



Red Hat Enterprise Linux 8

Deduplicating and compressing storage

Using VDO to optimize storage capacity in RHEL 8

Red Hat Enterprise Linux 8 Deduplicating and compressing storage

Using VDO to optimize storage capacity in RHEL 8

Legal Notice

Copyright © 2019 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.

Abstract

This documentation collection provides instructions on how to use the Virtual Data Optimizer (VDO) to manage deduplicated and compressed storage pools in Red Hat Enterprise Linux 8.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	6
CHAPTER 1. DEPLOYING VDO	7
1.1. INTRODUCTION TO VDO	7
1.2. VDO DEPLOYMENT SCENARIOS	7
KVM	7
File systems	8
iSCSI target	8
LVM	8
Encryption	9
1.3. VDO REQUIREMENTS	9
1.3.1. Placement of VDO in the storage stack	9
Additional resources	10
1.3.2. VDO memory requirements	10
The VDO module	10
The Universal Deduplication Service (UDS) index	11
Additional resources	11
1.3.3. VDO storage space requirements	11
Additional resources	11
1.3.4. Examples of VDO requirements by physical volume size	11
Primary storage deployment	12
Backup storage deployment	12
1.4. INSTALLING VDO	12
Procedure	12
1.5. CREATING A VDO VOLUME	13
Prerequisites	13
Procedure	13
Next steps	14
Additional resources	14
1.6. MOUNTING A VDO VOLUME	14
Prerequisites	14
Procedure	14
1.7. ENABLING PERIODIC BLOCK DISCARD	15
Procedure	15
1.8. MONITORING VDO	15
Prerequisites	15
Procedure	15
Additional resources	15
CHAPTER 2. MAINTAINING VDO	16
2.1. PREREQUISITES	16
2.2. MANAGING FREE SPACE ON VDO VOLUMES	16
2.2.1. Thin provisioning in VDO	16
2.2.2. Monitoring VDO	17
Prerequisites	17
Procedure	17
Additional resources	17
2.2.3. Reclaiming space for VDO on file systems	17
Procedure	17
2.2.4. Reclaiming space for VDO without a file system	17
Procedure	17

Additional resources	18
2.2.5. Reclaiming space for VDO on Fibre Channel or Ethernet network	18
Procedure	18
2.3. STARTING OR STOPPING VDO VOLUMES	18
2.3.1. Started and activated VDO volumes	18
2.3.2. Starting a VDO volume	19
Procedure	19
Additional resources	19
2.3.3. Stopping a VDO volume	19
Procedure	19
Additional resources	19
2.3.4. Related information	19
2.4. AUTOMATICALLY STARTING VDO VOLUMES AT SYSTEM BOOT	20
2.4.1. Started and activated VDO volumes	20
2.4.2. Activating a VDO volume	20
Procedure	20
Additional resources	20
2.4.3. Deactivating a VDO volume	21
Procedure	21
Additional resources	21
2.5. SELECTING A VDO WRITE MODE	21
2.5.1. VDO write modes	21
2.5.2. The internal processing of VDO write modes	22
2.5.3. Checking the write mode on a VDO volume	22
Procedure	22
2.5.4. Checking for a volatile cache	23
Procedure	23
2.5.5. Setting a VDO write mode	23
Prerequisites	24
Procedure	24
2.6. RECOVERING A VDO VOLUME AFTER AN UNCLEAN SHUTDOWN	24
2.6.1. VDO write modes	24
2.6.2. VDO volume recovery	25
Automatic and manual recovery	25
Additional resources	25
2.6.3. VDO operating modes	25
2.6.4. Recovering a VDO volume online	26
Procedure	26
2.6.5. Forcing an offline rebuild of a VDO volume metadata	27
Prerequisites	27
Procedure	27
2.6.6. Removing an unsuccessfully created VDO volume	27
Procedure	28
2.7. OPTIMIZING THE UDS INDEX	28
2.7.1. The UDS index	28
2.7.2. Recommended UDS index configuration	29
Additional resources	29
2.8. ENABLING OR DISABLING DEDUPLICATION IN VDO	29
2.8.1. Deduplication in VDO	30
2.8.2. Enabling deduplication on a VDO volume	30
Procedure	30
2.8.3. Disabling deduplication on a VDO volume	30
Procedure	30

2.9. ENABLING OR DISABLING COMPRESSION IN VDO	30
2.9.1. Compression in VDO	30
2.9.2. Enabling compression on a VDO volume	31
Procedure	31
2.9.3. Disabling compression on a VDO volume	31
Procedure	31
2.10. INCREASING THE SIZE OF A VDO VOLUME	31
2.10.1. Thin provisioning in VDO	32
2.10.2. Increasing the logical size of a VDO volume	32
Procedure	33
2.10.3. Increasing the physical size of a VDO volume	33
Prerequisites	33
Procedure	33
2.11. REMOVING VDO VOLUMES	33
2.11.1. Removing a working VDO volume	33
Procedure	33
2.11.2. Removing an unsuccessfully created VDO volume	33
Procedure	34
2.12. RELATED INFORMATION	34
CHAPTER 3. DISCARDING UNUSED BLOCKS	35
3.1. BLOCK DISCARD OPERATIONS	35
Requirements	35
3.2. TYPES OF BLOCK DISCARD OPERATIONS	35
Recommendations	35
3.3. PERFORMING BATCH BLOCK DISCARD	35
Prerequisites	35
Procedure	36
Additional resources	36
3.4. ENABLING ONLINE BLOCK DISCARD	36
Procedure	36
Additional resources	36
3.5. ENABLING PERIODIC BLOCK DISCARD	36
Procedure	37
CHAPTER 4. OVERVIEW OF PERSISTENT NAMING ATTRIBUTES	38
4.1. DISADVANTAGES OF NON-PERSISTENT NAMING ATTRIBUTES	38
4.2. FILE SYSTEM AND DEVICE IDENTIFIERS	38
File system identifiers	39
Device identifiers	39
Recommendations	39
4.3. DEVICE NAMES MANAGED BY THE UDEV MECHANISM IN /DEV/DISK/	39
4.3.1. File system identifiers	39
The UUID attribute in /dev/disk/by-uuid/	39
The Label attribute in /dev/disk/by-label/	40
4.3.2. Device identifiers	40
The WWID attribute in /dev/disk/by-id/	40
The Partition UUID attribute in /dev/disk/by-partuuid	41
The Path attribute in /dev/disk/by-path/	41
4.4. THE WORLD WIDE IDENTIFIER WITH DM MULTIPATH	41
4.5. LIMITATIONS OF THE UDEV DEVICE NAMING CONVENTION	42
4.6. LISTING PERSISTENT NAMING ATTRIBUTES	42
Procedure	43

4.7. MODIFYING PERSISTENT NAMING ATTRIBUTES	44
Prerequisites	44
Procedure	44
CHAPTER 5. USING THE WEB CONSOLE FOR MANAGING VIRTUAL DATA OPTIMIZER VOLUMES	45
5.1. PREREQUISITES	45
5.2. VDO VOLUMES IN THE WEB CONSOLE	45
Additional resources	46
5.3. CREATING VDO VOLUMES IN THE WEB CONSOLE	46
Prerequisites	46
Procedure	46
5.4. FORMATTING VDO VOLUMES IN THE WEB CONSOLE	47
Prerequisites	48
Procedure	48
5.5. EXTENDING VDO VOLUMES IN THE WEB CONSOLE	49
Prerequisites	49
Procedure	50

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
 1. Go to the [Bugzilla](#) website.
 2. As the Component, use **Documentation**.
 3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
 4. Click **Submit Bug**.

CHAPTER 1. DEPLOYING VDO

As a system administrator, you can use VDO to create deduplicated and compressed storage pools.

1.1. INTRODUCTION TO VDO

Virtual Data Optimizer (VDO) provides inline data reduction for Linux in the form of deduplication, compression, and thin provisioning. When you set up a VDO volume, you specify a block device on which to construct your VDO volume and the amount of logical storage you plan to present.

- When hosting active VMs or containers, Red Hat recommends provisioning storage at a 10:1 logical to physical ratio: that is, if you are utilizing 1 TB of physical storage, you would present it as 10 TB of logical storage.
- For object storage, such as the type provided by Ceph, Red Hat recommends using a 3:1 logical to physical ratio: that is, 1 TB of physical storage would present as 3 TB logical storage.

In either case, you can simply put a file system on top of the logical device presented by VDO and then use it directly or as part of a distributed cloud storage architecture.

Because VDO is thinly provisioned, the file system and applications only see the logical space in use and are not aware of the actual physical space available. Scripting should be used to monitor the actual available space and generate an alert if use exceeds a threshold: for example, when the VDO volume is 80% full. See [Section 2.2, “Managing free space on VDO volumes”](#) for details.

1.2. VDO DEPLOYMENT SCENARIOS

You can deploy VDO in a variety of ways to provide deduplicated storage for:

- both block and file access
- both local and remote storage

Because VDO exposes its deduplicated storage as a standard Linux block device, you can use it with standard file systems, iSCSI and FC target drivers, or as unified storage.

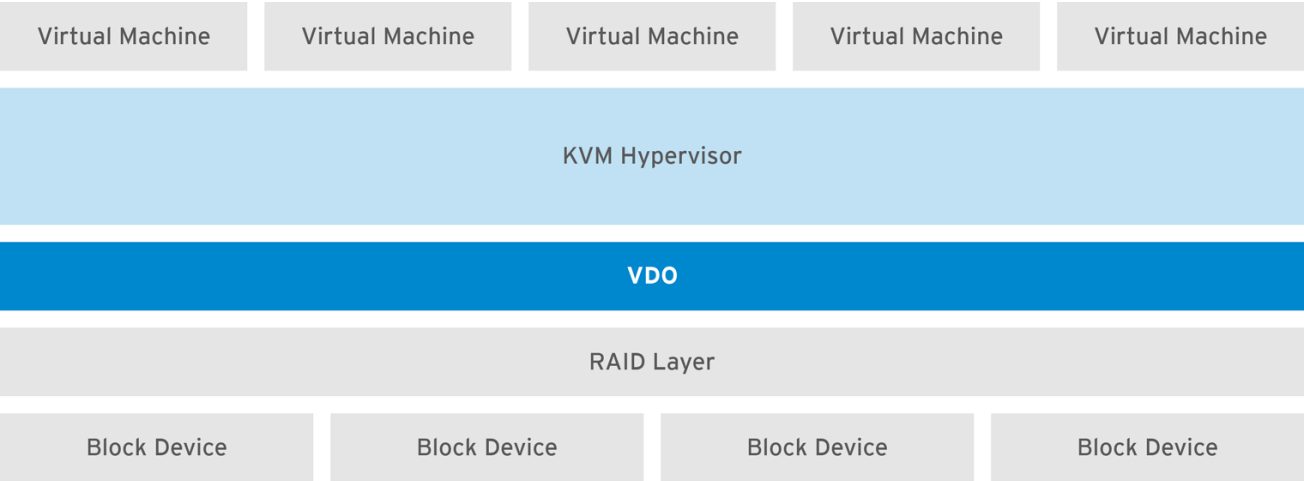


NOTE

VDO deployment with Ceph Storage is currently not supported.

KVM

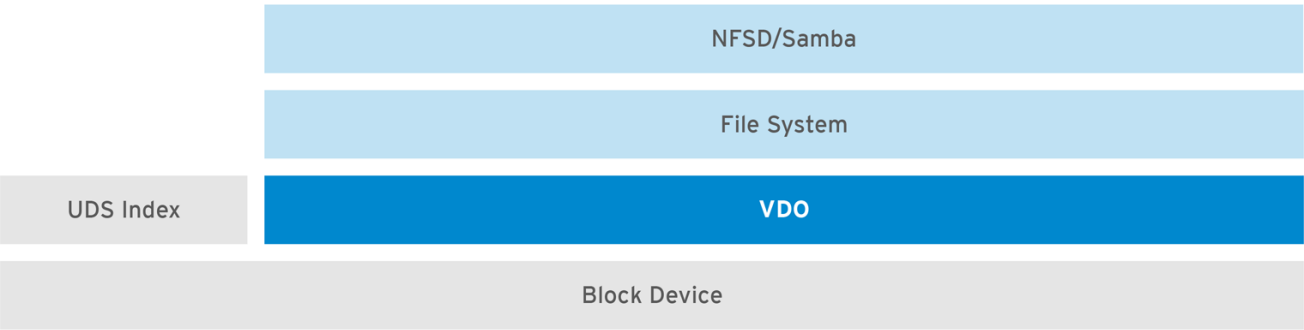
You can deploy VDO on a KVM server configured with Direct Attached Storage.



RHEL_462492_1117

File systems

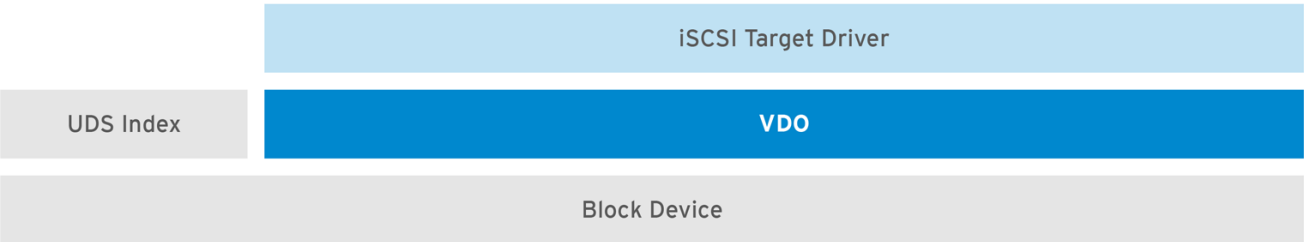
You can create file systems on top of VDO and expose them to NFS or CIFS users with the NFS server or Samba.



RHEL_466924_0218

iSCSI target

You can export the entirety of the VDO storage target as an iSCSI target to remote iSCSI initiators.

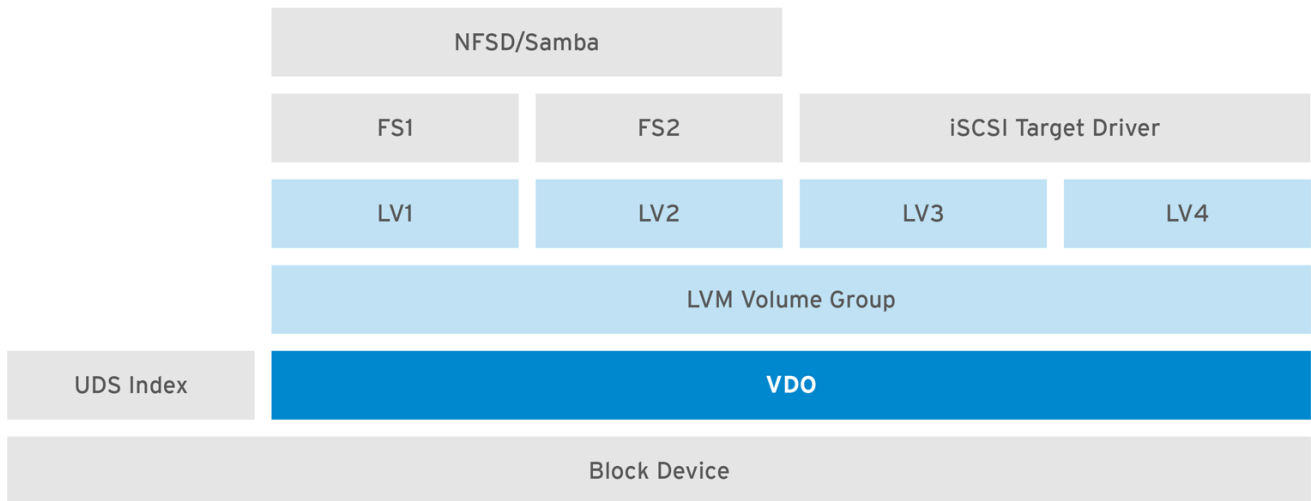


RHEL_466924_0218

LVM

On more feature-rich systems, you can use LVM to provide multiple logical unit numbers (LUNs) that are all backed by the same deduplicated storage pool.

In the following diagram, the VDO target is registered as a physical volume so that it can be managed by LVM. Multiple logical volumes (*LV1* to *LV4*) are created out of the deduplicated storage pool. In this way, VDO can support multiprotocol unified block or file access to the underlying deduplicated storage pool.

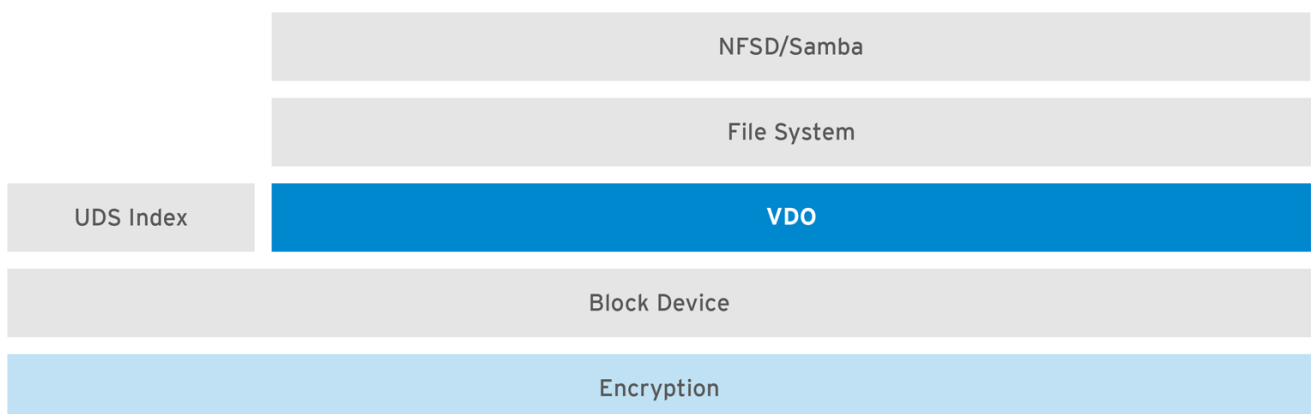


RHEL_466924_0218

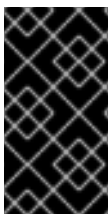
Deduplicated unified storage design enables for multiple file systems to collectively use the same deduplication domain through the LVM tools. Also, file systems can take advantage of LVM snapshot, copy-on-write, and shrink or grow features, all on top of VDO.

Encryption

Device Mapper (DM) mechanisms such as DM Crypt are compatible with VDO. Encrypting VDO volumes helps ensure data security, and any file systems above VDO are still deduplicated.



RHEL_466924_0218



IMPORTANT

Applying the encryption layer above VDO results in little if any data deduplication. Encryption makes duplicate blocks different before VDO can deduplicate them.

Always place the encryption layer below VDO.

1.3. VDO REQUIREMENTS

VDO has certain requirements on its placement and your system resources.

1.3.1. Placement of VDO in the storage stack

You should place certain storage layers under VDO and others above VDO.

A VDO volume is a thinly provisioned block device. To prevent running out of physical space, place the volume on top of storage that you can expand at a later time. Examples of such expandable storage are LVM volumes or MD RAID arrays.

You can place thick-provisioned layers on top of VDO, but you cannot rely on the guarantees of thick provisioning in that case. Because the VDO layer is thin-provisioned, the effects of thin provisioning apply to all layers above it. If you do not monitor the VDO device, you might run out of physical space on thick-provisioned volumes above VDO.

Red Hat recommends the following configurations:

Place only under VDO

- DM Multipath
- DM Crypt
- Software RAID (LVM or MD RAID)

Place only above VDO

- LVM cache
- LVM snapshots
- LVM thin provisioning

The following configurations are **not supported**:

- VDO on top of VDO volumes: storage → VDO → LVM → VDO
- VDO on top of LVM snapshots
- VDO on top of LVM cache
- VDO on top of a loopback device
- VDO on top of LVM thin provisioning
- Encrypted volumes on top of VDO: storage → VDO → DM-Crypt
- Partitions on a VDO volume
- RAID (LVM RAID, MD RAID, or any other type) on top of a VDO volume

Additional resources

- For more information on stacking VDO with LVM, see the [Stacking LVM volumes](#) article.

1.3.2. VDO memory requirements

Each VDO volume has two distinct memory requirements:

The VDO module

VDO requires 370 MB of DRAM plus an additional 268 MB per each 1 TB of physical storage managed by the volume.

The Universal Deduplication Service (UDS) index

UDS requires a minimum of 250 MB of DRAM, which is also the default amount that deduplication uses.

The memory required for the UDS index is determined by the index type and the required size of the deduplication window:

Index type	Deduplication window	Note
Dense	1 TB per 1 GB of RAM	A 1 GB dense index is generally sufficient for up to 4 TB of physical storage.
Sparse	10 TB per 1 GB of RAM	A 1 GB sparse index is generally sufficient for up to 40 TB of physical storage.

The UDS Sparse Indexing feature is the recommended mode for VDO. It relies on the temporal locality of data and attempts to retain only the most relevant index entries in memory. With the sparse index, UDS can maintain a deduplication window that is ten times larger than with dense, while using the same amount of memory.

Although the sparse index provides the greatest coverage, the dense index provides more deduplication advice. For most workloads, given the same amount of memory, the difference in deduplication rates between dense and sparse indexes is negligible.

Additional resources

- For concrete examples of UDS index memory requirements, see [Section 1.3.4, “Examples of VDO requirements by physical volume size”](#).

1.3.3. VDO storage space requirements

You can configure a VDO volume to use up to 256 TB of physical storage. Only a certain part of the physical storage is usable to store data. This section provides the calculations to determine the usable size of a VDO-managed volume.

VDO requires storage for two types of VDO metadata and for the UDS index:

- The first type of VDO metadata uses approximately 1 MB for each 4 GB of *physical storage* plus an additional 1 MB per slab.
- The second type of VDO metadata consumes approximately 1.25 MB for each 1 GB of *logical storage*, rounded up to the nearest slab.
- The amount of storage required for the UDS index depends on the type of index and the amount of RAM allocated to the index. For each 1 GB of RAM, a dense UDS index uses 17 GB of storage, and a sparse UDS index will use 170 GB of storage.

Additional resources

- For concrete examples of VDO storage requirements, see [Section 1.3.4, “Examples of VDO requirements by physical volume size”](#).

1.3.4. Examples of VDO requirements by physical volume size

The following tables provide approximate system requirements of VDO based on the size of the underlying physical volume. Each table lists requirements appropriate to the intended deployment, such as primary storage or backup storage.

The exact numbers depend on your configuration of the VDO volume.

Primary storage deployment

In the primary storage case, the UDS index is between 0.01% to 25% the size of the physical volume.

Table 1.1. Storage and memory requirements for primary storage

Physical volume size	RAM usage	Disk usage	Index type
10GB–1TB	250MB	2.5 GB	Dense
2–10TB	1GB	10GB	Dense
	250MB	22GB	Sparse
11–50TB	2GB	170GB	Sparse
51–100TB	3GB	255GB	Sparse
101–256TB	12GB	1020GB	Sparse

Backup storage deployment

In the backup storage case, the UDS index covers the size of the backup set but is not bigger than the physical volume. If you expect the backup set or the physical size to grow in the future, factor this into the index size.

Table 1.2. Storage and memory requirements for backup storage

Physical volume size	RAM usage	Disk usage	Index type
10GB–1TB	250MB	2.5 GB	Dense
2–10TB	2GB	170GB	Sparse
11–50TB	10GB	850GB	Sparse
51–100TB	20GB	1700GB	Sparse
101–256TB	26GB	3400GB	Sparse

1.4. INSTALLING VDO

This procedure installs software necessary to create, mount, and manage VDO volumes.

Procedure

- Install the **vdo** and **kmod-kvdo** packages:

install the `vdo` and `kmod-kvdo` packages:

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

1.5. CREATING A VDO VOLUME

This procedure creates a VDO volume on a block device.

Prerequisites

- Install the VDO software. See [Section 1.4, “Installing VDO”](#).
- Use expandable storage as the backing block device. For more information, see [Section 1.3.1, “Placement of VDO in the storage stack”](#).

Procedure

In all the following steps, replace *vdo-name* with the identifier you want to use for your VDO volume; for example, **vdo1**. You must use a different name and device for each instance of VDO on the system.

1. Find a persistent name for the block device where you want to create the VDO volume. For more information on persistent names, see [Chapter 4, Overview of persistent naming attributes](#). If you use a non-persistent device name, then VDO might fail to start properly in the future if the device name changes.
2. Create the VDO volume:

```
# vdo create \
  --name=vdo-name \
  --device=block-device \
  --vdoLogicalSize=logical-size
```

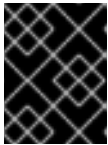
- Replace *block-device* with the persistent name of the block device where you want to create the VDO volume. For example, **/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f**.
- Replace *logical-size* with the amount of logical storage that the VDO volume should present:
 - For active VMs or container storage, use logical size that is **ten** times the physical size of your block device. For example, if your block device is 1TB in size, use **10T** here.
 - For object storage, use logical size that is **three** times the physical size of your block device. For example, if your block device is 1TB in size, use **3T** here.
- If the physical block device is larger than 16TiB, add the **--vdoSlabSize=32G** option to increase the slab size on the volume to 32GiB. Using the default slab size of 2GiB on block devices larger than 16TiB results in the **vdo create** command failing with the following error:

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

Example 1.1. Creating VDO for container storage

For example, to create a VDO volume for container storage on a 1TB block device, you might use:

```
# vdo create \
  --name=vdo1 \
  --device=/dev/disk/by-id/scsi-3600508b1001c264ad2af21e903ad031f \
  --vdoLogicalSize=10T
```



IMPORTANT

If a failure occurs when creating the VDO volume, remove the volume to clean up. See [Section 2.11.2, “Removing an unsuccessfully created VDO volume”](#) for details.

3. Create a file system on top of the VDO volume:

- For the XFS file system:

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

- For the ext4 file system:

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

4. Use the following command to wait for the system to register the new device node:

```
# udevadm settle
```

Next steps

1. Mount the file system. See [Section 1.6, “Mounting a VDO volume”](#) for details.
2. Enable the **discard** feature for the file system on your VDO device. See [Section 1.7, “Enabling periodic block discard”](#) for details.

Additional resources

- The **vdo(8)** man page

1.6. MOUNTING A VDO VOLUME

This procedure mounts a file system on a VDO volume, either manually or persistently.

Prerequisites

- A VDO volume has been created on your system. For instructions, see [Section 1.5, “Creating a VDO volume”](#).

Procedure

- To mount the file system on the VDO volume manually, use:

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

- To configure the file system to mount automatically at boot, add a line to the **/etc/fstab** file:

- For the XFS file system:

```
/dev/mapper/vdo-name mount-point xfs defaults,_netdev,x-systemd.device-timeout=0,x-  
systemd.requires=vdo.service 0 0
```

- For the ext4 file system:

```
/dev/mapper/vdo-name mount-point ext4 defaults,_netdev,x-systemd.device-timeout=0,x-  
systemd.requires=vdo.service 0 0
```

1.7. ENABLING PERIODIC BLOCK DISCARD

This procedure enables a **systemd** timer that regularly discards unused blocks on all supported file systems.

Procedure

- Enable and start the **systemd** timer:

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

1.8. MONITORING VDO

This procedure describes how to obtain usage and efficiency information from a VDO volume.

Prerequisites

- Install the VDO software. See [Section 1.4, “Installing VDO”](#).

Procedure

- Use the **vdostats** utility to get information about a VDO volume:

```
# vdostats --human-readable
```

Device	1K-blocks	Used	Available	Use%	Space saving%
/dev/mapper/node1osd1	926.5G	21.0G	905.5G	2%	73%
/dev/mapper/node1osd2	926.5G	28.2G	898.3G	3%	64%

Additional resources

- The **vdostats(8)** man page.

CHAPTER 2. MAINTAINING VDO

After deploying a VDO volume, you can perform certain tasks to maintain or optimize it. Some of the following tasks are required for the correct functioning of VDO volumes.

2.1. PREREQUISITES

- VDO is installed and deployed. See [Chapter 1, Deploying VDO](#).

2.2. MANAGING FREE SPACE ON VDO VOLUMES

VDO is a thinly provisioned block storage target. Because of that, you must actively monitor and manage space usage on VDO volumes.

2.2.1. Thin provisioning in VDO

VDO is a thinly provisioned block storage target. The amount of physical space that a VDO volume uses might differ from the size of the volume that is presented to users of the storage.

You can make use of this disparity to save on storage costs. Take care to avoid unexpectedly running out of storage space, if the data written does not achieve the expected rate of optimization.

Whenever the number of logical blocks (virtual storage) exceeds the number of physical blocks (actual storage), it becomes possible for file systems and applications to unexpectedly run out of space. For that reason, storage systems using VDO must provide you with a way of monitoring the size of the free pool on the VDO volume.

You can determine the size of this free pool by using the **vdostats** utility. The default output of this utility lists information for all running VDO volumes in a format similar to the Linux **df** utility. For example:

Device	1K-blocks	Used	Available	Use%
/dev/mapper/my-vdo	211812352	105906176	105906176	50%

When the physical storage capacity of a VDO volume is almost full, VDO reports a warning in the system log, similar to the following:

```
Oct 2 17:13:39 system lvm[13863]: Monitoring VDO pool my-vdo.
Oct 2 17:27:39 system lvm[13863]: WARNING: VDO pool my-vdo is now 80.69% full.
Oct 2 17:28:19 system lvm[13863]: WARNING: VDO pool my-vdo is now 85.25% full.
Oct 2 17:29:39 system lvm[13863]: WARNING: VDO pool my-vdo is now 90.64% full.
Oct 2 17:30:29 system lvm[13863]: WARNING: VDO pool my-vdo is now 96.07% full.
```



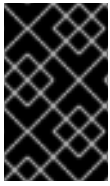
NOTE

These warning messages appear only when the **lvm2-monitor** service is running. It is enabled by default.

If the size of free pool drops below a certain level, you can take action by:

- Deleting data. This reclaims space whenever the deleted data is not duplicated. Deleting data frees the space only after discards are issued.
- Adding physical storage

- Deleting LUNs on top of VDO



IMPORTANT

Monitor physical space on your VDO volumes to prevent out-of-space situations. Running out of physical blocks might result in losing recently written, unacknowledged data on the VDO volume.

2.2.2. Monitoring VDO

This procedure describes how to obtain usage and efficiency information from a VDO volume.

Prerequisites

- Install the VDO software. See [Section 1.4, “Installing VDO”](#).

Procedure

- Use the **vdostats** utility to get information about a VDO volume:

```
# vdostats --human-readable
```

Device	1K-blocks	Used	Available	Use%	Space saving%
/dev/mapper/node1osd1	926.5G	21.0G	905.5G	2%	73%
/dev/mapper/node1osd2	926.5G	28.2G	898.3G	3%	64%

Additional resources

- The **vdostats(8)** man page.

2.2.3. Reclaiming space for VDO on file systems

This procedure reclaims storage space on a VDO volume that hosts a file system.

VDO cannot reclaim space unless file systems communicate that blocks are free using the **DISCARD**, **TRIM**, or **UNMAP** commands.

Procedure

- If the file system on your VDO volume supports discard operations, enable them. See [Chapter 3, *Discarding unused blocks*](#).
- For file systems that do not use **DISCARD**, **TRIM**, or **UNMAP**, you can manually reclaim free space. Store a file consisting of binary zeros and then delete that file.

2.2.4. Reclaiming space for VDO without a file system

This procedure reclaims storage space on a VDO volume that is used as a block storage target without a file system.

Procedure

- Use the **blkdiscard** utility.
For example, a single VDO volume can be carved up into multiple subvolumes by deploying LVM on top of it. Before deprovisioning a logical volume, use the **blkdiscard** utility to free the space previously used by that logical volume.

LVM supports the **REQ_DISCARD** command and forwards the requests to VDO at the appropriate logical block addresses in order to free the space. If you use other volume managers, they also need to support **REQ_DISCARD**, or equivalently, **UNMAP** for SCSI devices or **TRIM** for ATA devices.

Additional resources

- The **blkdiscard(8)** man page

2.2.5. Reclaiming space for VDO on Fibre Channel or Ethernet network

This procedure reclaims storage space on VDO volumes (or portions of volumes) that are provisioned to hosts on a Fibre Channel storage fabric or an Ethernet network using SCSI target frameworks such as LIO or SCST.

Procedure

- SCSI initiators can use the **UNMAP** command to free space on thinly provisioned storage targets, but the SCSI target framework needs to be configured to advertise support for this command. This is typically done by enabling thin provisioning on these volumes. Verify support for **UNMAP** on Linux-based SCSI initiators by running the following command:

```
# sg_vpd --page=0xb0 /dev/device
```

In the output, verify that the *Maximum unmap LBA count* value is greater than zero.

2.3. STARTING OR STOPPING VDO VOLUMES

You can start or stop a given VDO volume, or all VDO volumes, and their associated UDS indexes.

2.3.1. Started and activated VDO volumes

During the system boot, the **vdo systemd** unit automatically *starts* all VDO devices that are configured as *activated*.

The **vdo systemd** unit is installed and enabled by default when the **vdo** package is installed. This unit automatically runs the **vdo start --all** command at system startup to bring up all activated VDO volumes.

You can also create a VDO volume that does not start automatically by adding the **--activate=disabled** option to the **vdo create** command.

The starting order

Some systems might place LVM volumes both above VDO volumes and below them. On these systems, it is necessary to start services in the right order:

1. The lower layer of LVM must start first. In most systems, starting this layer is configured automatically when the LVM package is installed.
2. The **vdo systemd** unit must start then.
3. Finally, additional scripts must run in order to start LVM volumes or other services on top of the running VDO volumes.

How long it takes to stop a volume

Stopping a VDO volume takes time based on the speed of your storage device and the amount of data that the volume needs to write:

- The volume always writes around 1GiB for every 1GiB of the UDS index.
- With a sparse UDS index, the volume additionally writes the amount of data equal to the block map cache size plus up to 8MiB per slab.

2.3.2. Starting a VDO volume

This procedure starts a given VDO volume or all VDO volumes on your system.

Procedure

- To start a given VDO volume, use:

```
# vdo start --name=my-vdo
```

- To start all VDO volumes, use:

```
# vdo start --all
```

Additional resources

- The **vdo(8)** man page

2.3.3. Stopping a VDO volume

This procedure stops a given VDO volume or all VDO volumes on your system.

Procedure

1. Stop the volume.

- To stop a given VDO volume, use:

```
# vdo stop --name=my-vdo
```

- To stop all VDO volumes, use:

```
# vdo stop --all
```

2. Wait for the volume to finish writing data to the disk.

Additional resources

- The **vdo(8)** man page

2.3.4. Related information

- If restarted after an unclean shutdown, VDO performs a rebuild to verify the consistency of its metadata and repairs it if necessary. See [Section 2.6, “Recovering a VDO volume after an unclean shutdown”](#) for more information on the rebuild process.

2.4. AUTOMATICALLY STARTING VDO VOLUMES AT SYSTEM BOOT

You can configure VDO volumes so that they start automatically at system boot. You can also disable the automatic start.

2.4.1. Started and activated VDO volumes

During the system boot, the **vdo systemd** unit automatically *starts* all VDO devices that are configured as *activated*.

The **vdo systemd** unit is installed and enabled by default when the **vdo** package is installed. This unit automatically runs the **vdo start --all** command at system startup to bring up all activated VDO volumes.

You can also create a VDO volume that does not start automatically by adding the **--activate=disabled** option to the **vdo create** command.

The starting order

Some systems might place LVM volumes both above VDO volumes and below them. On these systems, it is necessary to start services in the right order:

1. The lower layer of LVM must start first. In most systems, starting this layer is configured automatically when the LVM package is installed.
2. The **vdo systemd** unit must start then.
3. Finally, additional scripts must run in order to start LVM volumes or other services on top of the running VDO volumes.

How long it takes to stop a volume

Stopping a VDO volume takes time based on the speed of your storage device and the amount of data that the volume needs to write:

- The volume always writes around 1GiB for every 1GiB of the UDS index.
- With a sparse UDS index, the volume additionally writes the amount of data equal to the block map cache size plus up to 8MiB per slab.

2.4.2. Activating a VDO volume

This procedure activates a VDO volume to enable it to start automatically.

Procedure

- To activate a specific volume:

```
# vdo activate --name=my-vdo
```

- To activate all volumes:

```
# vdo activate --all
```

Additional resources

- The **vdo(8)** man page

2.4.3. Deactivating a VDO volume

This procedure deactivates a VDO volume to prevent it from starting automatically.

Procedure

- To deactivate a specific volume:

```
# vdo deactivate --name=my-vdo
```

- To deactivate all volumes:

```
# vdo deactivate --all
```

Additional resources

- The **vdo(8)** man page

2.5. SELECTING A VDO WRITE MODE

You can configure write mode for a VDO volume, based on what the underlying block device requires. By default, VDO selects write mode automatically.

2.5.1. VDO write modes

VDO supports the following write modes:

sync

When VDO is in **sync** mode, the layers above it assume that a write command writes data to persistent storage. As a result, it is not necessary for the file system or application, for example, to issue FLUSH or force unit access (FUA) requests to cause the data to become persistent at critical points.

VDO must be set to **sync** mode only when the underlying storage guarantees that data is written to persistent storage when the write command completes. That is, the storage must either have no volatile write cache, or have a write through cache.

async

When VDO is in **async** mode, VDO does not guarantee that the data is written to persistent storage when a write command is acknowledged. The file system or application must issue FLUSH or FUA requests to ensure data persistence at critical points in each transaction.

VDO must be set to **async** mode if the underlying storage does not guarantee that data is written to persistent storage when the write command completes; that is, when the storage has a volatile write back cache.

**WARNING**

When VDO is running in **async** mode, it is not compliant with Atomicity, Consistency, Isolation, Durability (ACID). When there is an application or a file system that assumes ACID compliance on top of the VDO volume, **async** mode might cause unexpected data loss.

auto

The **auto** mode automatically selects **sync** or **async** based on the characteristics of each device. This is the default option.

2.5.2. The internal processing of VDO write modes

This section provides details on how the **sync** and **async** VDO write modes operate.

If the **kvdo** module is operating in synchronous mode:

1. It temporarily writes the data in the request to the allocated block and then acknowledges the request.
2. Once the acknowledgment is complete, an attempt is made to deduplicate the block by computing a MurmurHash-3 signature of the block data, which is sent to the VDO index.
3. If the VDO index contains an entry for a block with the same signature, **kvdo** reads the indicated block and does a byte-by-byte comparison of the two blocks to verify that they are identical.
4. If they are indeed identical, then **kvdo** updates its block map so that the logical block points to the corresponding physical block and releases the allocated physical block.
5. If the VDO index did not contain an entry for the signature of the block being written, or the indicated block does not actually contain the same data, **kvdo** updates its block map to make the temporary physical block permanent.

If **kvdo** is operating in asynchronous mode:

1. Instead of writing the data, it will immediately acknowledge the request.
2. It will then attempt to deduplicate the block in same manner as described above.
3. If the block turns out to be a duplicate, **kvdo** updates its block map and releases the allocated block. Otherwise, it writes the data in the request to the allocated block and updates the block map to make the physical block permanent.

2.5.3. Checking the write mode on a VDO volume

This procedure lists the active write mode on a selected VDO volume.

Procedure

- Use the following command to see the write mode used by a VDO volume:

```
# vdo status --name=my-vdo
```

The output lists:

- The *configured write policy*, which is the option selected from **sync**, **async**, or **auto**
- The *write policy*, which is the particular write mode that VDO applied, that is either **sync** or **async**

2.5.4. Checking for a volatile cache

This procedure determines if a block device has a volatile cache or not. You can use the information to choose between the **sync** and **async** VDO write modes.

Procedure

1. Use either of the following methods to determine if a device has a writeback cache:

- Read the `/sys/block/block-device/device/scsi_disk/identifier/cache_type` **sysfs** file. For example:

```
$ cat '/sys/block/sda/device/scsi_disk/7:0:0:0/cache_type'
```

```
write back
```

```
$ cat '/sys/block/sdb/device/scsi_disk/1:2:0:0/cache_type'
```

```
None
```

- Alternatively, you can find whether the above mentioned devices have a write cache or not in the kernel boot log:

```
sd 7:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
```

```
sd 1:2:0:0: [sdb] Write cache: disabled, read cache: disabled, supports DPO and FUA
```

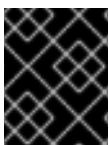
2. In the previous examples:

- Device **sda** indicates that it *has* a writeback cache. Use **async** mode for it.
- Device **sdb** indicates that it *does not have* a writeback cache. Use **sync** mode for it.

You should configure VDO to use the **sync** write mode if the **cache_type** value is **None** or **write through**.

2.5.5. Setting a VDO write mode

This procedure sets a write mode for a VDO volume, either for an existing one or when creating a new volume.



IMPORTANT

Using an incorrect write mode might result in data loss after a power failure, a system crash, or any unexpected loss of contact with the disk.

Prerequisites

- Determine which write mode is correct for your device. See [Section 2.5.4, “Checking for a volatile cache”](#).

Procedure

- You can set a write mode either on an existing VDO volume or when creating a new volume:

- To modify an existing VDO volume, use:

```
# vdo changeWritePolicy --writePolicy=sync/async/auto \
    --name=vdo-name
```

- To specify a write mode when creating a VDO volume, add the **--writePolicy=sync/async/auto** option to the **vdo create** command.

2.6. RECOVERING A VDO VOLUME AFTER AN UNCLEAN SHUTDOWN

You can recover a VDO volume after an unclean shutdown to enable it to continue operating. The task is mostly automated. Additionally, you can clean up after a VDO volume was unsuccessfully created because of a failure in the process.

2.6.1. VDO write modes

VDO supports the following write modes:

sync

When VDO is in **sync** mode, the layers above it assume that a write command writes data to persistent storage. As a result, it is not necessary for the file system or application, for example, to issue FLUSH or force unit access (FUA) requests to cause the data to become persistent at critical points.

VDO must be set to **sync** mode only when the underlying storage guarantees that data is written to persistent storage when the write command completes. That is, the storage must either have no volatile write cache, or have a write through cache.

async

When VDO is in **async** mode, VDO does not guarantee that the data is written to persistent storage when a write command is acknowledged. The file system or application must issue FLUSH or FUA requests to ensure data persistence at critical points in each transaction.

VDO must be set to **async** mode if the underlying storage does not guarantee that data is written to persistent storage when the write command completes; that is, when the storage has a volatile write back cache.



WARNING

When VDO is running in **async** mode, it is not compliant with Atomicity, Consistency, Isolation, Durability (ACID). When there is an application or a file system that assumes ACID compliance on top of the VDO volume, **async** mode might cause unexpected data loss.

auto

The **auto** mode automatically selects **sync** or **async** based on the characteristics of each device. This is the default option.

2.6.2. VDO volume recovery

When a VDO volume restarts after an unclean shutdown, VDO performs the following actions:

- Verifies the consistency of the metadata on the volume.
- Rebuilds a portion of the metadata to repair it if necessary.

Rebuilds are automatic and do not require user intervention.

VDO might rebuild different writes depending on the active write mode:

sync

If VDO was running on synchronous storage and write policy was set to **sync**, all data written to the volume are fully recovered.

async

If the write policy was **async**, some writes might not be recovered if they were not made durable. This is done by sending VDO a **FLUSH** command or a write I/O tagged with the FUA (force unit access) flag. You can accomplish this from user mode by invoking a data integrity operation like **fsync**, **fdatasync**, **sync**, or **umount**.

In either mode, some writes that were either unacknowledged or not followed by a flush might also be rebuilt.

Automatic and manual recovery

When a VDO volume enters **recovering** operating mode, VDO automatically rebuilds the unclean VDO volume after the it comes back online. This is called *online recovery*.

If VDO cannot recover a VDO volume successfully, it places the volume in **read-only** operating mode that persists across volume restarts. You need to fix the problem manually by forcing a rebuild.

Additional resources

- For more information on automatic and manual recovery and VDO operating modes, see [Section 2.6.3, “VDO operating modes”](#).

2.6.3. VDO operating modes

This section describes the modes that indicate whether a VDO volume is operating normally or is recovering from an error.

You can display the current operating mode of a VDO volume using the **vdostats --verbose device** command. See the *Operating mode* attribute in the output.

normal

This is the default operating mode. VDO volumes are always in **normal** mode, unless either of the following states forces a different mode. A newly created VDO volume starts in **normal** mode.

recovering

When a VDO volume does not save all of its metadata before shutting down, it automatically enters **recovering** mode the next time that it starts up. The typical reasons for entering this mode are sudden power loss or a problem from the underlying storage device.

In **recovering** mode, VDO is fixing the references counts for each physical block of data on the device. Recovery usually does not take very long. The time depends on how large the VDO volume is, how fast the underlying storage device is, and how many other requests VDO is handling simultaneously. The VDO volume functions normally with the following exceptions:

- Initially, the amount of space available for write requests on the volume might be limited. As more of the metadata is recovered, more free space becomes available.
- Data written while the VDO volume is recovering might fail to deduplicate against data written before the crash if that data is in a portion of the volume that has not yet been recovered. VDO can compress data while recovering the volume. You can still read or overwrite compressed blocks.
- During an online recovery, certain statistics are unavailable: for example, *blocks in use* and *blocks free*. These statistics become available when the rebuild is complete.
- Response times for reads and writes might be slower than usual due to the ongoing recovery work

You can safely shut down the VDO volume in **recovering** mode. If the recovery does not finish before shutting down, the device enters **recovering** mode again the next time that it starts up.

The VDO volume automatically exits **recovering** mode and moves to **normal** mode when it has fixed all the reference counts. No administrator action is necessary. For details, see [Section 2.6.4, “Recovering a VDO volume online”](#).

read-only

When a VDO volume encounters a fatal internal error, it enters **read-only** mode. Events that might cause **read-only** mode include metadata corruption or the backing storage device becoming read-only. This mode is an error state.

In **read-only** mode, data reads work normally but data writes always fail. The VDO volume stays in **read-only** mode until an administrator fixes the problem.

You can safely shut down a VDO volume in **read-only** mode. The mode usually persists after the VDO volume is restarted. In rare cases, the VDO volume is not able to record the **read-only** state to the backing storage device. In these cases, VDO attempts to do a recovery instead.

Once a volume is in read-only mode, there is no guarantee that data on the volume has not been lost or corrupted. In such cases, Red Hat recommends copying the data out of the read-only volume and possibly restoring the volume from backup.

If the risk of data corruption is acceptable, it is possible to force an offline rebuild of the VDO volume metadata so the volume can be brought back online and made available. The integrity of the rebuilt data cannot be guaranteed. For details, see [Section 2.6.5, “Forcing an offline rebuild of a VDO volume metadata”](#).

2.6.4. Recovering a VDO volume online

This procedure performs an online recovery on a VDO volume to recover metadata after an unclean shutdown.

Procedure

1. If the VDO volume is not already started, start it:

```
# vdo start --name=my-vdo
```

No additional steps are necessary. The recovery runs in the background.

2. If you rely on volume statistics like *blocks in use* and *blocks free*, wait until they are available.

2.6.5. Forcing an offline rebuild of a VDO volume metadata

This procedure performs a forced offline rebuild of a VDO volume metadata to recover after an unclean shutdown.



WARNING

This procedure might cause data loss on the volume.

Prerequisites

- The VDO volume is started.

Procedure

1. Check if the volume is in read-only mode. See the *operating mode* attribute in the command output:

```
# vdo status --name=my-vdo
```

If the volume is not in read-only mode, it is not necessary to force an offline rebuild. Perform an online recovery as described in [Section 2.6.4, "Recovering a VDO volume online"](#).

2. Stop the volume if it is running:

```
# vdo stop --name=my-vdo
```

3. Restart the volume with the **--forceRebuild** option:

```
# vdo start --name=my-vdo --forceRebuild
```

2.6.6. Removing an unsuccessfully created VDO volume

This procedure cleans up a VDO volume in an intermediate state. A volume is left in an intermediate state if a failure occurs when creating the volume. This might happen when, for example:

- The system crashes
- Power fails
- The administrator interrupts a running **vdo create** command

Procedure

- To clean up, remove the unsuccessfully created volume with the **--force** option:

```
# vdo remove --force --name=my-vdo
```

The **--force** option is required because the administrator might have caused a conflict by changing the system configuration since the volume was unsuccessfully created.

Without the **--force** option, the **vdo remove** command fails with the following message:

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my-vdo:
umount -f /dev/mapper/my-vdo
udevadm settle
dmsetup remove my-vdo
vdo: ERROR - VDO volume my-vdo previous operation (create) is incomplete
```

2.7. OPTIMIZING THE UDS INDEX

You can configure certain settings of the UDS index to optimize it on your system.



IMPORTANT

You cannot change the properties of the UDS index *after* creating the VDO volume.

2.7.1. The UDS index

VDO uses a high-performance deduplication index called UDS to detect duplicate blocks of data as they are being stored.

The UDS index provides the foundation of the VDO product. For each new piece of data, it quickly determines if that piece is identical to any previously stored piece of data. If the index finds match, the storage system can then internally reference the existing item to avoid storing the same information more than once.

The UDS index runs inside the kernel as the **uds** kernel module.

The *deduplication window* is the number of previously written blocks that the index remembers. The size of the deduplication window is configurable. For a given window size, the index requires a specific amount of RAM and a specific amount of disk space. The size of the window is usually determined by specifying the size of the index memory using the **--indexMem=size** option. VDO then determines the amount of disk space to use automatically.

The UDS index consists of two parts:

- A compact representation is used in memory that contains at most one entry per unique block.
- An on-disk component that records the associated block names presented to the index as they occur, in order.

UDS uses an average of 4 bytes per entry in memory, including cache.

The on-disk component maintains a bounded history of data passed to UDS. UDS provides deduplication advice for data that falls within this deduplication window, containing the names of the most recently seen blocks. The deduplication window allows UDS to index data as efficiently as possible while limiting the amount of memory required to index large data repositories. Despite the bounded nature of the deduplication window, most datasets which have high levels of deduplication also exhibit a high degree of temporal locality – in other words, most deduplication occurs among sets of blocks that were written at about the same time. Furthermore, in general, data being written is more likely to duplicate data that was recently written than data that was written a long time ago. Therefore, for a given workload over a given time interval, deduplication rates will often be the same whether UDS indexes only the most recent data or all the data.

Because duplicate data tends to exhibit temporal locality, it is rarely necessary to index every block in the storage system. Were this not so, the cost of index memory would outstrip the savings of reduced storage costs from deduplication. Index size requirements are more closely related to the rate of data ingestion. For example, consider a storage system with 100 TB of total capacity but with an ingestion rate of 1 TB per week. With a deduplication window of 4 TB, UDS can detect most redundancy among the data written within the last month.

2.7.2. Recommended UDS index configuration

This section describes the recommended options to use with the UDS index, based on your intended use case.

In general, Red Hat recommends using a **sparse** UDS index for all production use cases. This is an extremely efficient indexing data structure, requiring approximately one-tenth of a byte of DRAM per block in its deduplication window. On disk, it requires approximately 72 bytes of disk space per block. The minimum configuration of this index uses 256 MB of DRAM and approximately 25 GB of space on disk.

To use this configuration, specify the **--sparseIndex=enabled --indexMem=0.25** options to the **vdo create** command. This configuration results in a deduplication window of 2.5 TB (meaning it will remember a history of 2.5 TB). For most use cases, a deduplication window of 2.5 TB is appropriate for deduplicating storage pools that are up to 10 TB in size.

The default configuration of the index, however, is to use a **dense** index. This index is considerably less efficient (by a factor of 10) in DRAM, but it has much lower (also by a factor of 10) minimum required disk space, making it more convenient for evaluation in constrained environments.

In general, a deduplication window that is one quarter of the physical size of a VDO volume is a recommended configuration. However, this is not an actual requirement. Even small deduplication windows (compared to the amount of physical storage) can find significant amounts of duplicate data in many use cases. Larger windows may also be used, but in most cases, there will be little additional benefit to doing so.

Additional resources

- Speak with your Red Hat Technical Account Manager representative for additional guidelines on tuning this important system parameter.

2.8. ENABLING OR DISABLING DEDUPLICATION IN VDO

In some instances, you might want to temporarily disable deduplication of data being written to a VDO volume while still retaining the ability to read to and write from the volume. Disabling deduplication prevents subsequent writes from being deduplicated, but the data that was already deduplicated remains so.

2.8.1. Deduplication in VDO

Deduplication is a technique for reducing the consumption of storage resources by eliminating multiple copies of duplicate blocks.

Instead of writing the same data more than once, VDO detects each duplicate block and records it as a reference to the original block. VDO maintains a mapping from logical block addresses, which are used by the storage layer above VDO, to physical block addresses, which are used by the storage layer under VDO.

After deduplication, multiple logical block addresses can be mapped to the same physical block address. These are called shared blocks. Block sharing is invisible to users of the storage, who read and write blocks as they would if VDO were not present.

When a shared block is overwritten, VDO allocates a new physical block for storing the new block data to ensure that other logical block addresses that are mapped to the shared physical block are not modified.

2.8.2. Enabling deduplication on a VDO volume

This procedure restarts the associated UDS index and informs the VDO volume that deduplication is active again.



NOTE

Deduplication is enabled by default.

Procedure

- To restart deduplication on a VDO volume, use the following command:

```
# vdo enableDeduplication --name=my-vdo
```

2.8.3. Disabling deduplication on a VDO volume

This procedure stops the associated UDS index and informs the VDO volume that deduplication is no longer active.

Procedure

- To stop deduplication on a VDO volume, use the following command:

```
# vdo disableDeduplication --name=my-vdo
```

- You can also disable deduplication when creating a new VDO volume by adding the **--deduplication=disabled** option to the **vdo create** command.

2.9. ENABLING OR DISABLING COMPRESSION IN VDO

VDO provides data compression. You can disable it to maximize performance or to speed processing of data that is unlikely to compress, or re-enable it to increase space savings.

2.9.1. Compression in VDO

In addition to block-level deduplication, VDO also provides inline block-level compression using the HIOPS Compression™ technology.

VDO volume compression is on by default.

While deduplication is the optimal solution for virtual machine environments and backup applications, compression works very well with structured and unstructured file formats that do not typically exhibit block-level redundancy, such as log files and databases.

Compression operates on blocks that have not been identified as duplicates. When VDO sees unique data for the first time, it compresses the data. Subsequent copies of data that have already been stored are deduplicated without requiring an additional compression step.

The compression feature is based on a parallelized packaging algorithm that enables it to handle many compression operations at once. After first storing the block and responding to the requestor, a best-fit packing algorithm finds multiple blocks that, when compressed, can fit into a single physical block. After it is determined that a particular physical block is unlikely to hold additional compressed blocks, it is written to storage and the uncompressed blocks are freed and reused.

By performing the compression and packaging operations after having already responded to the requestor, using compression imposes a minimal latency penalty.

2.9.2. Enabling compression on a VDO volume

This procedure enables compression on a VDO volume to increase space savings.



NOTE

Compression is enabled by default.

Procedure

- To start it again, use the following command:

```
# vdo enableCompression --name=my-vdo
```

2.9.3. Disabling compression on a VDO volume

This procedure stops compression on a VDO volume to maximize performance or to speed processing of data that is unlikely to compress.

Procedure

- To stop compression on an existing VDO volume, use the following command:

```
# vdo disableCompression --name=my-vdo
```

- Alternatively, you can disable compression by adding the **--compression=disabled** option to the **vdo create** command when creating a new volume.

2.10. INCREASING THE SIZE OF A VDO VOLUME

You can increase the physical size of a VDO volume to utilize more underlying storage capacity, or the logical size to provide more capacity on the volume.

2.10.1. Thin provisioning in VDO

VDO is a thinly provisioned block storage target. The amount of physical space that a VDO volume uses might differ from the size of the volume that is presented to users of the storage.

You can make use of this disparity to save on storage costs. Take care to avoid unexpectedly running out of storage space, if the data written does not achieve the expected rate of optimization.

Whenever the number of logical blocks (virtual storage) exceeds the number of physical blocks (actual storage), it becomes possible for file systems and applications to unexpectedly run out of space. For that reason, storage systems using VDO must provide you with a way of monitoring the size of the free pool on the VDO volume.

You can determine the size of this free pool by using the **vdostats** utility. The default output of this utility lists information for all running VDO volumes in a format similar to the Linux **df** utility. For example:

```
Device          1K-blocks  Used    Available  Use%
/dev/mapper/my-vdo 211812352 105906176 105906176  50%
```

When the physical storage capacity of a VDO volume is almost full, VDO reports a warning in the system log, similar to the following:

```
Oct 2 17:13:39 system lvm[13863]: Monitoring VDO pool my-vdo.
Oct 2 17:27:39 system lvm[13863]: WARNING: VDO pool my-vdo is now 80.69% full.
Oct 2 17:28:19 system lvm[13863]: WARNING: VDO pool my-vdo is now 85.25% full.
Oct 2 17:29:39 system lvm[13863]: WARNING: VDO pool my-vdo is now 90.64% full.
Oct 2 17:30:29 system lvm[13863]: WARNING: VDO pool my-vdo is now 96.07% full.
```



NOTE

These warning messages appear only when the **lvm2-monitor** service is running. It is enabled by default.

If the size of free pool drops below a certain level, you can take action by:

- Deleting data. This reclaims space whenever the deleted data is not duplicated. Deleting data frees the space only after discards are issued.
- Adding physical storage
- Deleting LUNs on top of VDO



IMPORTANT

Monitor physical space on your VDO volumes to prevent out-of-space situations. Running out of physical blocks might result in losing recently written, unacknowledged data on the VDO volume.

2.10.2. Increasing the logical size of a VDO volume

This procedure increases the logical size of a given VDO volume. It enables you to initially create VDO volumes that have a logical size small enough to be safe from running out of space. After some period of time, you can evaluate the actual rate of data reduction, and if sufficient, you can grow the logical size of the VDO volume to take advantage of the space savings.

It is not possible to decrease the logical size of a VDO volume.

Procedure

- To grow the logical size, use:

```
# vdo growLogical --name=my-vdo \  
    --vdoLogicalSize=new-logical-size
```

When the logical size increases, VDO informs any devices or file systems on top of the volume of the new size.

2.10.3. Increasing the physical size of a VDO volume

This procedure increases the amount of physical storage available to a VDO volume.

It is not possible to shrink a VDO volume in this way.

Prerequisites

- The underlying block device has a larger capacity than the current physical size of the VDO volume.
If it does not, you can attempt to increase the size of the device. The exact procedure depends on the type of the device. For example, to resize an MBR or GPT partition, see the [Resizing a partition](#) section in the *Managing storage devices* guide.

Procedure

- Add the new physical storage space to the VDO volume:

```
# vdo growPhysical --name=my-vdo
```

2.11. REMOVING VDO VOLUMES

You can remove an existing VDO volume on your system.

2.11.1. Removing a working VDO volume

This procedure removes a VDO volume and its associated UDS index.

Procedure

1. Unmount the file systems and stop the applications that are using the storage on the VDO volume.
2. To remove the VDO volume from your system, use:

```
# vdo remove --name=my-vdo
```

2.11.2. Removing an unsuccessfully created VDO volume

This procedure cleans up a VDO volume in an intermediate state. A volume is left in an intermediate state if a failure occurs when creating the volume. This might happen when, for example:

- The system crashes
- Power fails
- The administrator interrupts a running **vdo create** command

Procedure

- To clean up, remove the unsuccessfully created volume with the **--force** option:

```
# vdo remove --force --name=my-vdo
```

The **--force** option is required because the administrator might have caused a conflict by changing the system configuration since the volume was unsuccessfully created.

Without the **--force** option, the **vdo remove** command fails with the following message:

```
[...]
A previous operation failed.
Recovery from the failure either failed or was interrupted.
Add '--force' to 'remove' to perform the following cleanup.
Steps to clean up VDO my-vdo:
umount -f /dev/mapper/my-vdo
udevadm settle
dmsetup remove my-vdo
vdo: ERROR - VDO volume my-vdo previous operation (create) is incomplete
```

2.12. RELATED INFORMATION

- You can use the **Ansible** tool to automate VDO deployment and administration. For details, see:
 - Ansible documentation: <https://docs.ansible.com/>
 - VDO Ansible module documentation:
https://docs.ansible.com/ansible/latest/modules/vdo_module.html

CHAPTER 3. DISCARDING UNUSED BLOCKS

You can perform or schedule discard operations on block devices that support them.

3.1. BLOCK DISCARD OPERATIONS

Block discard operations discard blocks that are no longer in use by a mounted file system. They are useful on:

- Solid-state drives (SSDs)
- Thinly-provisioned storage

Requirements

The block device underlying the file system must support physical discard operations.

Physical discard operations are supported if the value in the `/sys/block/device/queue/discard_max_bytes` file is not zero.

3.2. TYPES OF BLOCK DISCARD OPERATIONS

You can run discard operations using different methods:

Batch discard

Are run explicitly by the user. They discard all unused blocks in the selected file systems.

Online discard

Are specified at mount time. They run in real time without user intervention. Online discard operations discard only the blocks that are transitioning from used to free.

Periodic discard

Are batch operations that are run regularly by a **systemd** service.

All types are supported by the XFS and ext4 file systems and by VDO.

Recommendations

Red Hat recommends that you use batch or periodic discard.

Use online discard only if:

- the system's workload is such that batch discard is not feasible, or
- online discard operations are necessary to maintain performance.

3.3. PERFORMING BATCH BLOCK DISCARD

This procedure performs a batch block discard operation to discard unused blocks on a mounted file system.

Prerequisites

- The file system is mounted.
- The block device underlying the file system supports physical discard operations.

Procedure

- Use the **fstrim** utility:
 - To perform discard only on a selected file system, use:

```
# fstrim mount-point
```

- To perform discard on all mounted file systems, use:

```
# fstrim --all
```

If you execute the **fstrim** command on:

- a device that does not support discard operations, or
- a logical device (LVM or MD) composed of multiple devices, where any one of the device does not support discard operations,

the following message displays:

```
# fstrim /mnt/non_discard
```

```
fstrim: /mnt/non_discard: the discard operation is not supported
```

Additional resources

- The **fstrim(8)** man page

3.4. ENABLING ONLINE BLOCK DISCARD

This procedure enables online block discard operations that automatically discard unused blocks on all supported file systems.

Procedure

- Enable online discard at mount time:
 - When mounting a file system manually, add the **-o discard** mount option:

```
# mount -o discard device mount-point
```

- When mounting a file system persistently, add the **discard** option to the mount entry in the **/etc/fstab** file.

Additional resources

- The **mount(8)** man page
- The **fstab(5)** man page

3.5. ENABLING PERIODIC BLOCK DISCARD

This procedure enables a **systemd** timer that regularly discards unused blocks on all supported file systems.

Procedure

- Enable and start the **systemd** timer:

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

CHAPTER 4. OVERVIEW OF PERSISTENT NAMING ATTRIBUTES

As a system administrator, you need to refer to storage volumes using persistent naming attributes to build storage setups that are reliable over multiple system boots.

4.1. DISADVANTAGES OF NON-PERSISTENT NAMING ATTRIBUTES

Red Hat Enterprise Linux provides a number of ways to identify storage devices. It is important to use the correct option to identify each device when used in order to avoid inadvertently accessing the wrong device, particularly when installing to or reformatting drives.

Traditionally, non-persistent names in the form of `/dev/sd(major number)(minor number)` are used on Linux to refer to storage devices. The major and minor number range and associated **sd** names are allocated for each device when it is detected. This means that the association between the major and minor number range and associated **sd** names can change if the order of device detection changes.

Such a change in the ordering might occur in the following situations:

- The parallelization of the system boot process detects storage devices in a different order with each system boot.
- A disk fails to power up or respond to the SCSI controller. This results in it not being detected by the normal device probe. The disk is not accessible to the system and subsequent devices will have their major and minor number range, including the associated **sd** names shifted down. For example, if a disk normally referred to as **sdb** is not detected, a disk that is normally referred to as **sdc** would instead appear as **sdb**.
- A SCSI controller (host bus adapter, or HBA) fails to initialize, causing all disks connected to that HBA to not be detected. Any disks connected to subsequently probed HBAs are assigned different major and minor number ranges, and different associated **sd** names.
- The order of driver initialization changes if different types of HBAs are present in the system. This causes the disks connected to those HBAs to be detected in a different order. This might also occur if HBAs are moved to different PCI slots on the system.
- Disks connected to the system with Fibre Channel, iSCSI, or FCoE adapters might be inaccessible at the time the storage devices are probed, due to a storage array or intervening switch being powered off, for example. This might occur when a system reboots after a power failure, if the storage array takes longer to come online than the system take to boot. Although some Fibre Channel drivers support a mechanism to specify a persistent SCSI target ID to WWPN mapping, this does not cause the major and minor number ranges, and the associated **sd** names to be reserved; it only provides consistent SCSI target ID numbers.

These reasons make it undesirable to use the major and minor number range or the associated **sd** names when referring to devices, such as in the `/etc/fstab` file. There is the possibility that the wrong device will be mounted and data corruption might result.

Occasionally, however, it is still necessary to refer to the **sd** names even when another mechanism is used, such as when errors are reported by a device. This is because the Linux kernel uses **sd** names (and also SCSI host/channel/target/LUN tuples) in kernel messages regarding the device.

4.2. FILE SYSTEM AND DEVICE IDENTIFIERS

This section explains the difference between persistent attributes identifying file systems and block devices.

File system identifiers

File system identifiers are tied to a particular file system created on a block device. The identifier is also stored as part of the file system. If you copy the file system to a different device, it still carries the same file system identifier. On the other hand, if you rewrite the device, such as by formatting it with the **mkfs** utility, the device loses the attribute.

File system identifiers include:

- Unique identifier (UUID)
- Label

Device identifiers

Device identifiers are tied to a block device: for example, a disk or a partition. If you rewrite the device, such as by formatting it with the **mkfs** utility, the device keeps the attribute, because it is not stored in the file system.

Device identifiers include:

- World Wide Identifier (WWID)
- Partition UUID
- Serial number

Recommendations

- Some file systems, such as logical volumes, span multiple devices. Red Hat recommends accessing these file systems using file system identifiers rather than device identifiers.

4.3. DEVICE NAMES MANAGED BY THE UDEV MECHANISM IN /DEV/DISK/

This section lists different kinds of persistent naming attributes that the **udev** service provides in the **/dev/disk/** directory.

The **udev** mechanism is used for all types of devices in Linux, not just for storage devices. In the case of storage devices, Red Hat Enterprise Linux contains **udev** rules that create symbolic links in the **/dev/disk/** directory. This enables you to refer to storage devices by:

- Their content
- A unique identifier
- Their serial number.

Although **udev** naming attributes are persistent, in that they do not change on their own across system reboots, some are also configurable.

4.3.1. File system identifiers

The UUID attribute in **/dev/disk/by-uuid/**

Entries in this directory provide a symbolic name that refers to the storage device by a **unique identifier** (UUID) in the content (that is, the data) stored on the device. For example:

```
/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can use the UUID to refer to the device in the **/etc/fstab** file using the following syntax:

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can configure the UUID attribute when creating a file system, and you can also change it later on.

The Label attribute in **/dev/disk/by-label/**

Entries in this directory provide a symbolic name that refers to the storage device by a **label** in the content (that is, the data) stored on the device.

For example:

```
/dev/disk/by-label/Boot
```

You can use the label to refer to the device in the **/etc/fstab** file using the following syntax:

```
LABEL=Boot
```

You can configure the Label attribute when creating a file system, and you can also change it later on.

4.3.2. Device identifiers

The WWID attribute in **/dev/disk/by-id/**

The World Wide Identifier (WWID) is a persistent, **system-independent identifier** that the SCSI Standard requires from all SCSI devices. The WWID identifier is guaranteed to be unique for every storage device, and independent of the path that is used to access the device. The identifier is a property of the device but is not stored in the content (that is, the data) on the devices.

This identifier can be obtained by issuing a SCSI Inquiry to retrieve the Device Identification Vital Product Data (page **0x83**) or Unit Serial Number (page **0x80**).

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current **/dev/sd** name on that system. Applications can use the **/dev/disk/by-id/** name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems.

Example 4.1. WWID mappings

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	/dev/sda	A device with a page 0x83 identifier
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	/dev/sdb	A device with a page 0x80 identifier

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/ata-SAMSUNG_MZNLN256MHQ-000L7_S2WDNX0J336519-part3	/dev/sdc3	A disk partition

In addition to these persistent names provided by the system, you can also use **udev** rules to implement persistent names of your own, mapped to the WWID of the storage.

The Partition UUID attribute in /dev/disk/by-partuuid

The Partition UUID (PARTUUID) attribute identifies partitions as defined by GPT partition table.

Example 4.2. Partition UUID mappings

PARTUUID symlink	Non-persistent device
/dev/disk/by-partuuid/4cd1448a-01	/dev/sda1
/dev/disk/by-partuuid/4cd1448a-02	/dev/sda2
/dev/disk/by-partuuid/4cd1448a-03	/dev/sda3

The Path attribute in /dev/disk/by-path/

This attribute provides a symbolic name that refers to the storage device by the **hardware path** used to access the device.



WARNING

The Path attribute is unreliable, and Red Hat does not recommend using it.

4.4. THE WORLD WIDE IDENTIFIER WITH DM MULTIPATH

This section describes the mapping between the World Wide Identifier (WWID) and non-persistent device names in a Device Mapper Multipath configuration.

If there are multiple paths from a system to a device, DM Multipath uses the WWID to detect this. DM Multipath then presents a single "pseudo-device" in the **/dev/mapper/wwid** directory, such as **/dev/mapper/3600508b400105df70000e00000ac0000**.

The command **multipath -l** shows the mapping to the non-persistent identifiers:

- **Host:Channel:Target:LUN**
- **/dev/sd** name
- **major:minor** number

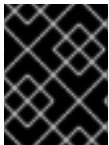
Example 4.3. WWID mappings in a multipath configuration

An example output of the **multipath -l** command:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
\_ 5:0:1:1 sdc 8:32 [active][undef]
\_ 6:0:1:1 sdg 8:96 [active][undef]
\_ round-robin 0 [prio=0][enabled]
\_ 5:0:0:1 sdb 8:16 [active][undef]
\_ 6:0:0:1 sdf 8:80 [active][undef]
```

DM Multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding **/dev/sd** name on the system. These names are persistent across path changes, and they are consistent when accessing the device from different systems.

When the **user_friendly_names** feature of DM Multipath is used, the WWID is mapped to a name of the form **/dev/mapper/mpathN**. By default, this mapping is maintained in the file **/etc/multipath/bindings**. These **mpathN** names are persistent as long as that file is maintained.



IMPORTANT

If you use **user_friendly_names**, then additional steps are required to obtain consistent names in a cluster.

4.5. LIMITATIONS OF THE UDEV DEVICE NAMING CONVENTION

The following are some limitations of the **udev** naming convention:

- It is possible that the device might not be accessible at the time the query is performed because the **udev** mechanism might rely on the ability to query the storage device when the **udev** rules are processed for a **udev** event. This is more likely to occur with Fibre Channel, iSCSI or FCoE storage devices when the device is not located in the server chassis.
- The kernel might send **udev** events at any time, causing the rules to be processed and possibly causing the **/dev/disk/by-*/** links to be removed if the device is not accessible.
- There might be a delay between when the **udev** event is generated and when it is processed, such as when a large number of devices are detected and the user-space **udev** service takes some amount of time to process the rules for each one. This might cause a delay between when the kernel detects the device and when the **/dev/disk/by-*/** names are available.
- External programs such as **blkid** invoked by the rules might open the device for a brief period of time, making the device inaccessible for other uses.

4.6. LISTING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to find out the persistent naming attributes of non-persistent storage devices.

Procedure

- To list the UUID and Label attributes, use the **lsblk** utility:

```
$ lsblk --fs storage-device
```

For example:

Example 4.4. Viewing the UUID and Label of a file system

```
$ lsblk --fs /dev/sda1
```

NAME	FSTYPE	LABEL	UUID	MOUNTPOINT
sda1	xf	Boot	afa5d5e3-9050-48c3-acc1-bb30095f3dc4	/boot

- To list the PARTUUID attribute, use the **lsblk** utility with the **--output +PARTUUID** option:

```
$ lsblk --output +PARTUUID
```

For example:

Example 4.5. Viewing the PARTUUID attribute of a partition

```
$ lsblk --output +PARTUUID /dev/sda1
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT	PARTUUID
sda1	8:1	0	512M	0	part	/boot	4cd1448a-01

- To list the WWID attribute, examine the targets of symbolic links in the **/dev/disk/by-id/** directory. For example:

Example 4.6. Viewing the WWID of all storage devices on the system

```
$ file /dev/disk/by-id/*
```

```
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001:
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1:
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2:
symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root:
../../dm-0
symbolic link to
/dev/disk/by-id/dm-name-rhel_rhel8-swap:
symbolic link
to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUdIVKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g:
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
```

```
QIWtEHtXGobe5bewllUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrIRLhzfMyOKd:
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-wu4QQm:
symbolic link to ../../sda2
```

4.7. MODIFYING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to change the UUID or Label persistent naming attribute of a file system.



NOTE

Changing **udev** attributes happens in the background and might take a long time. The **udevadm settle** command waits until the change is fully registered, which ensures that your next command will be able to utilize the new attribute correctly.

In the following commands:

- Replace *new-uuid* with the UUID you want to set; for example, **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**. You can generate a UUID using the **uuidgen** command.
- Replace *new-label* with a label; for example, **backup_data**.

Prerequisites

- If you are modifying the attributes of an XFS file system, unmount it first.

Procedure

- To change the UUID or Label attributes of an **XFS** file system, use the **xfs_admin** utility:

```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of an **ext4**, **ext3**, or **ext2** file system, use the **tune2fs** utility:

```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of a swap volume, use the **swaponlabel** utility:

```
# swaponlabel --uuid new-uuid --label new-label swap-device
# udevadm settle
```


CHAPTER 5. USING THE WEB CONSOLE FOR MANAGING VIRTUAL DATA OPTIMIZER VOLUMES

This chapter describes the Virtual Data Optimizer (VDO) configuration using the RHEL 8 web console. After reading it, you will be able to:

- Create VDO volumes
- Format VDO volumes
- Extend VDO volumes

5.1. PREREQUISITES

- The RHEL 8 web console is installed and accessible.
For details, see [Installing the web console](#).

5.2. VDO VOLUMES IN THE WEB CONSOLE

Red Hat Enterprise Linux 8 supports Virtual Data Optimizer (VDO). VDO is a block virtualization technology that combines:

Compression

For details, see [Enabling or disabling compression in VDO](#).

Deduplication

For details, see [Enabling or disabling deduplication in VDO](#).

Thin provisioning

For details, see [Thinly-provisioned logical volumes \(thin volumes\)](#).

Using these technologies, VDO:

- Saves storage space inline
- Compresses files
- Eliminates duplications
- Enables you to allocate more virtual space than how much the physical or logical storage provides
- Enables you to extend the virtual storage by growing

VDO can be created on top of many types of storage. In the RHEL 8 web console, you can configure VDO on top of:

- LVM



NOTE

It is not possible to configure VDO on top of thinly-provisioned volumes.

- Physical volume

- Software RAID

For details about placement of VDO in the Storage Stack, see [System Requirements](#).

Additional resources

- For details about VDO, see [Deduplicating and compressing storage](#).

5.3. CREATING VDO VOLUMES IN THE WEB CONSOLE

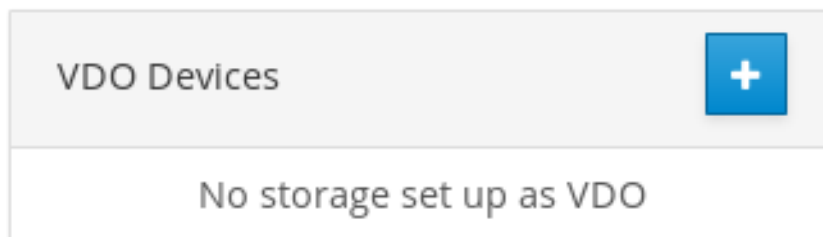
This section helps you to create a VDO volume in the RHEL web console.

Prerequisites

- Physical drives, LVMs, or RAID from which you want to create VDO.

Procedure

1. Log in to the RHEL 8 web console.
For details, see [Logging in to the web console](#).
2. Click **Storage**.
3. Click the + icon in the **VDO Devices** box.



4. In the **Name** field, enter a name of a VDO volume without spaces.
5. Select the drive that you want to use.
6. In the **Logical Size** bar, set up the size of the VDO volume. You can extend it more than ten times, but consider for what purpose you are creating the VDO volume:
 - For active VMs or container storage, use logical size that is ten times the physical size of the volume.
 - For object storage, use logical size that is three times the physical size of the volume.

For details, see [Deploying VDO](#).

7. In the **Index Memory** bar, allocate memory for the VDO volume.
For details about VDO system requirements, see [System Requirements](#).
8. Select the **Compression** option. This option can efficiently reduce various file formats.
For details, see [Enabling or disabling compression in VDO](#).
9. Select the **Deduplication** option.
This option reduces the consumption of storage resources by eliminating multiple copies of duplicate blocks. For details, see [Enabling or disabling deduplication in VDO](#).

10. [Optional] If you want to use the VDO volume with applications that need a 512 bytes block size, select **Use 512 Byte emulation**. This reduces the performance of the VDO volume, but should be very rarely needed. If in doubt, leave it off.
11. Click **Create**.

Create VDO Device

Name

Disk

☒ 5.72 GiB RAID Device 127

/dev/md/127

Logical Size

5.76

GiB

Index Memory

256

MiB

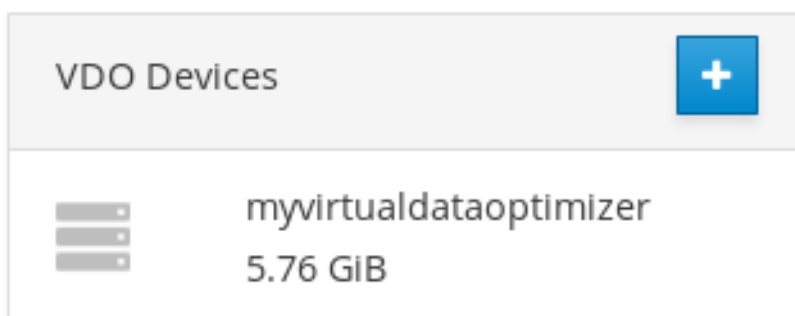
Options

☒ Compression
☒ Deduplication
☐ Use 512 Byte emulation

Cancel

Create

If the process of creating the VDO volume succeeds, you can see the new VDO volume in the **Storage** section and format it with a file system.



5.4. FORMATTING VDO VOLUMES IN THE WEB CONSOLE

VDO volumes act as physical drives. To use them, you need to format them with a file system.



WARNING

Formatting VDO will erase all data on the volume.

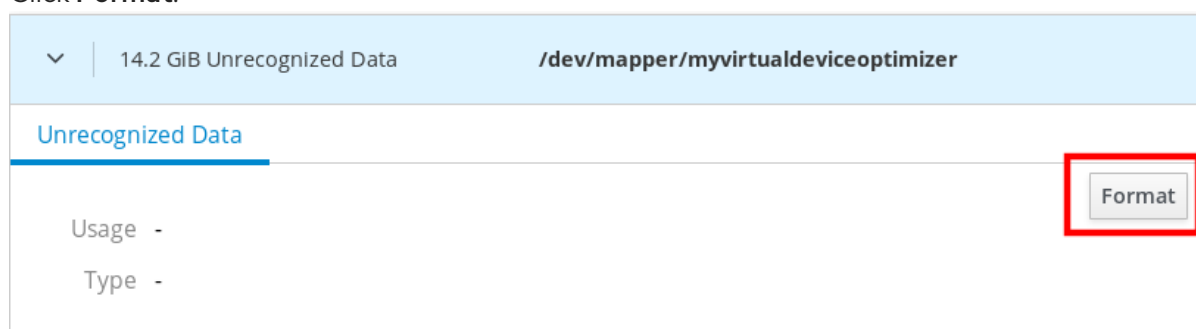
The following steps describe the procedure to format VDO volumes.

Prerequisites

- A VDO volume is created. For details, see [Section 5.3, “Creating VDO volumes in the web console”](#).

Procedure

1. Log in to the RHEL 8 web console.
For details, see [Logging in to the web console](#).
2. Click **Storage**.
3. Click the VDO volume.
4. Click on the **Unrecognized Data** tab.
5. Click **Format**.



6. In the **Erase** drop down menu, select:

Don't overwrite existing data

The RHEL web console rewrites only the disk header. The advantage of this option is the speed of formatting.

Overwrite existing data with zeros

The RHEL web console rewrites the whole disk with zeros. This option is slower because the program has to go through the whole disk. Use this option if the disk includes any data and you need to rewrite them.

7. In the **Type** drop down menu, select a filesystem:

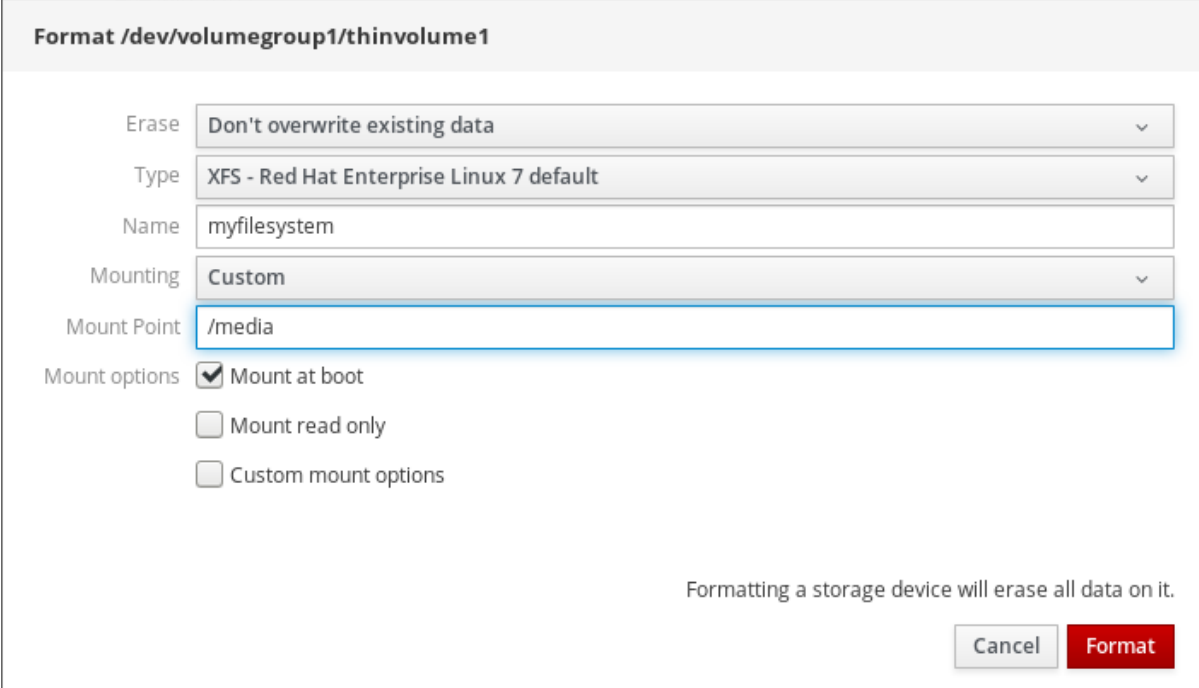
- The **XFS** file system supports large logical volumes, switching physical drives online without outage, and growing. Leave this file system selected if you do not have a different strong preference.
XFS does not support shrinking volumes. Therefore, you will not be able to reduce volume formatted with XFS.
- The **ext4** file system supports logical volumes, switching physical drives online without outage, growing, and shrinking.

You can also select a version with the LUKS (Linux Unified Key Setup) encryption, which allows you to encrypt the volume with a passphrase.

8. In the **Name** field, enter the logical volume name.
9. In the **Mounting** drop down menu, select **Custom**.
The **Default** option does not ensure that the file system will be mounted on the next boot.

10. In the **Mount Point** field, add the mount path.

11. Select **Mount at boot**.



Format /dev/volumegroup1/thinvolume1

Erase: Don't overwrite existing data

Type: XFS - Red Hat Enterprise Linux 7 default

Name: myfilesystem

Mounting: Custom

Mount Point: /media

Mount options: ☒ Mount at boot, ☐ Mount read only, ☐ Custom mount options

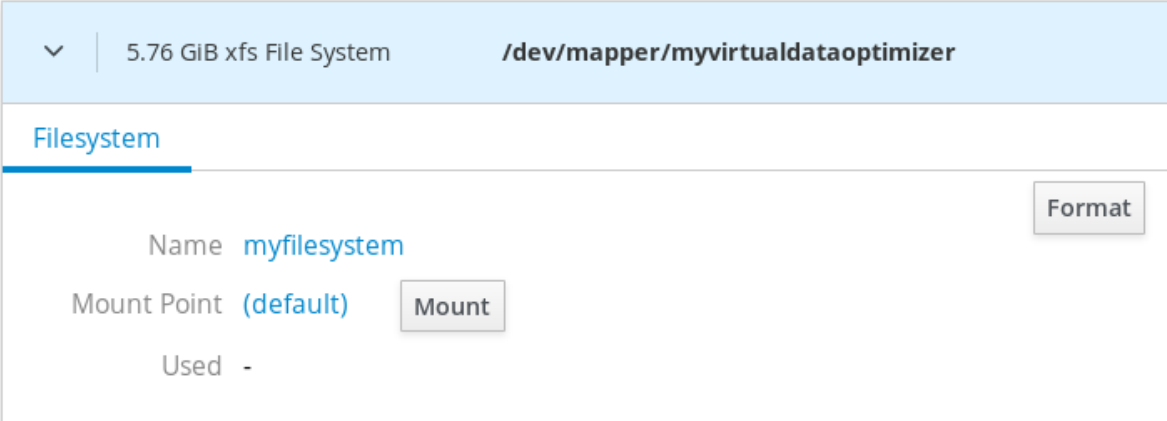
Formatting a storage device will erase all data on it.

Cancel Format

12. Click **Format**.

Formatting can take several minutes depending on the used formatting options and the volume size.

After a successful finish, you can see the details of the formatted VDO volume on the **Filesystem** tab.



5.76 GiB xfs File System /dev/mapper/myvirtualdataoptimizer

Filesystem

Name: myfilesystem

Mount Point: (default) Mount

Used: -

Format

13. To use the VDO volume, click **Mount**.

At this point, the system uses the mounted and formatted VDO volume.

5.5. EXTENDING VDO VOLUMES IN THE WEB CONSOLE

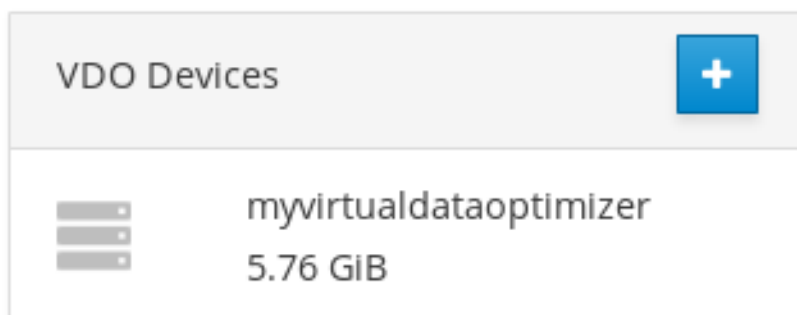
This section describes extending VDO volumes in the RHEL 8 web console.

Prerequisites

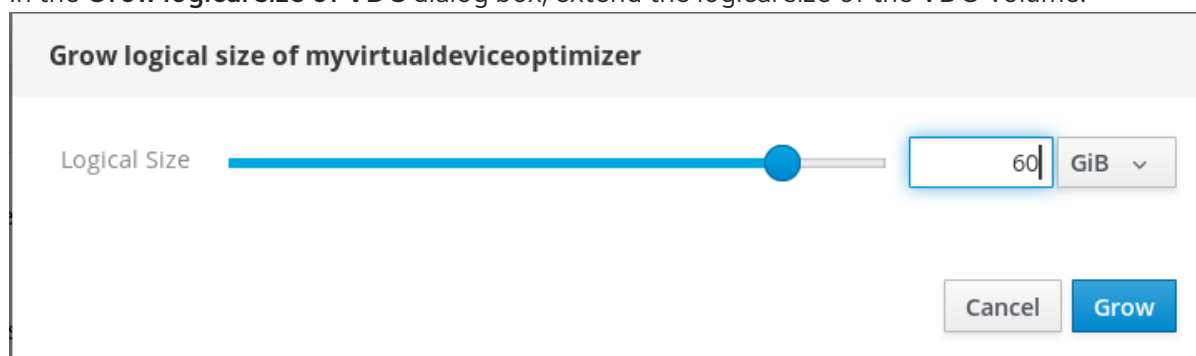
- The VDO volume created.

Procedure

1. Log in to the RHEL 8 web console.
For details, see [Logging in to the web console](#).
2. Click **Storage**.
3. Click your VDO volume in the **VDO Devices** box.



4. In the VDO volume details, click the **Grow** button.
5. In the **Grow logical size of VDO** dialog box, extend the logical size of the VDO volume.



Original size of the logical volume from the screenshot was 6 GB. As you can see, the RHEL web console enables you to grow the volume to more than ten times the size and it works correctly because of the compression and deduplication.

6. Click **Grow**.

If the process of growing VDO succeeds, you can see the new size in the VDO volume details.

VDO Device myvirtualdataoptimizer

Stop

Delete

Device File `/dev/mapper/myvirtualdataoptimizer`Backing Device `/dev/md/127`

Physical 1.11 MiB data + 3.72 GiB overhead used of 5.72 GiB (65%)

Logical 11.7 MiB used of 60 GiB (90% saved)

Grow

Index Memory 256 MiB

Compression

ON

Deduplication

ON