Rocky Enterprise Linux 9.2 Manual Pages on command 'lvmvdo.7'

$ man lvmvdo.7

LVMVDO(7) LVMVDO(7)

NAME

lvmvdo ? Support for Virtual Data Optimizer in LVM

DESCRIPTION

VDO is software that provides inline block-level deduplication, compression, and thin pro?
visioning capabilities for primary storage.

Deduplication is a technique for reducing the consumption of storage resources by elimi?
nating multiple copies of duplicate blocks. Compression takes the individual unique blocks
and shrinks them. These reduced blocks are then efficiently packed together into physical
blocks. Thin provisioning manages the mapping from logical blocks presented by VDO to
where the data has actually been physically stored, and also eliminates any blocks of all
zeroes.

With deduplication, instead of writing the same data more than once, VDO detects and
records each duplicate block as a reference to the original block. VDO maintains a mapping
from Logical Block Addresses (LBA) (used by the storage layer above VDO) to physical block
addresses (used by the storage layer under VDO). After deduplication, multiple logical
block addresses may be mapped to the same physical block address; these are called shared
blocks and are reference-counted by the software.

With compression, VDO compresses multiple blocks (or shared blocks) with the fast LZ4 al?
gorithm, and bins them together where possible so that multiple compressed blocks fit
within a 4 KB block on the underlying storage. Mapping from LBA is to a physical block ad?
dress and index within it for the desired compressed data. All compressed blocks are indi?
vidually reference counted for correctness.

Block sharing and block compression are invisible to applications using the storage, which read and write blocks as they would if VDO were not present. When a shared block is over? written, a new physical block is allocated for storing the new block data to ensure that other logical block addresses that are mapped to the shared physical block are not modi? fied.

To use VDO with lvm(8), you must install the standard VDO user-space tools vdoformat(8) and the currently non-standard kernel VDO module "kvdo".

The "kvdo" module implements fine-grained storage virtualization, thin provisioning, block sharing, and compression. The "uds" module provides memory-efficient duplicate identifi? cation. The user-space tools include vdostats(8) for extracting statistics from VDO vol? umes.

VDO TERMS

VDODataLV

VDO data LV

A large hidden LV with the _vdata suffix. It is created in a VG

used by the VDO kernel target to store all data and metadata blocks.

VDOPoolLV

VDO pool LV

A pool for virtual VDOLV(s), which are the size of used VDODataLV.

Only a single VDOLV is currently supported.

VDOLV

VDO LV

Created from VDOPoolLV.

Appears blank after creation.

VDO USAGE

The primary methods for using VDO with lvm2:

1. Create a VDOPoolLV and a VDOLV

Create a VDOPoolLV that will hold VDO data, and a virtual size VDOLV that the user can use. If you do not specify the virtual size, then the VDOLV is created with the maximum size that always fits into data volume even if no deduplication or compression can happen (i.e. it can hold the incompressible content of /dev/urandom). If you do not specify the name of VDOPoolLV, it is taken from the sequence of vpool0, vpool1 ...

Note: The performance of TRIM/Discard operations is slow for large volumes of VDO type.

Please try to avoid sending discard requests unless necessary because it might take con?
siderable amount of time to finish the discard operation.

lvcreate --type vdo -n VDOLV -L DataSize -V LargeVirtualSize VG/VDOPoolLV

lvcreate --vdo -L DataSize VG

Example

```
# lvcreate --type vdo -n vdo0 -L 10G -V 100G vg/vdopool0
# mkfs.ext4 -E nodiscard /dev/vg/vdo0
```

2. Convert an existing LV into VDOPoolLV

   Convert an already created or existing LV into a VDOPoolLV, which is a volume that can hold data and metadata. You will be prompted to confirm such conversion because it IRRE? VERSIBLY DESTROYS the content of such volume and the volume is immediately formatted by vdoformat(8) as a VDO pool data volume. You can specify the virtual size of the VDOLV as? sociated with this VDOPoolLV. If you do not specify the virtual size, it will be set to the maximum size that can keep 100% incompressible data there.

   lvconvert --type vdo-pool -n VDOLV -V VirtualSize VG/VDOPoolLV

   lvconvert --vdopool VG/VDOPoolLV

   Example

   ```
   # lvconvert --type vdo-pool -n vdo0 -V10G vg/ExistingLV
   ```

3. Change the default settings used for creating a VDOPoolLV

   VDO allows to set a large variety of options. Lots of these settings can be specified in lvm.conf or profile settings. You can prepare a number of different profiles in the /etc/lvm/profile directory and just specify the profile file name. Check the output of lvmconfig --type full for a detailed description of all individual VDO settings.

   Example

   ```
   # cat <<EOF > /etc/lvm/profile/vdo_create.profile
   allocation {
       vdo_use_compression=1
       vdo_use_deduplication=1
       vdo_use_metadata_hints=1
       vdo_minimum_io_size=4096
       vdo_block_map_cache_size_mb=128
       vdo_block_map_period=16380
       vdo_check_point_frequency=0
   ```

vdo_use_sparse_index=0

vdo_index_memory_size_mb=256

vdo_slab_size_mb=2048

vdo_ack_threads=1

vdo_bio_threads=1

vdo_bio_rotation=64

vdo_cpu_threads=2

vdo_hash_zone_threads=1

vdo_logical_threads=1

vdo_physical_threads=1

vdo_write_policy="auto"

vdo_max_discard=1

}

EOF

lvcreate --vdo -L10G --metadataprofile vdo_create vg/vdopool0

lvcreate --vdo -L10G --config 'allocation/vdo_cpu_threads=4' vg/vdopool1

4. Change the compression and deduplication of a VDOPoolLV

   Disable or enable the compression and deduplication for VDOPoolLV (the volume that main?

   tains all VDO LV(s) associated with it).

   lvchange --compression [y|n] --deduplication [y|n] VG/VDOPoolLV

   Example

   # lvchange --compression n vg/vdopool0

   # lvchange --deduplication y vg/vdopool1

5. Checking the usage of VDOPoolLV

   To quickly check how much data on a VDOPoolLV is already consumed, use lvs(8). The Data%

   field reports how much data is occupied in the content of the virtual data for the VDOLV

   and how much space is already consumed with all the data and metadata blocks in the

   VDOPoolLV. For a detailed description, use the vdostats(8) command.

   Note: vdostats(8) currently understands only /dev/mapper device names.

   Example

   # lvcreate --type vdo -L10G -V20G -n vdo0 vg/vdopool0

   # mkfs.ext4 -E nodiscard /dev/vg/vdo0

   # lvs -a vg

```
  LV               VG Attr       LSize  Pool     Origin Data%
  vdo0             vg vwi-a-v--- 20.00g vdopool0        0.01
  vdopool0         vg dwi-ao---- 10.00g                30.16
  [vdopool0_vdata] vg Dwi-ao---- 10.00g
# vdostats --all /dev/mapper/vg-vdopool0-vpool
/dev/mapper/vg-vdopool0 :
  version                            : 30
  release version                    : 133524
  data blocks used                   : 79
  ...
```

6. Extending the VDOPoolLV size

You can add more space to hold VDO data and metadata by extending the VDODataLV using the commands lvresize(8) and lvextend(8). The extension needs to add at least one new VDO slab. You can configure the slab size with the allocation/vdo_slab_size_mb setting.

You can also enable automatic size extension of a monitored VDOPoolLV with the activa?tion/vdo_pool_autoextend_percent and activation/vdo_pool_autoextend_threshold settings.

Note: You cannot reduce the size of a VDOPoolLV.

Note: You cannot change the size of a cached VDOPoolLV.

lvextend -L+AddingSize VG/VDOPoolLV

Example

```
# lvextend -L+50G vg/vdopool0
# lvresize -L300G vg/vdopool1
```

7. Extending or reducing the VDOLV size

You can extend or reduce a virtual VDO LV as a standard LV with the lvresize(8), lvex?tend(8), and lvreduce(8) commands.

Note: The reduction needs to process TRIM for reduced disk area to unmap used data blocks from the VDOPoolLV, which might take a long time.

lvextend -L+AddingSize VG/VDOLV

lvreduce -L-ReducingSize VG/VDOLV

Example

```
# lvextend -L+50G vg/vdo0
# lvreduce -L-50G vg/vdo1
# lvresize -L200G vg/vdo2
```

8. Component activation of a VDODataLV

   You can activate a VDODataLV separately as a component LV for examination purposes. The activation of the VDODataLV activates the data LV in read-only mode, and the data LV can? not be modified. If the VDODataLV is active as a component, any upper LV using this vol? ume CANNOT be activated. You have to deactivate the VDODataLV first to continue to use the VDOPoolLV.

   Example

   # lvchange -ay vg/vpool0_vdata

   # lvchange -an vg/vpool0_vdata

VDO TOPICS

1. Stacking VDO

   You can convert or stack a VDOPooLV with these currently supported volume types: linear, stripe, raid, and cache with cachepool.

2. VDOPoolLV on top of raid

   Using a raid type LV for a VDODataLV.

   Example

   # lvcreate --type raid1 -L 5G -n vdopool vg

   # lvconvert --type vdo-pool -V 10G vg/vdopool

3. Caching a VDODataLV or a VDOPoolLV

   VDODataLV (accepts also VDOPoolLV) caching provides a mechanism to accelerate reads and writes of already compressed and deduplicated data blocks together with VDO metadata.

   A cached VDO data LV cannot be currently resized. Also, the threshold based automatic re? size will not work.

   Example

   # lvcreate --type vdo -L 5G -V 10G -n vdo1 vg/vdopool

   # lvcreate --type cache-pool -L 1G -n cachepool vg

   # lvconvert --cache --cachepool vg/cachepool vg/vdopool

   # lvconvert --uncache vg/vdopool

4. Caching a VDOLV

   VDO LV cache allow you to 'cache' a device for better performance before it hits the pro? cessing of the VDO Pool LV layer.

   Example

   # lvcreate --type vdo -L 5G -V 10G -n vdo1 vg/vdopool

lvcreate --type cache-pool -L 1G -n cachepool vg

lvconvert --cache --cachepool vg/cachepool vg/vdo1

lvconvert --uncache vg/vdo1

5. Usage of Discard/TRIM with a VDOLV

   You can discard data on a VDO LV and reduce used blocks on a VDOPoolLV. However, the cur?
   rent performance of discard operations is still not optimal and takes a considerable
   amount of time and CPU. Unless you really need it, you should avoid using discard.

   When a block device is going to be rewritten, its blocks will be automatically reused for
   new data. Discard is useful in situations when user knows that the given portion of a VDO
   LV is not going to be used and the discarded space can be used for block provisioning in
   other regions of the VDO LV. For the same reason, you should avoid using mkfs with dis?
   card for a freshly created VDO LV to save a lot of time that this operation would take
   otherwise as device is already expected to be empty.

6. Memory usage

   The VDO target requires 370 MiB of RAM plus an additional 268 MiB per each 1 TiB of physi?
   cal storage managed by the volume.

   UDS requires a minimum of 250 MiB of RAM, which is also the default amount that deduplica?
   tion uses.

   The memory required for the UDS index is determined by the index type and the required
   size of the deduplication window and is controlled by the allocation/vdo_use_sparse_index
   setting.

   With enabled UDS sparse indexing, it relies on the temporal locality of data and attempts
   to retain only the most relevant index entries in memory and 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.

   A dense index with 1 GiB of RAM maintains a 1 TiB deduplication window, while a sparse in?
   dex with 1 GiB of RAM maintains a 10 TiB deduplication window. In general, 1 GiB is suf?
   ficient for 4 TiB of physical space with a dense index and 40 TiB with a sparse index.

7. Storage space requirements

   You can configure a VDOPoolLV to use up to 256 TiB of physical storage. Only a certain
   part of the physical storage is usable to store data. This section provides the calcula?

tions to determine the usable size of a VDO-managed volume.

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

? The first type of VDO metadata uses approximately 1 MiB for each 4 GiB of physical storage plus an additional 1 MiB per slab.

? The second type of VDO metadata consumes approximately 1.25 MiB for each 1 GiB 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 GiB of RAM, a dense UDS index uses 17 GiB of storage and a sparse UDS index will use 170 GiB of storage.

SEE ALSO

lvm(8), lvm.conf(5), lvmconfig(8), lvcreate(8), lvconvert(8), lvchange(8), lvextend(8), lvreduce(8), lvresize(8), lvremove(8), lvs(8), vdo(8), vdoformat(8), vdostats(8), mkfs(8)