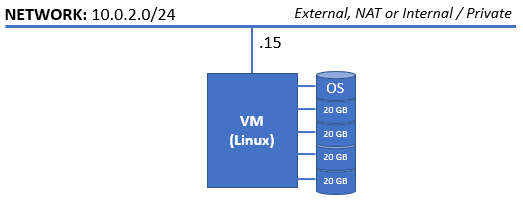
# Practice M1: Local Storage and Additional Techniques

This practice assumes that you are working in an on-premise environment

All tasks can be achieved under different configurations (host OS and/or virtualization solution) with the appropriate adjustments

This practice is oriented towards **Red Hat**-based distributions. You can use **CentOS 8.4**, **Fedora Server 34**, **Oracle Linux 8.4**, etc.

For this practice we will need an infrastructure like this one:



Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

## Part 1: Local Storage

Make sure, that you have prepared the required set of folders

**sudo mkdir -p /storage/{raid,lvm,btrfs,ext4,xfs}**

Every station has its **hosts** file prepared with records for other network members

### Software RAID

Let’s first examine our spare hard disks with **lsblk**

They should appear as four **20G** drives named **sdb**, **sdc**, **sdd**, and **sde**

Now, that we know the drives, let’s initialize them

We can use **fdisk** to prepare the **sdb** drive

**sudo fdisk /dev/sdb**

Make sure that you set the partition table to be **MBR**, created one primary partition with default values and set the type to **Linux raid autodetect** which is **fd**

There is a faster and script-ready way. Let’s execute the following for **sdc** drive

**sudo parted -s /dev/sdc -- mklabel msdos mkpart primary 2048s -0m set 1 raid on**

Let’s check if both approaches led to the same results

**sudo fdisk -l /dev/sd{b,c}**

Both partitions should look the same

We can process the other two drives following the second approach

**sudo parted -s /dev/sdd -- mklabel msdos mkpart primary 2048s -0m set 1 raid on**

**sudo parted -s /dev/sde -- mklabel msdos mkpart primary 2048s -0m set 1 raid on**

Now, we have our drives ready

Let’s make sure that we have the necessary tools installed

**sudo dnf install mdadm**

We can receive usage information for the **mdadm** with

**mdadm --help**

In order to ask for a specific set of commands, for example for array creation, execute

**mdadm --create --help**

#### RAID 0

In order to create a RAID 0 array, we need at least two devices

So, let’s use **sdb1** and **sdc1** to create the array

**sudo mdadm --create /dev/md0 --level 0 --raid-devices 2 /dev/sd{b,c}1**

We can check its status with

**cat /proc/mdstat**

It appears that our array is using **512k** sized chunks. It is the default value

Let’s change it, but first we will stop our array

**sudo mdadm --stop /dev/md0**

Erase the MD superblocks from the devices

**sudo mdadm --zero-superblock /dev/sd{b,c}1**

Use the following command to create a new array with **128k** chunks

**sudo mdadm --create /dev/md0 --level 0 --chunk 128 --raid-devices 2 /dev/sd{b,c}1**

Alternative approach to check the status of an array is to use

**sudo mdadm --detail /dev/md0**

We can shorten the command by substituting **--detail** with **-D**. Other actions have short names as well

#### RAID 1

Be sure that you stopped the array and clean up the devices

**sudo mdadm --stop /dev/md0**

**sudo mdadm --zero-superblock /dev/sd{b,c}1**

Use the following command to create a new RAID 1 array

**sudo mdadm --create /dev/md0 --level 1 --raid-devices 2 /dev/sd{b,c}1**

You will be asked to either change the metadata version with **--metadata 0.90** if you plan to use the array for **/boot**

If not, type **yes** and hit **Enter** to create the array

Check its status with

**sudo mdadm --detail /dev/md0**

#### RAID 5

Be sure that you stopped the array and clean up the devices

**sudo mdadm --stop /dev/md0**

**sudo mdadm --zero-superblock /dev/sd{b,c}1**

Use the following command to create a new RAID 5 array with **64k** chunks

**sudo mdadm --create /dev/md0 --level 5 --chunk 64 --raid-devices 3 /dev/sd{b,c,d}1**

Check its status with

**sudo mdadm --detail /dev/md0**

Let’s add one more device

**sudo mdadm /dev/md0 --add /dev/sde1**

We can check the status

**sudo mdadm --detail /dev/md0**

Now, we must reconfigure the array to use the new drive as well

**sudo mdadm /dev/md0 --grow --raid-devices 4 --backup-file /tmp/md0-grow.bak**

As a final step, we must extend the filesystem, if any, residing on the array

#### RAID 10

Be sure that you stopped the array and clean up the devices

**sudo mdadm --stop /dev/md0**

**sudo mdadm --zero-superblock /dev/sd{b,c,d,e}1**

Use the following command to create a new RAID 10 array with **64k** chunks

**sudo mdadm --create /dev/md0 --level 10 --chunk 64 --raid-devices 4 /dev/sd{b..e}1**

Check its status with

**sudo mdadm --detail /dev/md0**

#### Using an array

We must take some extra measures to ensure that the array will re-assemble automatically after reboot

There is a special configuration file **/etc/mdadm.conf** that must contain information about the array

The easiest way to achieve this is to execute

**sudo mdadm --detail --brief /dev/md0 | sudo tee -a /etc/mdadm.conf**

Alternatively, if we had multiple freshly created arrays, we could add all of them in one step

**sudo mdadm --detail --scan | sudo tee -a /etc/mdadm.conf**

Now, let’s create an **ext4** filesystem on the array

**sudo mkfs.ext4 /dev/md0**

We can mount the array in the **/storage/raid** folder

**sudo mount /dev/md0 /storage/raid**

To check if the mount was successful, we can use either **mount -l**, **lsblk**, **/etc/mtab** or **/proc/mounts**

Should we want these changes to persist, we must change the **/etc/fstab** file as well

For this we will need to find the UUID with

**sudo blkid /dev/md0**

#### Clean up

Unmount the array

**sudo umount /dev/md0**

Be sure that you stopped the array

**sudo mdadm --stop /dev/md0**

And then clean up the devices by erasing the MD superblocks from them

**sudo mdadm --zero-superblock /dev/sd{b,c,d,e}1**

Furthermore, you can use **dd** to mangle the drives’ content

If you added the array to the **/etc/mdadm.conf** file, remove it

If you edited the **/etc/fstab** file, revert the changes

Finally, use **wipefs** to remove any remaining structures on disks

**sudo wipefs --all /dev/sd{b..e}**

Check the result with **lsblk**

### LVM2

We will assume that we are starting with four clean spare hard disks

For the purpose of LVM, we can work with either hard drives or partitions

First, depending on our installation, we may need to install the required packages with:

**sudo dnf install lvm2**

#### Physical volumes

Knowing that we have **sdb**, **sdc**, **sdd**, and **sde**, we can start with LVM header creation

Let’s initialize the first available drive

**sudo pvcreate /dev/sdb**

We can ask for detailed information for the physical volume we just created with

**sudo pvdisplay /dev/sdb**

In order to increase the verbosity level, we can use **-v** (it is on by default), **-vv**, **-vvv**, or **-vvvv**

By default, each physical volume contains one copy of LVM metadata just after the LVM label

We can create a physical volume with either none or for example two copies of metadata. Let’s create with two

**sudo pvcreate --metadatacopies 2 /dev/sdc**

Verbosity can be added and/or increased for other commands as well, for example for the **pvcreate**

Let’s remove the two PVs

**sudo pvremove /dev/sdb /dev/sdc**

And create four PVs out of the four available drives

**sudo pvcreate /dev/sd{b..e} -vv**

Once done, we can ask for PV details with

**sudo pvs -v**

#### Volume groups

Let’s create a volume group using one of the PVs created so far

**sudo vgcreate vg\_demo /dev/sdb**

Now, let’s get a short information about the available VGs with

**sudo vgs**

Okay, let’s increase the verbosity level

**sudo vgs -v**

We can ask for detailed information about a particular VG with

**sudo vgdisplay vg\_demo**

And again, we can increase the verbosity with

**sudo vgdisplay vg\_demo -v**

Let’s create a second VG but this time with a different extent size

**sudo vgcreate vg\_test --physicalextentsize 8 /dev/sdc**

And check the summary with

**sudo vgs -v**

Now, that we know how to create a VG with different extent size, let’s delete it

**sudo vgremove vg\_test**

And extend the one that we created first to include the other three PVs. First, add just the **sdc**

**sudo vgextend vg\_demo /dev/sdc**

Check the detailed information

**sudo vgdisplay vg\_demo -v**

And add the other two available PVs

**sudo vgextend vg\_demo /dev/sdd /dev/sde**

#### Logical volumes

Let’s create one logical volume with 1GB in size

**sudo lvcreate -L 1G -n lv\_demo vg\_demo**

And another one with smaller size

**sudo lvcreate -L 25M -n lv\_demo\_tiny vg\_demo**

We can see that the second logical volume was created with different size not the requested one. A logical volume size must be a multiplication of the extent size

Ask for a brief information about the logical volumes

**sudo lvs -v**

Now, let’s execute the **lsblk** command

It appears that both logical volumes are residing on the **sdb** device. This has to do with the space allocation process

Let’s remove the smaller LV with

**sudo lvremove vg\_demo/lv\_demo\_tiny**

Confirm with **y**

#### Using a logical volume

Let’s create an **ext4** filesystem on the logical volume

**sudo mkfs.ext4 /dev/vg\_demo/lv\_demo**

We can mount the array in the **/storage/lvm** folder

**sudo mount /dev/vg\_demo/lv\_demo /storage/lvm**

Create a **hello.txt** file with some text and store it in the **/storage/lvm** folder

**echo 'Hello LVM' | sudo tee /storage/lvm/hello.txt**

#### Growing a logical volume

Let’s extend our logical volume

We can give a fixe required size with

**sudo lvextend -L10G /dev/vg\_demo/lv\_demo**

Extending can be done with a step

**sudo lvextend -L +5G /dev/vg\_demo/lv\_demo**

Of course, we can ask the logical volume to occupy the whole available space on the PVs with

**sudo lvextend -l 100%PVS /dev/vg\_demo/lv\_demo**

Now, we must extend our filesystem as well with

**sudo resize2fs /dev/vg\_demo/lv\_demo**

And check if the file is still there and readable

**cat /storage/lvm/hello.txt**

#### Shrinking a logical volume

When shrinking a logical volume and the filesystem residing on it, first we should do a backup of our data

Next step is to unmount the filesystem with

**sudo umount /storage/lvm**

Let’s check the filesystem with

**sudo e2fsck -f /dev/vg\_demo/lv\_demo**

Now, we are ready to reduce or shrink the filesystem down to 5GB for example

**sudo resize2fs /dev/vg\_demo/lv\_demo 5G**

We are ready to reduce the logical volume

We can do it with a subtraction, for example with 10GB

**sudo lvreduce -L -10G vg\_demo/lv\_demo**

Or we can use a desired size

**sudo lvreduce -L 5G vg\_demo/lv\_demo**

Re-check the filesystem just in case

**sudo e2fsck -f /dev/vg\_demo/lv\_demo**

Finally, we can re-mount our filesystem

**sudo mount /dev/vg\_demo/lv\_demo /storage/lvm**

And check if our file is there and readable

**cat /storage/lvm/hello.txt**

Everything should be just fine

#### Snapshots

Let’s create a snapshot of our logical volume

**sudo lvcreate -s -L 1G -n lv\_demo\_snap /dev/vg\_demo/lv\_demo**

And check its details

**sudo lvdisplay vg\_demo/lv\_demo\_snap**

Now, check the details of the initial logical volume

**sudo lvdisplay vg\_demo/lv\_demo**

Let’s check with

**sudo lvs**

And let’s execute **lsblk**. It appears that one more level has been created

Prepare a mount point

**sudo mkdir /storage/lvm-snap**

And mount the snapshot as well

**sudo mount /dev/vg\_demo/lv\_demo\_snap /storage/lvm-snap**

We can compare both the original and the snapshot, they should be the same

Now, change the text in the original file with:

**echo '... some more text' | sudo tee -a /storage/lvm/hello.txt**

And compare them again. The one in the snapshot should remain unchanged

Okay, let’s delete the **hello.txt** file from the original volume

**sudo rm /storage/lvm/hello.txt**

Now, check if the file is still available in the snapshot

**ls -al /storage/lvm-snap/**

We can restore it by just copying back from the snapshot, or restore the whole snapshot

Depending on the circumstances, we can go either way

Let’s restore the snapshot

First, we must unmount both the origin and the snapshot

**sudo umount /storage/lvm /storage/lvm-snap**

The actual operation is called merge and it is started with

**sudo lvconvert --merge /dev/vg\_demo/lv\_demo\_snap**

Then, we can check all logical volumes with

**sudo lvs**

And then, mount the origin

**sudo mount /dev/vg\_demo/lv\_demo /storage/lvm**

And finally, check if the file is back there

**cat /storage/lvm/hello.txt**

#### Thin provisioning

Let’s prepare a new logical volume

**sudo lvcreate -L 1G --thinpool tp\_demo vg\_demo**

Let’s check

**sudo lvs**

Our new **tp\_demo** appears, beside the existing ones

Now, let’s create a volume with

**sudo lvcreate -V 5G --thin -n tp\_demo\_lv vg\_demo/tp\_demo**

We can ask for list of volumes with

**sudo lvs**

Make a mount point

**sudo mkdir /storage/lvm-thin**

Create a filesystem

**sudo mkfs.ext4 /dev/vg\_demo/tp\_demo\_lv**

And mount it

**sudo mount /dev/vg\_demo/tp\_demo\_lv /storage/lvm-thin**

Now, let’s check the effect of creating a filesystem on the volume

**sudo lvdisplay vg\_demo/tp\_demo\_lv**

And on the pool

**sudo lvdisplay vg\_demo/tp\_demo**

Some space has been consumed, so we should monitor and be pro-active

If we start copying files, this space will go away, and the filesystem will crash. We should be careful

#### Striping

By default, space coming from PVs is consumed in a linear fashion

We can create a striped logical volume

**sudo lvcreate -L 1G -n lv\_demo\_stripe1 -i 4 vg\_demo**

Now, check the how the stripes are distributed with

**sudo lvdisplay vg\_demo/lv\_demo\_stripe1 -m**

Okay, let’s create one more, but this time with different stripe size and distribution

**sudo lvcreate -L 1G -n lv\_demo\_stripe2 -i 2 -I 128 vg\_demo /dev/sdc /dev/sdd**

And check its details

**sudo lvdisplay vg\_demo/lv\_demo\_stripe2 -m**

Let’s execute **lsblk**

The picture gets messy 😉

#### Clean up

First unmount all LVM related mounts

**sudo umount /storage/lvm\***

Next, delete the volume group together with all logical volumes

**sudo vgremove vg\_demo --force**

Wipe all four drives

**sudo wipefs --all /dev/sd[b-e]**

## Part 2: Advanced Filesystems

Let’s explore both the BTRFS and ZFS

We will need just one machine. Let’s continue with the server from the first part

***Please note that mounting-related steps can be done on Fedora 3x***

### Advanced Filesystems (BTRFS)

Let’s start with BTRFS

First, we must make sure that the required packages are installed

Unfortunately, **Red Had** decided to remove the support of BTRFS in 8.x, while there is support for it in **Fedora 3x**

Under **CentOS**, we must add additional repository

**sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org**

**sudo dnf install https://www.elrepo.org/elrepo-release-8.el8.elrepo.noarch.rpm**

Enable all channels (or at least the testing one) for the newly added repository

**sudo vi /etc/yum.repos.d/elrepo.repo**

Install the package

**sudo dnf install btrfs-progs**

#### Single disk

Let’s create a filesystem over a single disk

**sudo mkfs.btrfs -d single /dev/sdb**

As we can see both the metadata and system data are duplicated

Should we want, we can set them to single mode as well

**sudo mkfs.btrfs -d single -m single -f /dev/sdb**

We can mount it with

**sudo mount /dev/sdb /storage/btrfs**

Let’s create a few small files with

**for i in $(seq 1 5); do sudo dd if=/dev/zero of=/storage/btrfs/file$i.img bs=1M count=10 ; sync ; done**

Check with

**df -hT**

Now, check device status with

**sudo btrfs device usage /storage/btrfs**

Repeat the procedure with files creation but create bigger files

**for i in $(seq 1 5); do sudo dd if=/dev/zero of=/storage/btrfs/file$i.img bs=10M count=100 ; sync ; done**

Check disk usage again

**df -hT**

And now the device usage

**sudo btrfs device usage /storage/btrfs**

#### Multiple disks (Linear)

Now, we can add two more disks with

**sudo btrfs device add /dev/sdc /dev/sdd /storage/btrfs**

Check disk usage again

**df -hT**

And now the device usage

**sudo btrfs device usage /storage/btrfs**

We can have another view on this with

**sudo btrfs filesystem show /storage/btrfs**

Let’s re-balance the filesystem

**sudo btrfs balance start -d -m /storage/btrfs**

Check disk usage again

**df -hT**

No changes here

And now the device usage

**sudo btrfs device usage /storage/btrfs**

And the alternative view

**sudo btrfs filesystem show /storage/btrfs**

Why not remove one drive? Let’s do it

Remove the third drive with

**sudo btrfs device delete /dev/sdd /storage/btrfs**

The data is first relocated and then the device is removed

We can use the known commands to check

#### Multiple disks (RAID)

Let’s convert the existing filesystem to RAID1

**sudo btrfs balance start -dconvert=raid1 -mconvert=raid1 /storage/btrfs**

And of course, check again what is the situation

Now, let’s add the other two disks

**sudo btrfs device add /dev/sdd /dev/sde /storage/btrfs**

And convert the file system to RAID10 for data and RAID1 for metadata

**sudo btrfs balance start -dconvert=raid10 -mconvert=raid1 /storage/btrfs**

#### Subvolumes

We can use subvolumes to introduce alternative roots which function as independent filesystems

Let’s create one

**sudo btrfs subvolume create /storage/btrfs/svol**

Now, we can ask for a list of subvolumes

**sudo btrfs subvolume list /storage/btrfs**

Create an empty file there

**sudo touch /storage/btrfs/svol/empty\_file**

And check the hierarchy

**tree /storage/btrfs**

Let’s create another mount point for the subvolume

**sudo mkdir -p /storage/btrfs-svol**

And mount it there

**sudo mount -o subvolid=279 /dev/sdb /storage/btrfs-svol**

Should we want to delete a subvolume, first we must make sure that it is empty and then use (skip it for now)

**sudo btrfs subvolume delete /storage/btrfs/svol**

#### Snapshots

Snapshots are special type of subvolume. They contain a copy of the current state of another subvolume

Let’s create a simple text file

**echo 'Hello BTRFS' | sudo tee /storage/btrfs/hello.txt**

And then create a snapshot of the main (root) BTRFS volume

**sudo btrfs subvolume snapshot /storage/btrfs /storage/btrfs/snap**

Check the directory hierarchy

**tree /storage/btrfs**

Change the original text file

**echo '... some additional text' | sudo tee -a /storage/btrfs/hello.txt**

Check the content of both files

Now, if we remove the original file, we can easily copy the one from the snapshot (either by mounting it as a regular filesystem or directly)

Unmount all BTRFS filesystems

**sudo umount /storage/btrfs /storage/btrfs-svol**

And clean the devices

**sudo wipefs --all /dev/sd{b..e}**

### Advanced Filesystems (ZFS)

Let’s continue with ZFS on Linux

Add the ZFS on Linux repository

**sudo dnf install https://zfsonlinux.org/epel/zfs-release.el8\_4.noarch.rpm**

**sudo rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-zfsonlinux**

By default, the **zfs-release** package is configured to install **DKMS** style packages so they will work with a wide range of kernels

In order to install the **kABI-tracking** kmods the default repository in the **/etc/yum.repos.d/zfs.repo** file must be switch from **zfs** to **zfs-kmod**

**sudo dnf config-manager --disable zfs**

**sudo dnf config-manager --enable zfs-kmod**

Install the necessary packages

**sudo dnf install zfs**

Execute the following to autoload the module

**echo "zfs" | sudo tee -a /etc/modules-load.d/zfs.conf**

Reboot the system

Okay, we have the binaries, the module and the services are auto-loading, and four spare 20GB drives, so let’s start

Create two mount points

**sudo mkdir -p /storage/zfs{m,s,r}**

#### Striped pool

In order to create a striped pool, we must execute (skip it)

**sudo zpool create zfs-stripe /dev/sdb /dev/sdc**

This will mount the pool in **/zfs-stripe** folder

Should we want a custom mount point, we can execute

**sudo zpool create -m /storage/zfss zfs-stripe /dev/sdb /dev/sdc**

Now, we can ask for its status with

**sudo zpool status zfs-stripe**

Or we can list all pools with

**sudo zfs list**

#### Mirrored pool

More or less the procedure is the same as with the stripe mode

In order to create a mirrored pool, but mount it in a custom point

**sudo zpool create -m /storage/zfsm zfs-mirror mirror /dev/sdd /dev/sde**

Now, we can ask for its status with

**sudo zpool status zfs-mirror**

Or we can list all pools with

**sudo zfs list**

#### RAID5-like pool

Let’s unmount both pools

**sudo umount /storage/zfs\***

Then execute the **destroy** command

**sudo zpool destroy zfs-mirror**

**sudo zpool destroy zfs-stripe**

Then clean the drives

**sudo wipefs --all /dev/sd[b-e]**

In order to create a RAID5-like (RAIDZ) pool, but mount it in a custom point, we must execute

**sudo zpool create -m /storage/zfsr zfs-raidz raidz /dev/sdb /dev/sdc /dev/sdd**

Check the status with the known commands

#### Clean up

In order to delete a pool, first we must unmount it

**sudo umount /storage/zfsr**

Then execute the **destroy** command

**sudo zpool destroy zfs-raidz**

We can clean the drives

**sudo wipefs --all /dev/sd[b-e]**

## Part 3: Additional Storage Techniques

Let’s explore a few more storage topics

We will need just one machine. Let’s continue with the server from the second part

### Quotas

Prepare drive **sdb** by creating a partition and filesystem

**sudo parted -s /dev/sdb -- mklabel msdos mkpart primary 2048s -0m set 1**

**sudo mkfs.ext4 /dev/sdb1**

Prepare drive **sdc** by creating a partition and filesystem

**sudo parted -s /dev/sdc -- mklabel msdos mkpart primary 2048s -0m set 1**

**sudo mkfs.xfs -f /dev/sdc1**

Mount both in **/etc/fstab**

#### Ext4 quotas

Open the **/etc/fstab** and change the record for the **ext4** filesystem to enable user quota

Just substitute **defaults** with **usrquota**

Save and close the file

Unmount the filesystem

**sudo umount /storage/ext4**

And mount it again

**sudo mount -av**

If a warning SELinux message appears, execute this

**sudo restorecon /storage/ext4**

Now, let’s create the quota database

**sudo quotacheck -mu /dev/sdb1**

If the command is missing, you can install the following package and then re-run the command

**sudo dnf install quota**

Let’s check the filesystem’s content

**ls -al /storage/ext4/**

We can see that there is a quota file (database) for users

Turn on the quota for the partition

**sudo quotaon /dev/sdb1**

Check the current state of space usage (or quota utilization) on the partition

**sudo repquota -uv /dev/sdb1**

Let’s create a new **demo** user

**sudo useradd demo**

And set its password to something easy

**sudo passwd demo**

Now, set the quota for the demo user

**sudo setquota -u demo 20000 25000 0 0 /dev/sdb1**

And check the quota utilization again

**sudo repquota -uv /dev/sdb1**

Let’s change the quota with

**sudo edquota -u demo**

Set the soft limit to **18000**

Save and close

Let’s add one more user named **knight**

**sudo useradd knight**

And set quota for it based on the **demo** user

**sudo edquota -u knight -p demo**

Check the quota utilization again

**sudo repquota -uv /dev/sdb1**

Now, let’s give right to everyone to write in the folder with

**sudo chmod 777 /storage/ext4**

Switch to the **demo** user

**su demo**

Create a **5 MB** file named **fill.dat** with

**dd if=/dev/zero of=/storage/ext4/fill.dat**

Ooops, we forgot to set block size and block count

Fortunately, because of the quota the operation was interrupted

Delete the file and this time create a file with the right size

**rm /storage/ext4/fill.dat**

**dd if=/dev/zero of=/storage/ext4/fill.dat bs=1M count=5**

Everything is going as expected

Add one more file but **15 MB** in size

**dd if=/dev/zero of=/storage/ext4/fill-15M.dat bs=15M count=1**

We received a warning

Exit to the regular user

Ask for the quota utilization

**sudo repquota -uv /dev/sdb1**

We can see that the soft quota is breached for the **demo** user

#### XFS quotas

Open the **/etc/fstab** and change the record for the **xfs** filesystem to enable user quota

Just substitute **defaults** with **usrquota,uqnoenforce**

Save and close the file

Unmount the filesystem

**sudo umount /storage/xfs**

And mount it again

**sudo mount -av**

If a warning SELinux message appears, execute this

**sudo restorecon /storage/xfs**

Now, let’s set some quota limits

**sudo xfs\_quota -xc 'limit -u bsoft=20m bhard=25m demo' /dev/sdc1**

Change folder’s permissions

**sudo chmod 777 /storage/xfs**

Switch to the **demo** user

**su demo**

And create a **15 MB** file

**dd if=/dev/zero of=/storage/xfs/fill-15M.dat bs=15M count=1**

Return to the regular user and ask for quota utilization report

**sudo xfs\_quota -c 'quota demo' /dev/sdc1**

We can switch again to the **demo** user

Let’s try to create a second file but this time **20 MB** file

**dd if=/dev/zero of=/storage/xfs/fill-20M.dat bs=20M count=1**

It allowed us. Strange

Exit the user and ask for a report

**sudo xfs\_quota -c 'quota demo' /dev/sdc1**

Most probably this situation has something to do with the options in the **/etc/fstab**

So, with XFS we can monitor and report on quota breaches and at the same time not disrupting the users

Should we want to enforce quota rules, we must remove the **uqnoenforce** option in the **fstab** file

We can use the **xfs\_quota** in interactive manner as well

### Encryption

#### Preparation

For this part we will need the **DM\_CRYPT** module

Let’s check is it available as part of the kernel or it is loadable on demand

**grep -i DM\_CRYPT /boot/config-$(uname -r)**

We can check if it is already loaded with

**sudo lsmod | grep dm\_crypt**

If not, we can try to load it

**sudo modprobe dm\_crypt**

And check again if it is loaded

**sudo lsmod | grep dm\_crypt**

Make sure that the **cryptsetup** package is installed

**sudo dnf install cryptsetup**

Finally, make sure that there is a partition available for encryption

Let’s create a small partition on one of the spare drives

**sudo parted -s /dev/sdd -- mklabel msdos mkpart primary 2048s 1024m set 1**

#### Encrypt a partition

It is a good practice to **shred** the partition before the encryption process. This will make harder to be determined where the encrypted data resides

Initiate the encryption procedure

**sudo cryptsetup -y luksFormat /dev/sdd1**

Answer with **YES**

Then enter and re-enter encryption passphrase

Now, we can check what options were used during the encryption

**sudo cryptsetup luksDump /dev/sdd1**

Usually, there is a tab-completion for the sub-commands

We can explore the set of sub-commands

For example, we can try the **isLuks** and test our encrypted and some other partition

Then, we must use the **echo $?** command to check the result

#### Work with encrypted partition

Let’s open the encrypted partition

**sudo cryptsetup luksOpen /dev/sdd1 encr**

Now, check for the **encr** partition

**ls /dev/mapper/**

Let’s create a filesystem on it

**sudo mkfs.xfs /dev/mapper/encr**

Once we are done, we can close the partition

**sudo cryptsetup luksClose encr**

Let’s open it again but this time with a different name

**sudo cryptsetup luksOpen /dev/sdd1 enc-data**

Of course, it appears with the new name

**ls /dev/mapper**

#### Mount encrypted partition on boot

Prepare a mount point

**sudo mkdir -p /storage/enc**

We must either use one and the same name every time in order to be able to use it in **/etc/fstab** or obtain the UUID

**sudo blkid /dev/sdd1**

Now open the **/etc/crypttab** file

**sudo vi /etc/crypttab**

And paste

**enc-data UUID=<uuid>**

Save and close the file

Now, open the **/etc/fstab** file

**sudo vi /etc/fstab**

As last record enter

**/dev/mapper/enc-data /storage/enc xfs defaults 0 0**

Save and close the file

Close the encrypted partition if open

**sudo cryptsetup luksClose enc-data**

Then, re-open it

**sudo cryptsetup luksOpen /dev/sdd1 enc-data**

Now, try to mount it

**sudo mount -av**

Now, reboot the machine and use the console of the virtualization solution to interact with the VM

When the booting stops, it will ask you to enter the password to open the encrypted partition

Once, you enter it correctly the boot process will continue

### Automounting

Let’s install the required package

**sudo dnf install autofs**

Now, that we have the package installed, let’s examine the set of configuration files

**ls -al /etc/auto\***

The main configuration file is the **/etc/autofs.conf**

Should we want to change, for example, the timeout after which to unmount the resource, we must do it here

Let’s start the service and enable it on boot

**sudo systemctl enable --now autofs**

#### Automount encrypted device

Remove the **/etc/fstab** entry related to the mounting of the encrypted partition

Remove the **/etc/crypttab** entry or file as well

Unmount it

**sudo umount /storage/enc**

And then close it

**sudo cryptsetup luksClose enc-data**

Open the **/etc/auto.misc** file and add a row bellow the **cd** one

**enc -fstype=xfs :/dev/mapper/enc-data**

Close and save the file

Restart the **autofs** service

**sudo systemctl restart autofs**

Open the encrypted volume

**sudo cryptsetup luksOpen /dev/sdd1 enc-data**

If asked for the password used during the encryption, then enter it

Check the **/misc** directory

**ls -al /misc**

It is empty

Navigate to **/misc/enc**

You are allowed to enter

Now, if you ask for the mounted filesystems, you will notice that the one, residing on the encrypted partition is currently mounted

#### Automount NFS export

*You can test this in a later stage when we complete the M3 module*

Check what are NFS exports we have currently

**sudo exportfs -s**

Try to mount it locally

**sudo mount server:/storage/nfs/share /mnt**

Check with

**mount -l**

Now, unmount it

**sudo umount /mnt**

Let’s edit the **/etc/auto.misc** file and add

**nfs -fstype=nfs,ro,soft,intr server:/storage/nfs/share**

Save and close the file

Restart the **autofs** service

Go to **/misc/nfs** folder

If you check the mounts, you will notice that it was auto-mounted

You can browse its content