# Storage Exercises

Note: we have no extra disks to use for these exercises, so we will have to make do with using

1) Imitation disks which work only for labelling purposes

2) Logical volumes – which at least are genuine

The system (root filesystem and all) is supported by disk /dev/sda

The whole of /dev/sdb is taken up with an azure filesystem mounted on /mnt/resource. Please leave this disk alone. It is not ours!

Just to make sure we have a good starting point, run the following scripts:

|  |
| --- |
| # /scenariolabs/Storage/resetStorageToOriginal  # /scenariolabs/Storage/initialiseStorage |

The inititialiseStorage script will take about one minute to run.

## Using gdisk to create slices.

First let us create our imitation disk:

|  |
| --- |
| dd of=/dev/sdc if=/dev/zero bs=500M count=1 |

Now let to use gdisk

|  |
| --- |
| gdisk /dev/sdc  GPT fdisk (gdisk) version 0.8.10    Partition table scan:  MBR: not present  BSD: not present  APM: not present  GPT: not present    Creating new GPT entries. |

– print the current table: p

|  |
| --- |
| Command (? for help): p  Disk /dev/sdc: 1024000 sectors, 500.0 MiB  Logical sector size: 512 bytes  Disk identifier (GUID): F9AEABCA-8CEC-45D0-B34A-9F8B521F7B9E  Partition table holds up to 128 entries  First usable sector is 34, last usable sector is 1023966  Partitions will be aligned on 2048-sector boundaries  Total free space is 1023933 sectors (500.0 MiB)    Number Start (sector) End (sector) Size Code Name    Command (? for help): |

Create slice 1: n and accept slice 1 as the default.

|  |
| --- |
| Command (? for help): n  Partition number (1-128, default 1):  First sector (34-1023966, default = 2048) or {+-}size{KMGTP}: |

Accept the suggested first sector

|  |
| --- |
| Command (? for help): n  Partition number (1-128, default 1):  First sector (34-1023966, default = 2048) or {+-}size{KMGTP}:  Last sector (2048-1023966, default = 1023966) or {+-}size{KMGTP}: |

Type in 50M for the size

|  |
| --- |
| Command (? for help): n  Partition number (1-128, default 1):  First sector (34-1023966, default = 2048) or {+-}size{KMGTP}:  Last sector (2048-1023966, default = 1023966) or {+-}size{KMGTP}: 50M  Current type is 'Linux filesystem'  Hex code or GUID (L to show codes, Enter = 8300): |

Accept the default code 8300, which indicates the slice is to be used to house a filesystem.

|  |
| --- |
| Command (? for help): n  Partition number (1-128, default 1):  First sector (34-1023966, default = 2048) or {+-}size{KMGTP}:  Last sector (2048-1023966, default = 1023966) or {+-}size{KMGTP}: 50M  Current type is 'Linux filesystem'  Hex code or GUID (L to show codes, Enter = 8300):  Changed type of partition to 'Linux filesystem'    Command (? for help): |

Print to confirm the details are right

|  |
| --- |
| Command (? for help): p  Disk /dev/sdc: 1024000 sectors, 500.0 MiB  Logical sector size: 512 bytes  Disk identifier (GUID): 3FF8C55B-21CF-445D-A199-2139C88B78B0  Partition table holds up to 128 entries  First usable sector is 34, last usable sector is 1023966  Partitions will be aligned on 2048-sector boundaries  Total free space is 923580 sectors (451.0 MiB)    Number Start (sector) End (sector) Size Code Name  1 2048 102400 49.0 MiB 8300 Linux filesystem    Command (? for help): |

Create partition 2. type n:

|  |
| --- |
| Command (? for help): n  Partition number (2-128, default 2): |

Accept the suggested partition number and the suggested first sector.

|  |
| --- |
| Command (? for help): n  Partition number (2-128, default 2):  First sector (34-1023966, default = 104448) or {+-}size{KMGTP}:  Last sector (104448-1023966, default = 1023966) or {+-}size{KMGTP}: |

Type **100M**

|  |
| --- |
| Command (? for help): n  Partition number (2-128, default 2):  First sector (34-1023966, default = 104448) or {+-}size{KMGTP}:  Last sector (104448-1023966, default = 1023966) or {+-}size{KMGTP}: 100M  Current type is 'Linux filesystem'  Hex code or GUID (L to show codes, Enter = 8300): |

Accept the code as before and print the table to check the details.

|  |
| --- |
| Command (? for help): n  Partition number (2-128, default 2):  First sector (34-1023966, default = 104448) or {+-}size{KMGTP}:  Last sector (104448-1023966, default = 1023966) or {+-}size{KMGTP}: 100M  Current type is 'Linux filesystem'  Hex code or GUID (L to show codes, Enter = 8300):  Changed type of partition to 'Linux filesystem'    Command (? for help): p  Disk /dev/sdc: 1024000 sectors, 500.0 MiB  Logical sector size: 512 bytes  Disk identifier (GUID): 3FF8C55B-21CF-445D-A199-2139C88B78B0  Partition table holds up to 128 entries  First usable sector is 34, last usable sector is 1023966  Partitions will be aligned on 2048-sector boundaries  Total free space is 823227 sectors (402.0 MiB)    Number Start (sector) End (sector) Size Code Name  1 2048 102400 49.0 MiB 8300 Linux filesystem  2 104448 204800 49.0 MiB 8300 Linux filesystem |

Write and quit.

|  |
| --- |
| umber Start (sector) End (sector) Size Code Name  1 2048 102400 49.0 MiB 8300 Linux filesystem  2 104448 204800 49.0 MiB 8300 Linux filesystem    Command (? for help): w    Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING  PARTITIONS!!    Do you want to proceed? (Y/N): y  OK; writing new GUID partition table (GPT) to /dev/sdc.  Warning: The kernel is still using the old partition table.  The new table will be used at the next reboot.  The operation has completed successfully.  [root@ML-RefVm-198965 Storage]#partprobe |

The partprobe command causes the kernel to refresh its in memory copy of the label.

**Using gdisk to delete a partition**

|  |
| --- |
| # gdisk /dev/sdc  GPT fdisk (gdisk) version 0.8.10    Partition table scan:  MBR: protective  BSD: not present  APM: not present  GPT: present    Found valid GPT with protective MBR; using GPT.    Command (? for help): |

P to show the current partition table

|  |
| --- |
| Command (? for help): p  Disk /dev/sdc: 1024000 sectors, 500.0 MiB  Logical sector size: 512 bytes  Disk identifier (GUID): 8F78BB6B-58B6-4304-9FC8-9F18651E3E63  Partition table holds up to 128 entries  First usable sector is 34, last usable sector is 1023966  Partitions will be aligned on 2048-sector boundaries  Total free space is 823227 sectors (402.0 MiB)    Number Start (sector) End (sector) Size Code Name  1 2048 102400 49.0 MiB 8300 Linux filesystem  2 104448 204800 49.0 MiB 8300 Linux filesystem    Command (? for help): |

D to delete

|  |
| --- |
| Command (? for help): d  Partition number (1-2): |

Select partition 2 then p to print

|  |
| --- |
| Partition number (1-2): 2    Command (? for help): p  Disk /dev/sdc: 1024000 sectors, 500.0 MiB  Logical sector size: 512 bytes  Disk identifier (GUID): 8F78BB6B-58B6-4304-9FC8-9F18651E3E63  Partition table holds up to 128 entries  First usable sector is 34, last usable sector is 1023966  Partitions will be aligned on 2048-sector boundaries  Total free space is 923580 sectors (451.0 MiB)    Number Start (sector) End (sector) Size Code Name  1 2048 102400 49.0 MiB 8300 Linux filesystem |

W – write and quit

|  |
| --- |
| Command (? for help): w    Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING  PARTITIONS!!    Do you want to proceed? (Y/N): Y  OK; writing new GUID partition table (GPT) to /dev/sdc.  Warning: The kernel is still using the old partition table.  The new table will be used at the next reboot.  The operation has completed successfully.  [root@ML-RefVm-198965 Storage]# |

### Creating and using logical volumes.

Logical volumes act like disk slices. In sophisticated setups these volumes might be mirrored to allow them to have more resilience than normal disk slices. They might be striped to give better performance than a disk slice might do. Here they are simple logical volumes which at least have the property of being growable, which disk slices, alas, are not.

Every logical volume belongs to a Volume Group. Volume Groups also own physical disk space which is used to support the (often complex) storage demands of the volumes. Our volume group, rootvg, is already in place.

|  |
| --- |
| # lvcreate -L 100M -n smallVol rootvg  Logical volume "smallVol" created. |

This creates a volume called “smalVoll”, its only 100 Mebibyte big.

Now we create a filesystem within the volume.

|  |
| --- |
| # mkfs.xfs /dev/rootvg/smallVol  meta-data=/dev/rootvg/smallVol isize=512 agcount=4, agsize=6400 blks  = sectsz=512 attr=2, projid32bit=1  = crc=1 finobt=0, sparse=0  data = bsize=4096 blocks=25600, imaxpct=25  = sunit=0 swidth=0 blks  naming =version 2 bsize=4096 ascii-ci=0 ftype=1  log =internal log bsize=4096 blocks=855, version=2  = sectsz=512 sunit=0 blks, lazy-count=1  realtime =none extsz=4096 blocks=0, rtextents=0  [root@ML-RefVm-198965 Storage]# |

Making a mount point for the new filesystem

|  |
| --- |
| # mkdir /data |

Mount the filesystem – the device file holding the filesystem is /dev/rootvg/smallVol

|  |
| --- |
| # mount –t xfs /dev/rootvg/smallVol /data |

The filesystem is empty

|  |
| --- |
| # df -h /data  Filesystem Size Used Avail Use% Mounted on  /dev/mapper/rootvg-smallVol 97M 5.3M 92M 6% /data |

Adding a little data in the form of a large file

|  |
| --- |
| # dd if=/dev/zero of=/data/large bs=80M count=1  1+0 records in  1+0 records out  83886080 bytes (84 MB) copied, 0.117301 s, 715 MB/s |

The filesystem is quite full now.

|  |
| --- |
| ]# df -h /data  Filesystem Size Used Avail Use% Mounted on  /dev/mapper/rootvg-smallVol 97M 86M 12M 89% /data |

### Growing a filesystem

First we need to grow the logical volume using the lvextend command,.

|  |
| --- |
| lvextend -L +500M /dev/rootvg/smallVol  Size of logical volume rootvg/smallVol changed from 100.00 MiB (25 extents) to 600.00 MiB (150 extents).  Logical volume rootvg/smallVol successfully resized. |

Note this has not had any effect on the filesystem.

|  |
| --- |
| # df -h /data  Filesystem Size Used Avail Use% Mounted on  /dev/mapper/rootvg-smallVol 97M 86M 12M 89% /data |

We need to grow the filesystem structure into the new space.

|  |
| --- |
| # xfs\_growfs /dev/rootvg/smallVol  meta-data=/dev/mapper/rootvg-smallVol isize=512 agcount=4, agsize=6400 blks  = sectsz=512 attr=2, projid32bit=1  = crc=1 finobt=0 spinodes=0  data = bsize=4096 blocks=25600, imaxpct=25  = sunit=0 swidth=0 blks  naming =version 2 bsize=4096 ascii-ci=0 ftype=1  log =internal bsize=4096 blocks=855, version=2  = sectsz=512 sunit=0 blks, lazy-count=1  realtime =none extsz=4096 blocks=0, rtextents=0  data blocks changed from 25600 to 153600 |

This has quite an impact on the filesystem usage.

|  |
| --- |
| # df -h /data  Filesystem Size Used Avail Use% Mounted on  /dev/mapper/rootvg-smallVol 597M 87M 511M 15% /data |

## Repairing a filesystem

Filesystems are intricate on-disk structures. Disks are slow to update, consequently a lot of filesystem information is stored in memory (cache) and updated there, only later to be flushed to disk. This means that, at any given moment, the filesystem structures exist partly in memory and partly on disk. The CPU looks at the disk through the memory caches and sees an integral structure. However, if the system suddenly crashes the memory-based information disappears. When the system recovers, it is left only with the disk-based information which as its fragile set of pointers and cross references in smithereens. A repair process is needed to re-integrate the filesystem, putting back pointers and counters and such like to obtain a system that is usable again.

This exercise demonstrates filesystem repair. To do this we damage the filesystem using something called a ‘phantom write’, that is a write into the filesystem that take places outside the auspices of the official filesystem updates. For example, a third system may somehow have access to SAN based disks and accidentally write to the wrong disk.

Part of the exercise is to demonstrate the use of the lost+found directory. This directory is used to house orphaned inodes. The repair process reconnects them by putting them in lost and found. A file’s name is stored in as a directory entry. This entry identifies an associated inode by its number. If a directory is lost, all the filenames are gone but the inodes may well still be healthy and working. Should they be reconnected in lost+found, they are given a new name which is simply taken from the inode number. In order to help make sense of the offering lost+found has to make, it is prudent to have first taken a note of which file name is associated with which inode, and store it separately from the filesystem. On a real system, this simple task could be done in a cron job.

### Loading the filesystem with data and creating an inode index.

Using the /data filesystem we created earlier:

|  |
| --- |
| cp -r /var/log/\* /data |

How full is it now?

|  |
| --- |
| df -h /data  Filesystem Size Used Avail Use% Mounted on  /dev/mapper/rootvg-smallVol 597M 370M 228M 62% /data |

Create the inode index file for later use.

|  |
| --- |
| find /data -exec ls -id {} \; > /scenariolabs/Storage/inodeLog |

### Damaging the filesystem!

|  |
| --- |
| umount /data  dd if=/dev/zero of=/dev/rootvg/smallVol bs=1M skip=1 count=1  1+0 records in  1+0 records out  1048576 bytes (1.0 MB) copied, 0.00156312 s, 671 MB/s |

### Repairing the filesystem! (Output abbreviated)

|  |
| --- |
| xfs\_repair /dev/rootvg/smallVol  Phase 1 - find and verify superblock...  bad primary superblock - bad magic number !!!    attempting to find secondary superblock...  ..........found candidate secondary superblock...  verified secondary superblock...  writing modified primary superblock  - reporting progress in intervals of 15 minutes  Phase 2 - using internal log  - zero log...  - scan filesystem freespace and inode maps...  Metadata CRC error detected at xfs\_agf block 0x1/0x200  Metadata CRC error detected at xfs\_agi block 0x2/0x200  bad magic # 0x0 for agf 0  ...  agi unlinked bucket 0 is 0 in ag 0 (inode=0)  agi unlinked bucket 1 is 0 in ag 0 (inode=0)  agi unlinked bucket 2 is 0 in ag 0 (inode=0)  agi unlinked bucket 3 is 0 in ag 0 (inode=0)  ...  Metadata corruption detected at xfs\_inode block 0x40/0x2000  Metadata corruption detected at xfs\_inode block 0x40/0x2000  M...  bad CRC for inode 64  bad magic number 0x0 on inode 64  bad version number 0x0 on inode 64  ...  bad magic number 0x0 on inode 127  bad version number 0x0 on inode 127  inode identifier 0 mismatch on inode 127  bad CRC for inode 64, will rewrite  bad magic number 0x0 on inode 64, resetting magic number  bad version number 0x0 on inode 64, resetting version number  inode identifier 0 mismatch on inode 64  cleared root inode 64  ...  resetting inode 1114185 nlinks from 31 to 29  resetting inode 1245454 nlinks from 22 to 21  resetting inode 1310822 nlinks from 6 to 5  resetting inode 1376324 nlinks from 5 to 4  resetting inode 1441857 nlinks from 7 to 6  resetting inode 1441896 nlinks from 18 to 17  resetting inode 1441907 nlinks from 32 to 30  resetting inode 1442118 nlinks from 4 to 3  resetting inode 1507399 nlinks from 5 to 4  resetting inode 1507406 nlinks from 32 to 30  - 09:11:59: verify and correct link counts - 24 of 24 allocation groups done  Note - stripe unit (0) and width (0) were copied from a backup superblock.  Please reset with mount -o sunit=<value>,swidth=<value> if necessary  Metadata corruption detected at xfs\_dir3\_block block 0xfa088/0x1000  libxfs\_writebufr: write verifer failed on xfs\_dir3\_block bno 0xfa088/0x1000  Metadata corruption detected at xfs\_dir3\_block block 0x3e8c0/0x1000  libxfs\_writebufr: write verifer failed on xfs\_dir3\_block bno 0x3e8c0/0x1000  Metadata corruption detected at xfs\_dir3\_block block 0x25880/0x1000  libxfs\_writebufr: write verifer failed on xfs\_dir3\_block bno 0x25880/0x1000  releasing dirty buffer (bulk) to free list!releasing dirty buffer (bulk) to free list!releasing dirty buffer (bulk) to free list!done |

### Mounting the filesystem and checking lost+found.

|  |
| --- |
| mount -t xfs /dev/rootvg/smallVol /data  ls /data/lost+found  1048847 1114474 1245475 1376567 137984 137994 138254 1507645 196950 262472 328010 393287 393524 458820 459062 524614 590173 655656 65601 65608 65898 65915 786506 852042 917574 983391  1114185 1179955 1311055 1376577 137992 138242 1442119 196678 262215 327748 393285 393514 393532 458822 459072 590143 655432 65600 65604 65632 65914 721203 786763 852277 917800 |

Here we can see inodes, which themselves are healthy but whose directories have been deleted. The question is, what was their original name? This we can find out in the inode log we created earlier. Note, actual inode numbers may vary. Follow the exercise using the ones you are given.

|  |
| --- |
| grep '^1048847' /scenariolabs/Storage/inodeLog  1048847 /data/lib/yum/yumdb/d/7b1da4da6affa8135a3c294886b7d86115f2e666-dbus-libs-1.10.24-13.el7\_6-x86\_64 |

We can restore it back to its original position. First make sure the parent directory is in place.

|  |
| --- |
| mkdir -p /data/lib/yum/yumdb/d |

Now we can simply move it back into place

|  |
| --- |
| mv /data/lost+found/1048847 /data/lib/yum/yumdb/d/7b1da4da6affa8135a3c294886b7d86115f2e666-dbus-libs-1.10.24-13.el7\_6-x86\_64 |

## More Work with Logical Volumes

Creating Physical Volume

Physical Volumes are just disks that have been assigned to the Volume manager. We do not have real disks to play with, but we have three loopback devices. These are just imitation disks, ie files that have been setup to look like real devices. They are called

/dev/loop0

/dev/loop1

/dev/loop2

Add them LVM

|  |
| --- |
| # pvcreate /dev/loop0  Physical volume "/dev/loop0" successfully created.  # pvcreate /dev/loop1  Physical volume "/dev/loop1" successfully created.  # pvcreate /dev/loop2  Physical volume "/dev/loop2" successfully created. |

List the Physical Volume’s names.

|  |
| --- |
| # pvdisplay | grep 'PV Name'  PV Name /dev/sda2  PV Name /dev/loop0  PV Name /dev/loop2  PV Name /dev/loop1 |

Look at the attributes of Physical Volume /dev/loop0

|  |
| --- |
| pvdisplay /dev/loop0  "/dev/loop0" is a new physical volume of "1.00 GiB"  --- NEW Physical volume ---  PV Name /dev/loop0  VG Name  PV Size 1.00 GiB  Allocatable NO  PE Size 0  Total PE 0  Free PE 0  Allocated PE 0  PV UUID CjiRlA-EeN1-CH13-tXei-Ug38-YDlq-BYjapr |

Now create a Volume Group out of all /dev/loop0 and /dev/loop1. Claa the Volume Group ‘loopvg’

|  |
| --- |
| # vgcreate loopvg /dev/loop0 /dev/loop1  Volume group "loopvg" successfully created |

Show the Volume Group attributes.

|  |
| --- |
| # vgdisplay loopvg  --- Volume group ---  VG Name loopvg  System ID  Format lvm2  Metadata Areas 2  Metadata Sequence No 1  VG Access read/write  VG Status resizable  MAX LV 0  Cur LV 0  Open LV 0  Max PV 0  Cur PV 2  Act PV 2  VG Size 1.99 GiB  PE Size 4.00 MiB  Total PE 510  Alloc PE / Size 0 / 0  Free PE / Size 510 / 1.99 GiB  VG UUID 34XsgX-3bTP-Argk-gQMf-XEru-zw3p-oWdfhM |

Note the PE size. Every Volume Group lays out its disks with Physical Extents.

Look at the attributes of Physical Volume /dev/loop0 again

|  |
| --- |
| # pvdisplay /dev/loop0  --- Physical volume ---  PV Name /dev/loop0  VG Name loopvg  PV Size 1.00 GiB / not usable 4.00 MiB  Allocatable yes  PE Size 4.00 MiB  Total PE 255  Free PE 255  Allocated PE 0  PV UUID CjiRlA-EeN1-CH13-tXei-Ug38-YDlq-BYjapr |

The Volume (disk) has been laid out with 255 Physical Extents, which are like disk slices. These are the units of allocatable space.

Create a 100M logical Volume call ‘catvol’. Catvol, is short for concatenated volume, which this volume will eventually become. It should take its space from loopvg.

|  |
| --- |
| # lvcreate -n catvol -L 100M loopvg  Logical volume "catvol" created. |

Display catvol’s attributes

|  |
| --- |
| # lvdisplay loopvg/catvol  --- Logical volume ---  LV Path /dev/loopvg/catvol  LV Name catvol  VG Name loopvg  LV UUID 7DHKCu-hwNA-SJSf-hBra-itFg-y1tP-6PliWC  LV Write Access read/write  LV Creation host, time ML-RefVm-198965, 2020-06-17 10:32:34 +0000  LV Status available  # open 0  LV Size 100.00 MiB  Current LE 25  Segments 1  Allocation inherit  Read ahead sectors auto  - currently set to 8192  Block device 253:6 |

Note it has 25 Logical Extents.

Use the –m option to lvdisplay to see a map of this Logical Volume’s extents

|  |
| --- |
| # lvdisplay -m loopvg/catvol  --- Logical volume ---  LV Path /dev/loopvg/catvol  LV Name catvol  VG Name loopvg  LV UUID 7DHKCu-hwNA-SJSf-hBra-itFg-y1tP-6PliWC  LV Write Access read/write  LV Creation host, time ML-RefVm-198965, 2020-06-17 10:32:34 +0000  LV Status available  # open 0  LV Size 100.00 MiB  Current LE 25  Segments 1  Allocation inherit  Read ahead sectors auto  - currently set to 8192  Block device 253:6    --- Segments ---  Logical extents 0 to 24:  Type linear  Physical volume /dev/loop0  Physical extents 0 to 24 |

This volume is using PE’s 0 to 25 on PV /dev/loop0

Create a 100M Logical Volume called simpvol.

|  |
| --- |
| # lvcreate -n simpvol -L 100M loopvg  Logical volume "simpvol" created. |

Show simpvol’s attributers including the map.

|  |
| --- |
| # lvdisplay -m loopvg/simpvol  --- Logical volume ---  LV Path /dev/loopvg/simpvol  LV Name simpvol  VG Name loopvg  LV UUID 6VfUr1-ngnZ-43h6-W2R2-t0Qh-dXqN-dtYOIn  LV Write Access read/write  LV Creation host, time ML-RefVm-198965, 2020-06-17 10:37:21 +0000  LV Status available  # open 0  LV Size 100.00 MiB  Current LE 25  Segments 1  Allocation inherit  Read ahead sectors auto  - currently set to 8192  Block device 253:7    --- Segments ---  Logical extents 0 to 24:  Type linear  Physical volume /dev/loop0  Physical extents 25 to 49 |

Simpvol has just taken the next 25 PEs from /dev/loop0

Grow catvol by 50M

|  |
| --- |
| # lvextend -L +50M loopvg/catvol  Rounding size to boundary between physical extents: 52.00 MiB.  Size of logical volume loopvg/catvol changed from 100.00 MiB (25 extents) to 152.00 MiB (38 extents).  Logical volume loopvg/catvol successfully resized. |

Show its map

|  |
| --- |
| # lvdisplay -m loopvg/catvol  <output abbreviated>  --- Segments ---  Logical extents 0 to 24:  Type linear  Physical volume /dev/loop0  Physical extents 0 to 24    Logical extents *25 to 37:*  Type linear  Physical volume /dev/loop0  Physical extents 50 to 62 |

Note that catvol has taken the next available PEs from /dev/loop0. This means that catvol is no longer contiguous. It takes space from two distinct areas of disk. These areas ‘add together’ or are concatenated.

Delete catvol and simpvol

|  |
| --- |
| # lvremove loopvg/catvol loopvg/simpvol  Do you really want to remove active logical volume loopvg/catvol? [y/n]: y  Logical volume "catvol" successfully removed  Do you really want to remove active logical volume loopvg/simpvol? [y/n]: y  Logical volume "simpvol" successfully removed |

Add a two-way striped volume called zebravol

|  |
| --- |
| # lvcreate -i 2 -L 100M -n zebravol loopvg  Using default stripesize 64.00 KiB.  Rounding size 100.00 MiB (25 extents) up to stripe boundary size 104.00 MiB(26 extents).  Logical volume "zebravol" created. |

Looks at zebravol’s map

|  |
| --- |
| ]# lvdisplay -m loopvg/zebravol  --- Logical volume ---  LV Path /dev/loopvg/zebravol  LV Name zebravol  VG Name loopvg  LV UUID s6y2lE-Q3cr-TPJ1-15zZ-K2U6-l9xz-CeI2VO  LV Write Access read/write  LV Creation host, time ML-RefVm-198965, 2020-06-17 10:46:40 +0000  LV Status available  # open 0  LV Size 104.00 MiB  Current LE 26  Segments 1  Allocation inherit  Read ahead sectors auto  - currently set to 8192  Block device 253:6    --- Segments ---  Logical extents 0 to 25:  Type striped  Stripes 2  Stripe size 64.00 KiB  Stripe 0:  Physical volume /dev/loop0  Physical extents 0 to 12  Stripe 1:  Physical volume /dev/loop1  Physical extents 0 to 12 |

Note that the volume uses two disks and takes 13 PEs from each. Data will be written to both disks more or less simultaneously. The first 64KiB (Kibibyte) will be written to /dev/loop0 and whilst that is happening the second 64KiB will be written to /dev/loop1.

Delete zebravol and create ‘wallvol’ in loopvg, 100M in size but mirrored.

|  |
| --- |
| # lvremove loopvg/zebravol  Do you really want to remove active logical volume loopvg/zebravol? [y/n]: y  Logical volume "zebravol" successfully removed  # lvcreate -n wallvol -L 100M -m1 loopvg  Logical volume "wallvol" created. |

Note the –m option determines the number of *additional* copies of the data you want.

Look at wallvol’s map.

|  |
| --- |
| # lvdisplay -m loopvg/wallvol  --- Logical volume ---  LV Path /dev/loopvg/wallvol  LV Name wallvol  VG Name loopvg  LV UUID QSehNP-OUjQ-ucDB-x0Eg-cF4f-W30R-0UZdqp  LV Write Access read/write  LV Creation host, time ML-RefVm-198965, 2020-06-17 10:55:39 +0000  LV Status available  # open 0  LV Size 100.00 MiB  Current LE 25  Mirrored volumes 2  Segments 1  Allocation inherit  Read ahead sectors auto  - currently set to 8192  Block device 253:10    --- Segments ---  Logical extents 0 to 24:  Type raid1  Monitoring monitored  Raid Data LV 0  Logical volume wallvol\_rimage\_0  Logical extents 0 to 24  Raid Data LV 1  Logical volume wallvol\_rimage\_1  Logical extents 0 to 24  Raid Metadata LV 0 wallvol\_rmeta\_0  Raid Metadata LV 1 wallvol\_rmeta\_1 |

Note wallvol takes 25 extents from each disk, that’s 50 in all. The structure of wallvol is complicated and its discussion is beyond the scope of the course.