨 그룹이 시스템에 있습니다.
3. **lvcreate --type ProfileBundle --name myvdo myvg -L logical-size-of-pool --virtualsize virtual-size-of-vdo** 명령을 사용하는 씬 프로비저닝 VDO 볼륨. 자세한 내용은 LVM-VDO 볼륨 생성을 참조하십시오.

절차

1. 씬 프로비저닝된 VDO 볼륨의 최적 논리 크기를 결정합니다.

```
# vdostats myvg-vpool0-vpool

Device                  1K-blocks Used     Available  Use% Space saving%
myvg-vpool0-vpool   104856576  29664088 75192488   28%   69%
```

   공간 절감 비율을 계산하려면 다음 수식을 사용하십시오.

```
Savings ratio = 1 / (1 - Space saving%)
```

   이 예에서, In this example,

   - 약 **3.22:1** 공간 절약 비율은 약 80GB의 데이터 세트에 있습니다.
   - 데이터 세트 크기를 비율으로 곱하면 동일한 공간 절약이 있는 더 많은 데이터가 VDO 볼륨에 기록되면 256GB의 잠재적인 논리 크기를 얻을 수 있습니다.
   - 이 수치를 200GB로 조정하면 동일한 공간 절약 비율을 고려하여 여유 물리적 공간이 안전한 논리 크기를 얻을 수 있습니다.

2. VDO 볼륨에서 사용 가능한 물리 공간을 모니터링합니다.

```
# vdostats myvg-vpool0-vpool
```

   이 명령은 주기적으로 실행되어 VDO 볼륨의 사용 가능한 물리 공간을 모니터링할 수 있습니다.

3. 선택 사항: 사용 가능한 **/usr/share/doc/vdo/examples/monitor/monitor_check_vdostats_physicalSpace.pl** 스크립트를 사용하여 VDO 볼륨의 물리 공간 사용에 대한 경고를 확인합니다.

```
# /usr/share/doc/vdo/examples/monitor/monitor_check_vdostats_physicalSpace.pl myvg-vpool0-vpool
```

4. VDO 볼륨을 생성할 때 **dmeventd** 모니터링 서비스는 VDO 볼륨의 물리 공간 사용량을 모니터링합니다. VDO 볼륨을 만들거나 시작할 때 기본적으로 활성화되어 있습니다.
   journalctl 명령을 사용하여 VDO 볼륨을 모니터링하는 동안 로그에 **dmeventd** 의 출력을 확인합니다.

```
lvm[8331]: Monitoring VDO pool myvg-vpool0-vpool.
...
```

```
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 84.63% full.
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 91.01% full.
lvm[8331]: WARNING: VDO pool myvg-vpool0-vpool is now 97.34% full.
```

5. 사용 가능한 물리 공간이 거의 없는 VDO 볼륨을 해결합니다. VDO 볼륨에 물리적 공간을 추가할 수 있지만 볼륨을 확장하기 전에 볼륨 공간이 가득 차면 I/O를 볼륨에 일시적으로 중지해야 할 수 있습니다.
볼륨에서 I/O를 일시적으로 중지하려면 다음 단계를 실행합니다. 여기서 VDO 볼륨 *myvdo* 에는 */users/homeDir* 경로에 마운트된 파일 시스템이 포함되어 있습니다.

   a. 파일 시스템을 정지합니다.

   ```
   # xfs_freeze -f /users/homeDir

   # vgextend myvg /dev/vdc2

   # lvextend -l new_size myvg/vpool0-name

   # xfs_freeze -u /users/homeDir
   ```

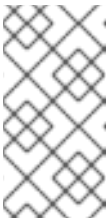
   b. 파일 시스템을 마운트 해제합니다.

   ```
   # umount /users/homeDir

   # vgextend myvg /dev/vdc2

   # lvextend -l new_size myvg/vpool0-name

   # mount -o discard /dev/myvg/myvdo /users/homeDir
   ```



참고

캐시된 데이터가 있는 파일 시스템을 마운트 해제하거나 해제하면 캐시된 데이터의 쓰기가 수행되므로 VDO 볼륨의 물리적 공간을 채울 수 있습니다. VDO 볼륨에서 사용 가능한 물리 공간에 대한 모니터링 임계값을 설정할 때 캐시된 파일 시스템 데이터의 최대 양을 고려하십시오.

6. 파일 시스템에서 더 이상 사용하지 않는 블록은 **fstrim** 유틸리티를 사용하여 정리할 수 있습니다. VDO 볼륨 상단에 마운트된 파일 시스템에 대해 **fstrim** 을 실행하면 해당 볼륨의 사용 가능한 물리 공간이 증가할 수 있습니다. **fstrim** 유틸리티를 사용하면 VDO 볼륨에 삭제가 전송되고 이전에 사용된 블록에 대한 참조를 제거하는 데 사용됩니다. 이러한 블록 중 하나가 단일 참조인 경우 물리적 공간을 사용할 수 있습니다.

   a. VDO 통계를 확인하여 현재 사용 가능한 공간이 무엇인지 확인합니다.

   ```
   # vdostats --human-readable myvg-vpool0-vpool

    Device              Size  Used  Available Use%  Space saving%
   myvg-vpool0-vpool  100.0G 95.0G 5.0G      95%   73%
   ```

   b. 사용되지 않는 블록 삭제:

   ```
   # fstrim /users/homeDir
   ```

c. VDO 볼륨의 사용 가능한 물리 공간을 확인합니다.

```
# vdostats --human-readable myvg-vpool0-vpool


 Device              Size    Used   Available Use%  Space saving%
myvg-vpool0-vpool  100.0G   30.0G  70.0G     30%    43%
```

이 예제에서는 파일 시스템에서 **fstrim** 을 실행한 후 VDO 볼륨에서 사용할 물리적 공간의 65G를 반환할 수 있었습니다.

참고

낮은 수준의 중복 제거 및 압축으로 볼륨을 삭제하면 중복 제거 및 압축 수준이 높은 볼륨을 삭제하는 것보다 물리적 공간을 회수할 수 있습니다. 높은 수준의 중복 제거 및 압축이 있는 볼륨은 이미 사용되지 않은 블록을 버리는 것보다 더 광범위한 정리를 필요로 할 수 있습니다.

4장. 기존 VDO 볼륨을 LVM으로 가져오기

VDO 관리자가 생성한 VDO 볼륨을 LVM으로 가져올 수 있습니다. 결과적으로 LVM 툴을 사용하여 볼륨을 논리 볼륨으로 관리할 수 있습니다.

참고

가져오기 작업은 취소할 수 없습니다. 기존 VDO 볼륨을 LVM으로 변환한 후 VDO 데이터 액세스는 LVM 명령을 통해서만 액세스할 수 있으며 **VDO** 관리자는 더 이상 볼륨을 제어하지 않습니다.

사전 요구 사항

- VDO 소프트웨어를 설치합니다. 자세한 내용은 VDO 설치를 참조하십시오.

절차

1. VDO 관리자가 생성한 기존 **VDO** 볼륨을 논리 볼륨으로 변환합니다. 다음 명령에서 이름이 볼륨 그룹 이름, *lv -name* 을 논리 볼륨 이름으로, */dev/sdg1* 을 VDO 장치로 바꿉니다.

```
# lvm_import_vdo --name vg-name/lv-name /dev/sdg1


Convert VDO device "/dev/sdg1" to VDO LV "vg-name/lv-name"? [y|N]: Yes
Stopping VDO vdo-name
Converting VDO vdo-name
    Opening /dev/disk/by-id/scsi-36d094660575ece002291bd67517f677a-part1 exclusively
    Loading the VDO superblock and volume geometry
    Checking the VDO state
    Converting the UDS index
    Converting the VDO
    Conversion completed for '/dev/disk/by-id/scsi-36d094660575ece002291bd67517f677a-
part1': VDO is now offset by 2097152 bytes
 Physical volume "/dev/sdg1" successfully created.
 Volume group "vg-name" successfully created
 WARNING: Logical volume vg-name/lv-name_vpool not zeroed.
 Logical volume "lv-name_vpool" created.
 WARNING: Converting logical volume vg-name/lv-name_vpool to VDO pool volume
WITHOUT formating.
 WARNING: Using invalid VDO pool data MAY DESTROY YOUR DATA!
 Logical volume "lv-name" created.
 Converted vg-name/lv-name_vpool to VDO pool volume and created virtual vg-name/lv-
name VDO volume.
```

2. 선택 사항: VDO LV에 파일 시스템을 만듭니다.
3. 선택 사항: LVM-VDO 볼륨을 마운트합니다. 자세한 내용은 LVM-VDO 볼륨 마운트를 참조하십시오.

검증

- LVM 장치를 나열하여 VDO 볼륨을 LVM으로 가져오는 것이 성공했는지 확인합니다.

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

```
  LV                VG      Attr       LSize  Pool         Origin Data%  Meta%  Move Log Cpy%Sync
Convert Devices
  lv-name           vg-name vwi-a-v--- 25.00g lv-name_vpool        0.00
lv-name_vpool(0)
  lv-name_vpool     vg-name dwi------- <1.82t                      0.31                         lv-
name_vpool_vdata(0)
  [lv-name_vpool_vdata] vg-name Dwi-ao---- <1.82t
/dev/sdg1(0)
```

추가 리소스

- **lvm_import_vdo(8)**, **lvmvdo(7)**, and **systemd.mount(5)** man pages

5장. LVM-VDO 볼륨의 TRIM 옵션

VDO 볼륨에 사용되지 않는 공간을 알리는 **discard** 옵션을 사용하여 파일 시스템을 마운트할 수 있습니다. 또 다른 옵션은 즉시 삭제를 위해 온디맨드 삭제 또는 **mount -o discard** 명령인 **fstrim** 애플리케이션을 사용하는 것입니다.

fstrim 애플리케이션을 사용하는 경우 관리자는 추가 프로세스를 예약하고 모니터링해야 하는 반면 **mount -o discard** 명령을 사용하면 가능한 경우 즉시 공간을 복구할 수 있습니다.

현재 **fstrim** 애플리케이션을 사용하여 **discard** 마운트 옵션이 아닌 사용하지 않는 블록을 폐기하는 것이 좋습니다. 이 옵션의 성능에 미치는 영향은 매우 심각할 수 있기 때문입니다. 이러한 이유로 **nodiscard** 가 기본값입니다.

5.1. VDO에서 삭제 마운트 옵션 활성화

이 절차에서는 VDO 볼륨에서 **삭제** 옵션을 활성화합니다.

사전 요구 사항

- LVM-VDO 볼륨이 시스템에 있습니다.

절차

- 볼륨에서 **삭제** 를 활성화합니다.

```
# mount -o discard /dev/vg-name/vdo-name mount-point
```

추가 리소스

- **XFS(5)**, **mount(8)** 및 **lvmvdo(7)** 도움말 페이지

5.2. 정기적인 TRIM 작업 설정

이 절차에서는 시스템에서 예약된 TRIM 작업을 활성화합니다.

사전 요구 사항

- LVM-VDO 볼륨이 시스템에 있습니다.

절차

- 타이머를 활성화하고 시작합니다.

```
# systemctl enable --now fstrim.timer
```

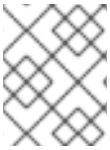
검증

- 타이머가 활성화되었는지 확인합니다.

```
# systemctl list-timers fstrim.timer
```

예 5.1. 확인 절차의 출력 가능

```
# systemctl list-timers fstrim.timer
NEXT                        LEFT        LAST  PASSED  UNIT         ACTIVATES
Mon 2021-05-10 00:00:00 EDT  5 days left  n/a   n/a     fstrim.timer fstrim.service
```



참고

fstrim.timer 는 마운트된 모든 파일 시스템에서 실행되므로 VDO 볼륨에 대한 참조가 표시되지 않습니다.

추가 리소스

- **fstrim(8)** 도움말 페이지

6장. VDO 성능 최적화

VDO 커널 드라이버는 여러 스레드를 사용하여 작업을 가속화합니다. 하나의 스레드가 I/O 요청을 위해 모든 작업을 수행하는 대신 작업을 다른 스레드에 할당된 작은 부분으로 분할합니다. 이러한 스레드는 요청을 처리할 때 서로 통신합니다. 이렇게 하면 하나의 스레드가 일정한 잠금 및 잠금 해제 없이 공유 데이터를 처리할 수 있습니다.

하나의 스레드가 작업을 완료하면 VDO에는 이미 다른 작업이 준비됩니다. 이렇게 하면 스레드가 바빠지고 작업을 전환하는 데 소요되는 시간이 줄어듭니다. VDO는 또한 큐에 I/O 작업 추가 또는 중복 제거 인덱스에 메시지를 처리하는 등 느린 작업에 별도의 스레드를 사용합니다.

6.1. VDO 스레드 유형

VDO는 다양한 스레드 유형을 사용하여 특정 작업을 처리합니다.

논리 영역 스레드(kvdo:logQ)

기본 스토리지 시스템에서 VDO 장치의 사용자에게 제공되는LBN(Logical block numbers)과 물리적 블록 번호(PBN) 간의 매핑을 유지 관리합니다. 또한 동일한 블록에 대한 동시 쓰기를 방지합니다. 논리 스레드는 읽기 및 쓰기 작업 둘 다에서 활성화됩니다. 일반적으로 처리는 균등하게 분배되지만 특정 액세스 패턴은 때때로 하나의 스레드에서 작업을 집중시킬 수 있습니다. 예를 들어 특정 블록 맵 페이지에서 LBN에 자주 액세스하는 경우 하나의 논리 스레드가 이러한 모든 작업을 처리할 수 있습니다.

물리 영역 스레드(kvdo:physQ)

쓰기 작업 중 데이터 블록 할당 및 참조 수를 처리합니다.

I/O 제출 스레드(kvdo:bioQ)

VDO에서 스토리지 시스템으로 블록 I/O(**bio**) 작업 전송을 처리합니다. 다른 VDO 스레드의 I/O 요청을 처리하고 기본 장치 드라이버에 전달합니다. 이러한 스레드는 장치 관련 데이터 구조와 상호 작용하고, 장치 드라이버 커널 스레드에 대한 요청을 생성하며 전체 장치 요청 큐로 인해 I/O 요청이 차단될 때 지연을 방지합니다.

CPU 처리 스레드(kvdo:cpuQ)

다른 스레드 유형에 대한 데이터 구조에 대한 배타적 액세스 권한을 차단하거나 필요로 하지 않는 CPU 집약적인 작업을 처리합니다. 이러한 작업에는 해시 값 계산 및 데이터 블록 압축이 포함됩니다.

I/O 승인 스레드(kvdo:ackQ)

직접 I/O를 수행하는 커널 페이지 캐시 또는 애플리케이션 스레드와 같은 상위 수준 구성 요소에 대한 I/O 요청 완료 신호를 보냅니다. 해당 CPU 사용량과 메모리 경합에 미치는 영향은 커널 수준 코드의 영향을 받습니다.

해시 영역 스레드 (kvdo:hashQ)

잠재적인 중복 제거 작업을 처리하기 위해 일치하는 해시로 I/O 요청을 조정합니다. 중복 제거 요청을 생성 및 관리하더라도 상당한 계산을 수행하지 않습니다. 일반적으로 단일 해시 영역 스레드로 충분합니다.

중복 제거 스레드(kvdo:dedupeQ)

I/O 요청을 처리하고 중복 제거 인덱스와 통신합니다. 이 작업은 차단을 방지하기 위해 별도의 스레드에서 수행됩니다. 또한 인덱스가 빠르게 응답하지 않는 경우 중복 제거를 생략하는 시간 초과 메커니즘이 있습니다. VDO 장치당 중복 제거 스레드가 하나만 있습니다.

저널 스레드(kvdo:journalQ)

복구 저널을 업데이트하고 쓰기를 위해 저널 블록을 예약합니다. 이 작업은 여러 스레드로 나눌 수 없습니다. VDO 장치당 저널 스레드가 하나만 있습니다.

Packer 스레드 (kvdo:packerQ)

압축이 활성화된 경우 쓰기 작업 중에 작동합니다. CPU 스레드에서 압축 데이터 블록을 수집하여 낭비되는 공간을 줄입니다. VDO 장치당 하나의 팩터 스레드만 있습니다.

6.2. 성능 병목 현상 확인

VDO 성능에서 병목 현상을 식별하는 것은 시스템 효율성을 최적화하는 데 중요합니다. 수행할 수 있는 기본 단계 중 하나는 병목 현상이 CPU, 메모리 또는 백업 스토리지 속도에 있는지 확인하는 것입니다. 가장 느린 구성 요소를 정확하게 파악한 후 성능 향상을 위한 전략을 개발할 수 있습니다.

낮은 성능의 근본 원인이 하드웨어 문제가 되지 않도록 하려면 스토리지 스택에서 VDO를 사용하여 테스트를 실행합니다.

VDO의 **journalQ** 스레드는 특히 VDO 볼륨에서 쓰기 작업을 처리할 때 자연적인 병목 현상입니다. 다른 스레드 유형에 **journalQ** 스레드보다 사용률이 높으면 해당 유형의 스레드를 추가하여 이 문제를 해결할 수 있습니다.

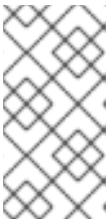
6.2.1. top을 사용하여 VDO 성능 분석

top 유틸리티를 사용하여 VDO 스레드의 성능을 검사할 수 있습니다.

절차

1. 개별 스레드를 표시합니다.

```
$ top -H
```



참고

top 과 같은 툴은 캐시 또는 메모리 지연으로 인해 생산적인 CPU 사이클과 사이클을 구분할 수 없습니다. 이러한 툴은 캐시 경합 및 메모리 액세스를 실제 작업으로 해석합니다. 노드 간에 스레드를 이동하는 것은 CPU 사용률 감소와 같이 표시될 수 있으며 초당 작업을 늘릴 수 있습니다.

2. **f** 키를 눌러 필드 관리자를 표시합니다.
3. **(Cryostat)** 키를 사용하여 **P = Last Used Cpu (SMP)** 필드로 이동합니다.
4. 스페이스바를 눌러 **P = Last Used Cpu (SMP)** 필드를 선택합니다.
5. **q** 키를 눌러 필드 관리자를 종료합니다. 이제 **top** 유틸리티에서 개별 코어의 CPU 부하를 표시하고 각 프로세스 또는 스레드마다 최근에 사용된 CPU를 나타냅니다. **1** 를 눌러 CPU별 통계로 전환할 수 있습니다.

추가 리소스

- **top(1)** 도움말 페이지
- 상위 결과 해석

6.2.2. 상위 결과 해석

VDO 스레드의 성능을 분석하는 동안 다음 표를 사용하여 **top** 유틸리티의 결과를 해석합니다.

표 6.1. 상위 결과 해석

값	설명	제안
스레드 또는 CPU 사용량이 70%를 초과했습니다.	스레드 또는 CPU가 과부하됩니다. 사용량이 높은 경우 실제 작업이 없는 CPU에 예약된 VDO 스레드가 발생할 수 있습니다. 이는 과도한 하드웨어 중단, 메모리 충돌 또는 리소스 경쟁으로 인해 발생할 수 있습니다.	이 코어를 실행하는 유형의 스레드 수를 늘립니다.
낮은 **%id** 및 **%wa** 값	핵심은 적극적으로 작업을 처리하는 것입니다.	작업이 필요하지 않습니다.
낮은 **%hi** 값	핵심은 표준 처리 작업을 수행하는 것입니다.	성능을 개선하기 위해 코어를 더 추가합니다. NUMA 충돌을 방지합니다.
• **%hi** 의 높은 값[a] • 코어에 하나의 스레드만 할당됩니다. • **%ID** 가 0 • **%WA** 값이 0	코어는 과도하게 배출됩니다.	커널 스레드 및 장치 인터럽트 처리를 다른 코어에 다시 할당합니다.
• **kvdo:bioQ** 스레드는 **D** 상태에서 자주 발생합니다.	VDO는 I/O 요청으로 스토리지 시스템을 지속적으로 사용하고 있습니다.[b]	CPU 사용률이 매우 낮은 경우 I/O 제출 스레드 수를 줄입니다.
kvdo:bioQ 스레드는 **S** 상태에서 자주 발생합니다.	VDO에는 필요한 것보다 더 많은 **kvdo:bioQ** 스레드가 있습니다.	**kvdo:bioQ** 스레드 수를 줄입니다.
I/O 요청당 CPU 사용률이 높습니다.	I/O 요청당 CPU 사용률이 더 많은 스레드로 증가합니다.	CPU, 메모리 또는 잠금 경합을 확인합니다.

[a] 몇 % 이상

[b] 이는 스토리지 시스템이 여러 요청을 처리할 수 있거나 요청 처리가 효율적인 경우 유용합니다.

6.2.3. perf로 VDO 성능 분석

perf 유틸리티를 사용하여 VDO의 CPU 성능을 확인할 수 있습니다.

사전 요구 사항

- **perf** 패키지가 설치되어 있습니다.

절차

1. 성능 프로필을 표시합니다.

```
# perf top
```

2. **perf** 결과를 해석하여 CPU 성능을 분석합니다.

표 6.2. perf 결과 해석

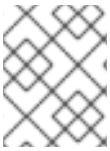
값	설명	제안
kvdo:bioQ 스레드는 과도한 사이클 수집 회전 잠금	VDO 아래의 장치 드라이버에서 너무 많은 경합이 발생할 수 있습니다.	**kvdo:bioQ** 스레드 수 감소
높은 CPU 사용량	NUMA 노드 간 연결. 프로세서에서 지원하는 경우 **stalled-cycles-backend**,**cache-misses**, **node-load-misses** 와 같은 카운터를 확인합니다. 누락 속도가 높으면 가능한 경합을 나타내는 다른 툴에서 CPU 사용량이 높은 CPU 사용량을 다시 조합한 stalls가 발생할 수 있습니다.	VDO 커널 스레드에 대한 CPU 선호도 또는 인터럽트 처리기의 IRQ 선호도를 구현하여 처리를 단일 노드로 제한합니다.

추가 리소스

- **perf-top(1)** 도움말 페이지

6.2.4. sar를 사용하여 VDO 성능 분석

sar 유틸리티를 사용하여 VDO 성능에 대한 주기적인 보고서를 생성할 수 있습니다.



참고

모든 블록 장치 드라이버가 **sar** 유틸리티에 필요한 데이터를 제공할 수 있는 것은 아닙니다. 예를 들어 MD RAID와 같은 장치는 **%util** 값을 보고하지 않습니다.

사전 요구 사항

- Cryo **stat 유틸리티** 를 설치합니다.

```
# yum install sysstat
```

절차

1. 1초 간격으로 디스크 I/O 통계를 표시합니다.

```
$ sar -d 1
```

2. **sar** 결과를 해석하여 VDO 성능을 분석합니다.

표 6.3. sar 결과 해석

값	설명	제안
• 기본 스토리지 장치의 **%util** 값은 100% 미만입니다. • VDO는 100 %로 비어 있습니다. • **bioQ** 스레드는 많은 CPU 시간을 사용하고 있습니다.	VDO에는 빠른 장치를 위한 **bioQ** 스레드가 너무 적습니다.	**bioQ** 스레드를 더 추가합니다. 스핀 잠금 경합으로 인해 **bioQ** 스레드를 추가할 때 특정 스토리지 드라이버가 느려질 수 있습니다.

추가 리소스

- **SAR(1)** 도움말 페이지

6.3. VDO 스레드 재배포

VDO는 요청을 처리할 때 다양한 작업에 다양한 스레드 풀을 사용합니다. 최적의 성능은 사용 가능한 스토리지, CPU 리소스 및 워크로드 유형에 따라 다른 풀의 올바른 스레드 수 설정에 따라 달라집니다. VDO 작업을 여러 스레드에 분배하여 VDO 성능을 향상시킬 수 있습니다.

VDO는 병렬 처리를 통한 성능을 극대화하는 것을 목표로 합니다. 사용 가능한 CPU 리소스 및 병목 현상과 같은 요인에 따라 병목 현상이 발생한 작업에 더 많은 스레드를 할당하여 개선할 수 있습니다. 스레드 사용률이 높은 경우 (70-80 % 이상) 지연이 발생할 수 있습니다. 따라서 이러한 경우 스레드 수를 늘리는 것이 도움이 될 수 있습니다. 그러나 과도한 스레드는 성능을 방해하고 추가 비용이 발생할 수 있습니다.

최적의 성능을 위해 다음 작업을 수행합니다.

- 다양한 예상 워크로드로 VDO를 테스트하여 성능을 평가하고 최적화합니다.
- 50% 이상의 사용률을 가진 풀의 스레드 수를 늘립니다.
- 개별 스레드 사용률이 낮은 경우에도 전체 사용률이 50%를 초과하면 VDO에서 사용할 수 있는 코어 수를 늘립니다.

6.3.1. NUMA 노드 간에 VDO 스레드 그룹화

NUMA 노드 간에 메모리에 액세스하는 것은 로컬 메모리 액세스보다 느립니다. 코어가 노드 내에서 마지막 수준 캐시를 공유하는 Intel 프로세서에서 단일 노드 내에서 공유되는 것보다 노드 간에 데이터를 공유할 때 캐시 문제가 더 중요합니다. 많은 VDO 커널 스레드는 배타적 데이터 구조를 관리하지만 종종 I/O 요청에 대한 메시지를 교환합니다. VDO 스레드가 여러 노드에 분산되거나 스케줄러에서 노드 간에 스레드를 다시 할당하면 경합이 발생할 수 있으며, 이는 동일한 리소스에 대해 경쟁하는 여러 노드가 될 수 있습니다.

동일한 NUMA 노드에서 특정 스레드를 그룹화하여 VDO 성능을 향상시킬 수 있습니다.

하나의 NUMA 노드에서 서로 그룹 관련 스레드

- I/O 승인(**ackQ**) 스레드

- 상위 수준의 I/O 제출 스레드:
  - 직접 I/O 처리 사용자 모드 스레드
  - 커널 페이지 캐시 플러시 스레드

장치 액세스 최적화

- 장치 액세스 타이밍이 NUMA 노드마다 다른 경우 스토리지 장치 컨트롤러에 가장 가까운 노드에서 **bioQ** 스레드를 실행합니다.

경합 최소화

- **logQ** 또는 **physQ** 스레드와 동일한 노드에서 I/O 제출 및 스토리지 장치 인터럽트 처리를 실행합니다.
- 동일한 노드에서 다른 VDO 관련 작업을 실행합니다.
- 하나의 노드에서 모든 VDO 작업을 처리할 수 없는 경우 스레드를 다른 노드로 이동할 때 메모리 경합을 고려하십시오. 예를 들어 처리 및 **bioQ** 스레드를 중단하는 장치를 다른 노드로 이동합니다.

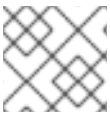
6.3.2. CPU 선호도 구성

VDO 스레드의 CPU 선호도를 조정하는 경우 특정 스토리지 장치 드라이버에서 VDO 성능을 향상시킬 수 있습니다.

스토리지 장치 드라이버의 인터럽트(IRQ) 처리기가 상당한 작업을 수행하고 드라이버가 스레드 IRQ 처리기를 사용하지 않는 경우, 시스템 스케줄러에서 VDO 성능을 최적화하는 기능을 제한할 수 있습니다.

최적의 성능을 위해 다음 작업을 수행합니다.

- 코어가 과부하된 경우 특정 코어를 IRQ 처리로 전용하고 VDO 스레드 선호도를 조정합니다. **%hi** 값이 다른 코어보다 몇 퍼센트를 초과하면 코어가 과부하됩니다.
- 사용 중인 IRQ 코어에서 **kvdo:journalQ** 스레드와 같은 싱글톤 VDO 스레드를 실행하지 않도록 합니다.
- 개별 CPU 사용이 높은 경우에만 IRQs에서 다른 스레드 유형을 사용 중인 코어를 유지합니다.



참고

시스템 재부팅 시 구성이 유지되지 않습니다.

절차

- CPU 선호도를 설정합니다.

```
# taskset -c <cpu-numbers> -p <process-id>
```

& **lt;cpu-numbers** >를 프로세스를 할당하려는 쉼표로 구분된 CPU 번호 목록으로 바꿉니다. & **lt;process-id** >를 CPU 선호도를 설정하려는 실행 중인 프로세스의 ID로 바꿉니다.

예 6.1. CPU 코어 1 및 2에서kvdo 프로세스에 대한 CPU 유사성 설정

```
# for pid in `ps -eo pid,comm | grep kvdo | awk '{ print $1 }'`
do
    taskset -c "1,2" -p $pid
done
```

검증

- 선호도 세트를 표시합니다.

```
# taskset -p <cpu-numbers> -p <process-id>
```

& **lt;cpu-numbers** >를 프로세스를 할당하려는 쉼표로 구분된 CPU 번호 목록으로 바꿉니다. & **lt;process-id** >를 CPU 선호도를 설정하려는 실행 중인 프로세스의 ID로 바꿉니다.

추가 리소스

- **taskset(1)** 도움말 페이지

6.4. 성능 향상을 위해 블록 맵 캐시 크기 증가

VDO 볼륨의 캐시 크기를 늘려 읽기 및 쓰기 성능을 향상시킬 수 있습니다.

읽기 및 쓰기 대기 시간이 연장되었거나 애플리케이션 요구 사항에 부합하지 않는 스토리지에서 읽은 상당한 양의 데이터가 있는 경우 캐시 크기를 조정해야 할 수 있습니다.

주의

블록 맵 캐시를 늘리면 캐시는 지정한 메모리 양과 추가 15%의 메모리를 사용합니다. 더 큰 캐시 크기는 더 많은 RAM을 사용하고 전체 시스템 안정성에 영향을 미칩니다.

다음 예제에서는 시스템의 캐시 크기를 128Mb에서 640Mb로 변경하는 방법을 보여줍니다.

절차

1. VDO 볼륨의 현재 캐시 크기를 확인합니다.

```
# lvs -o vdo_block_map_cache_size
  VDOBlockMapCacheSize
               128.00m
               128.00m
```

2. VDO 볼륨을 비활성화합니다.

```
# lvchange -an vg_name/vdo_volume
```

3. VDO 설정을 변경합니다.

```
# lvchange --vdosettings "block_map_cache_size_mb=640" vg_name/vdo_volume
```

640 을 새 캐시 크기(MB)로 바꿉니다.

참고

캐시 크기는 128MB에서 16TB 범위 내에서 4096의 배수, 논리 스레드당 최소 16MB여야 합니다. 다음에 VDO 장치가 시작될 때 변경 사항이 적용됩니다. 이미 실행중인 장치는 영향을 받지 않습니다.

4. VDO 볼륨을 활성화합니다.

```
# lvchange -ay vg_name/vdo_volume
```

검증

- 현재 VDO 볼륨 구성을 확인합니다.

```
# lvs -o vdo_block_map_cache_size vg_name/vdo_volume
  VDOBlockMapCacheSize
               640.00m
```

추가 리소스

- **lvchange(8)** 도움말 페이지

6.5. 삭제 작업 가속화

VDO는 시스템의 모든 VDO 장치에 대해 허용되는 최대 DISCARD(TRIM) 섹터 크기를 설정합니다. 기본 크기는 8개의 섹터로, 하나의 4-KiB 블록에 해당합니다. DISCARD 크기를 늘리면 삭제 작업의 속도를 크게 향상시킬 수 있습니다. 그러나 삭제 성능을 개선하고 다른 쓰기 작업의 속도를 유지하는 사이에 절충이 있습니다.

최적의 DISCARD 크기는 스토리지 스택에 따라 다릅니다. 매우 크고 매우 작은 DISCARD 섹터 모두 잠재적으로 성능을 저하시킬 수 있습니다. 다양한 값으로 실험하여 우수한 결과를 제공하는 것을 발견하십시오.

로컬 파일 시스템을 저장하는 VDO 볼륨의 경우 기본 설정인 8개 섹터의 DISCARD 크기를 사용하는 것이 가장 적합합니다. SCSI 대상 역할을 하는 VDO 볼륨의 경우 2048 섹터(1MB 삭제에 해당)와 같이 보통 큰 DISCARD 크기가 가장 적합합니다. 최대 DISCARD 크기가 5MB 삭제로 변환되는 10240 섹터를 초과하지 않는 것이 좋습니다. 크기를 선택할 때 VDO는 8 섹터보다 작으면 삭제 작업을 효과적으로 처리하지 못할 수 있으므로 8의 배수인지 확인합니다.

절차

1. DISCARD 섹터의 새 최대 크기를 설정합니다.

```
# echo <number-of-sectors> > /sys/kvdo/max_discard_sectors
```

& **lt;number-of-sectors&gt**;를 섹터 수로 바꿉니다. 이 설정은 재부팅할 때까지 유지됩니다.

2. 선택 사항: 재부팅 시 DISCARD 섹터를 영구적으로 변경하려면 사용자 지정 **systemd** 서비스를 생성합니다.

a. 다음 콘텐츠를 사용하여 새 **/etc/systemd/system/max_discard_sectors.service** 파일을 만듭니다.

```
[Unit]
Description=Set maximum DISCARD sector
[Service]
ExecStart=/usr/bin/echo <number-of-sectors> > /sys/kvdo/max_discard_sectors

[Install]
WantedBy=multi-user.target
```

& **lt;number-of-sectors>**를 섹터 수로 바꿉니다.

b. 파일을 저장하고 종료합니다.

c. 서비스 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

d. 새 서비스를 활성화합니다.

```
# systemctl enable max_discard_sectors.service
```

검증

- 선택 사항: scaling governor 변경을 영구적으로 수행한 경우 **max_discard_sectors.service** 가 활성화되어 있는지 확인합니다.

```
# systemctl is-enabled max_discard_sectors.service
```

6.6. CPU 빈도 스케일링 최적화

기본적으로 RHEL은 CPU 빈도 스케일링을 사용하여 CPU 빈도 스케일링을 사용하여 CPU의 부하가 많은 경우 전원을 줄이고 heat를 줄입니다. 전력 절감에 비해 성능을 우선시하기 위해 CPU를 최대 클럭 속도로 작동하도록 구성할 수 있습니다. 이렇게 하면 CPU에서 최대 효율성으로 데이터 중복 제거 및 압축 프로세스를 처리할 수 있습니다. CPU를 가장 높은 빈도로 실행하면 리소스 집약적인 작업을 더 빠르게 실행하여 데이터 감소 및 스토리지 최적화 측면에서 VDO의 전반적인 성능을 향상시킬 수 있습니다.

주의

높은 성능을 위해 CPU 빈도 스케일링을 조정하면 전력 소비와 heat 생성이 증가할 수 있습니다. 부적절하게 잘린 시스템에서는 과열이 발생할 수 있으며 열 제한으로 인해 성능 향상을 제한할 수 있습니다.

절차

1. 사용 가능한 CPU 관리자를 표시합니다.

```
$ cpupower frequency-info -g
```

2. 성능 우선 순위를 지정하도록 스케일링 관리자를 변경합니다.

```
# cpupower frequency-set -g performance
```

이 설정은 재부팅할 때까지 유지됩니다.

3. 선택 사항: 재부팅 시 스케일링 governor를 영구적으로 변경하려면 사용자 지정 **systemd** 서비스를 생성합니다.

a. 다음 콘텐츠를 사용하여 새 **/etc/systemd/system/cpufreq.service** 파일을 만듭니다.

```
[Unit]
Description=Set CPU scaling governor to performance

[Service]
ExecStart=/usr/bin/cpupower frequency-set -g performance

[Install]
WantedBy=multi-user.target
```

b. 파일을 저장하고 종료합니다.

c. 서비스 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

d. 새 서비스를 활성화합니다.

```
# systemctl enable cpufreq.service
```

검증

- 현재 사용되는 CPU 빈도 정책을 표시합니다.

```
$ cpupower frequency-info -p
```

- 선택 사항: scaling governor 변경을 영구적으로 수행한 경우 **cpufreq.service** 가 활성화되어 있는지 확인합니다.

```
# systemctl is-enabled cpufreq.service
```