PSAwise2324Team10Aufgabe07

File Server

We decided to set up the file server on VM1 and use RAID 5 as the storage configuration, since it is popular and has many advantages.

Namely, RAID 5 provides good performance and fault tolerance by striping across multiple drives and utilizing parity. Furthermore, the storage capacity is used efficiently as it needs only one drive for parity information. RAID 5 can also tolerate the failure of a single drive without data loss.

However, it is important to note that in case of a drive failure, the system is in a vulnerable state until the failed drive and its data are replaced. If a second drive fails during this period, data loss can occur.

Hardware Setup

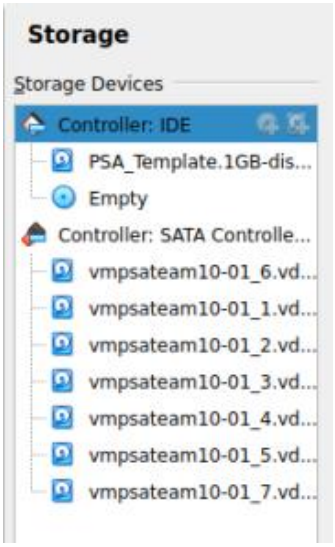
The new disks need to be added first, so you have to turn off the VM.

Add a new AHCI/SATA controller to the VM, which will control the disks:

```
vboxmanage storagectl vmpsateam10-01 --name "SATA Controller" --add sata --
controller IntelAHCI --portcount 1
```

We added the controller to port 1 because port 0 is already used by the original disk.

Now, you can add new VDI disks to the VM. We decided to create seven disks, because we need at least six to confidently reach the 10+GB requirement of the exercise and one more for parity data. Open settings>storage of VM1 in VirtualBox and click "add new hard disk" next to the name of the controller. In the menu that opens, select the type "VirtualBox Disk Image (VDI)" and set the size of each new disk to 2GB. Repeat until all disks have been created.



After booting VM1, the output of `lsblk` should look as follows:

```
patricia@vmpsateam10-01:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda      8:0    0   40G  0 disk
├─sda1   8:1    0   31K  0 part
├─sda2   8:2    0 47.7M  0 part
├─sda3   8:3    0  416M  0 part /boot/efi
└─sda4   8:4    0 39.5G  0 part /
sdb      8:16   0    2G  0 disk
sdc      8:32   0    2G  0 disk
sdd      8:48   0    2G  0 disk
sde      8:64   0    2G  0 disk
sdf      8:80   0    2G  0 disk
sdg      8:96   0    2G  0 disk
sdh      8:112  0    2G  0 disk
sr0     11:0    1 1024M  0 rom
```

To implement RAID 5, you need to install the `mdadm` utility tool using apt. You can then create the array on sdb to sdh using this command: `sudo mdadm --create --verbose /dev/md0 --level=5 --raid-devices=7 /dev/sdb /dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg /dev/sdh`.
The new space containing the array is called `/dev/md0` following convention.
The output should look like this:

```
patricia@vmpsateam10-01:~$ sudo mdadm --create --verbose /dev/md0 --level=5 --raid-devices=7
/dev/sdb /dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg /dev/sdh
mdadm: layout defaults to left-symmetric
mdadm: layout defaults to left-symmetric
mdadm: chunk size defaults to 512K
mdadm: size set to 2094080K
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

You can also verify the creation of the array by looking at the contents of `/proc/mdstat`:

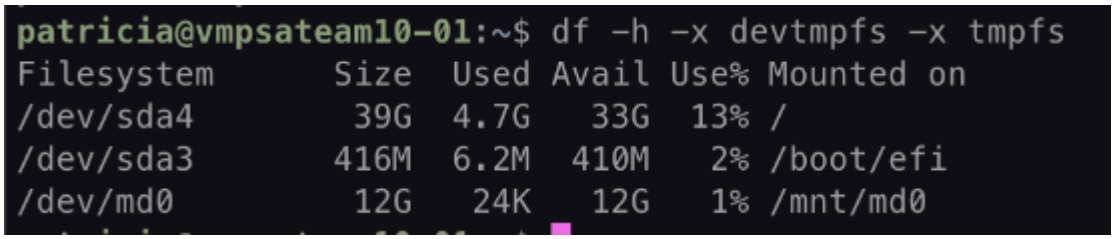
```
patricia@vmpsateam10-01:~$ cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdh[7] sdg[5] sdf[4] sde[3] sdd[2] sdc[1] sdb[0]
      12564480 blocks super 1.2 level 5, 512k chunk, algorithm 2 [7/7] [UUUUUUU]

unused devices: <none>
```

Finally, create a filesystem in the new storage space.

Use the following commands to format and mount the filesystem:

```
sudo mkfs.ext4 -F /dev/md0
sudo mkdir -p /mnt/md0
sudo mount /dev/md0 /mnt/md0
```

If you run `df -h -x devtmpfs -x tmpfs` after this, you should find the /mnt/md0 section listed with 12GB of free space.



To make sure that the array is reassembled correctly at boot, run the following commands.

```
sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
sudo update-initramfs -u
echo '/dev/md0 /mnt/md0 ext4 defaults,nofail,discard 0 0' | sudo tee -a
/etc/fstab
```

The first command scans the active array and appends to the config file. After updating the config file, we need to update initramfs as well. This allows for the array to be available during the early boot process.

Data Migration

Now that everything is set up, you can migrate the data to the fileserver. As suggested in the exercise sheet, we decided to use rsync to copy files and NFS to make the data on the file server accessible from client machines.

File Server Configuration

Install the NFS server package on the NFS server VM by running `sudo apt install nfs-kernel-server` , then start and enable it.
In order for NFS to be able to send and receive data, you have to add the following rules to the firewall (NFS uses port 2049 by default):

```
chain input {
    <...>
    tcp dport 2049 accept
    <...>
}
chain output {
    <...>
    tcp sport 2049 accept
    <...>
}
```

Then, create the new directories for the file server data. We chose to store all home directories, our webserver data files and our database data files on the file server:

```
sudo mkdir -p /mnt/md0/home_dirs
sudo mkdir -p /mnt/md0/www
sudo mkdir -p /mnt/md0/postgresql
```

In case of the home directories, we also created VM-specific subdirectories (`/home_dirs/vm3_home` etc.), since we wanted to keep the home directories of each VM separate.
We decided to not move the database backup to the fileserver, because VM1 (the fileserver) also hosts the database and keeping the backup there would defeat its purpose.

Before you copy the data with rsync, make sure to turn off all services that might be accessing it to prevent data loss and create a backup of the data. In our case, we cloned the VMs in question and created local copies of the directories as well. (If storage space is limited, it would be best to remove the backup from the VM and put it somewhere else after the data transfer. Since this is a course at university and we had enough space left, we decided keep the backups on the VMs for convenience.)

To copy files from a VM in the network to the file server, you need to remotely run rsync with sudo privileges on the machine the data is originally on. In order to do that, we temporarily added the line `<username> ALL=NOPASSWD:/usr/bin/rsync` at the bottom (!) of the client VM's `/etc/sudoers` file and specified `--rsync-path="sudo rsync"` in the command. The full command looks as follows:

```
sudo rsync -aAXUH --rsync-path="sudo rsync"
<username>@<ip_of_source_vm>:<directory_to_copy>
/mnt/md0/<dest_directory_on_file_server>
```

"source_vm" in this context is the VM where the data that we want to move to the fileserver is located before the transfer.

The options aAxUH specify the following:

- a = archive mode (make copy more similar to the original; copy recursively, preserve permissions, symlinks etc.)
- A = preserve access control lists
- x = stay within this filesystem
- U = preserve extended attributes
- H = preserve hardlinks

After running this command and verifying that all files have been copied and no errors occurred, empty the original directory on the source VM. This is where the file shares for this data will be mounted.

In order for the client machines to be able to access the directories on the fileserver, you have to list them in `/etc/exports` on the file server VM:

```
/mnt/md0/www            192.168.10.5(rw,sync,no_root_squash,no_subtree_check)
# specifies IP of our webserver
/mnt/md0/postgresql     192.168.10.1(rw,sync,no_root_squash,no_subtree_check)
# this VM also hosts the DB
/mnt/md0/home_dirs      192.168.0.0/16(rw,no_root_squash,no_subtree_check) #
home directories are accessible from all PSA IP addresses
```

The first segment is the file path to the directory on the fileserver, second is the IP address of the machine that that directory should be shared with, plus a few options that determine with what restrictions the directory should be shared.

The options do the following:

- rw = allows reading and writing
- sync = changes are written to the disk after every operation (to prevent data loss)
- no_root_squash = allows user on client VM to access the files with root privileges if the user has them (instead of being "squashed" to unprivileged user)
- no_subtree_check = disable subtree_check; subtree_check can cause certain errors/reliability issues (man page quote: "more problems than it's worth")
    - subtree_check = checks that accessed file's path is entirely within exported directory

Make sure to apply these changes by running `exportfs -a`.

Client Configuration

On any VM you want to access the directories shared by the file server from, install the NFS client package using `sudo apt install nfs-common`.
Then add the following rules to the firewall to allow NFS traffic (NFS uses port 2049 by default):

```
chain input {
    <...>
    ip saddr 192.168.10.1 tcp sport 2049 accept
    <...>
}
chain output {
    <...>
    ip daddr 192.168.10.1 tcp dport 2049 accept
    <...>
}
```

Next, specify a mount point on this machine for the directory you want to access in /etc/fstab. This way, the specified NFS directory will be mounted at bootup and the data on the file server will be accessible. Here is an example of how this would look for the /mnt/md0/www directory, which we mounted at /var/www on VM5.

```
192.168.10.1:/mnt/md0/www   /var/www   nfs   rw,soft,sync,timeo=10   0   0
```

This line is structured as follows:

1. file path on fileserver
2. mount point on client
3. file system type
4. options
5. file system check order (not really relevant here)

As options, we chose

- rw = read write access
- soft = if a request times out, view operation as failed and stop trying (can cause data loss in some circumstances)
  (`hard,intr` used to arguably be better, but `intr` is deprecated and ignored by our kernel version. Without `intr`, `hard` could get stuck forever requiring manual interruption, so we decided to use `soft` instead.)
- sync = write all changes to disk immediately (increases reliability)
- timeo = timeout after this amount of time if access fails

We read that the combination `hard,intr` used to arguably be better, but `intr` is deprecated and ignored in our kernel version, which leaves only `hard` and could cause issues, so we chose to use soft instead)

SMB

SMB is, like NFS, a file sharing protocol. While NFS is designed for Unix systems, SMB was made to share files with Windows machines. Samba is a popular implementation of the SMB protocol, which we used for this exercise.

First of all, install Samba on the file server VM with `sudo apt install samba`, then enable it. You have to add rules permitting port 445 and 139 to the firewall, in order for Samba to work. (input dport and output sport on VM1)
Navigate to the directory /etc/samba and open the file smb.conf with a text editor. You have to add a share section for each individual directory. The following is an example showing what our share section for the home directories looks like:

```
[home_dirs]
   comment = home directories of team 10 # explains what this is
   hosts allow = 192.168.0.0/16 # only machines with one of these IP addresses
can access these files
   path = /mnt/md0/home_dirs # location on VM1
```

```
   read only = no # users have write permission
   browsable = yes # whether this share shows up in net view/browse list
```

After saving the changes, you can use `testparm` to check if the syntax is correct. If it is, run `sudo systemctl restart smbd` to apply changes.
Unlike NFS, where we used mount points, access to files on SMB is user based. To create a user, run the following command:

```
sudo smbpasswd -a <username>
```

You will be prompted for a password. For simplicity's sake, we used "psa" as the password for every user. To check whether the users have been created, you can run `sudo pdbedit -L` and see if the user appears in the list.
To check whether you can actually access files with this user, connect to another VM, `sudo apt install smbclient`, allow traffic from/to VM1 on port 445 and 139 in the firewall and then run this command:

```
smbclient //192.168.10.1/<share_section_name> -U <username>
```

If everything works, this should connect you to the specified share in interactive mode, where you have access to the files in the directory. (If you want to test this out, you can do so on VM5. We didn't install smbclient on every VM, since it's just for testing and isn't intended for access by Linux machines anyway)

We created a samba user for every PSA participant on VM1 using this script:

```
#!/bin/bash

# introduces users variable
source ./psa_users.sh

for username in "${users[@]}"; do
    # Check if the user exists
    if sudo pdbedit -L | grep -q "$username"  &>/dev/null; then
        echo "already exists: $username"
    else
        echo "ADDING $username..."
        # for some reason, the EOF block has to be formatted this way
        # will throw error if it fails
        sudo smbpasswd -a "$username"<<EOF > /dev/null
psa
psa
EOF

    fi
done
```

Other Team's Home Directories

We used autofs to create non-static mountpoints for the home directories of other PSA teams. To start with, install autofs on the file server VM with `sudo apt install autofs`. Then, create the file `/etc/auto.homes` and create an entry for every home directory that is supposed to be accessed:

```
#1
schoe -fstype=nfs,rw 192.168.1.1:/datapool/home/schoe
steph -fstype=nfs,rw 192.168.1.1:/datapool/home/steph
#2
pahll -fstype=nfs,rw 192.168.2.9:/fileserver/home/pahll
frisc -fstype=nfs,rw 192.168.2.9:/fileserver/home/frisc
meuse -fstype=nfs,rw 192.168.2.9:/fileserver/home/meuse
#3
zette -fstype=nfs,rw 192.168.3.1:/mnt/data/user_home/zette
klaku -fstype=nfs,rw 192.168.3.1:/mnt/data/user_home/klaku
#4
dietr -fstype=nfs,rw 192.168.4.1:/mnt/myraid/vm1/home/home/dietr
tongu -fstype=nfs,rw 192.168.4.1:/mnt/myraid/vm1/home/home/tongu
#5
haugs -fstype=nfs,rw 192.168.5.4:/fs/home_dirs/haugs
prinz -fstype=nfs,rw 192.168.5.4:/fs/home_dirs/prinz
#6
stoec -fstype=nfs,rw 192.168.6.11:/home/stoec
wittm -fstype=nfs,rw 192.168.6.11:/home/wittm
#7
grote -fstype=nfs,rw 192.168.7.1:/mnt/md0/vmpsateam07-20/home/grote
deike -fstype=nfs,rw 192.168.7.1:/mnt/md0/vmpsateam07-20/home/deike
#9
wothg -fstype=nfs,rw 192.168.9.5:/home/wothg
songl -fstype=nfs,rw 192.168.9.5:/home/songl
```

We didn't include our own (team 10) home directories since they are local to this machine and they already have permanent, static mountpoints via fstab.

Include this configuration in the main configuration file, `/etc/auto.master`, by appending `+auto.homes`. Furthermore, add the following line to `auto.master` to specify the correct location for the home directories. The names at the start of the lines in `auto.homes` will be appended to this path:

```
/remote_homes     /etc/auto.homes
```

We decided to separate the remote home directories from our own by putting them in their own direcotry, `/remote_homes`.

Finally, restart autofs with the command `sudo systemctl restart autofs`.

You can now access the remote home directories (they won't be displayed until you try to access them).

Testing

For this worksheet, we wrote two separate tests. The first is supposed to be run on the fileserver (VM1), it checks whether the remote home directories, the database files and the home directories of this machine are in order.

```
#!/bin/bash

failed_tests=0

fail() {
    ((failed_tests++))
    echo "FAIL $@"
}

ok() {
    echo "OK $@"
}

# checks if the directory contains files
non_empty() {
    if [ -z "$(sudo ls -A $@)" ]; then
        fail "$@ is empty."
    else
        ok "$@ contains data."
    fi
}

dir_exists() {
    if [ -d "$@" ]; then
        ok "$@ exists."
    else
        fail "$@ not found."
    fi
}

# checks if certain directory exists and contains something
dir_exists_not_empty() {
    if [ -d "$@" ]; then
        ok "$@ exists."
        non_empty "$@"
    else
        fail "$@ not found."
    fi
}

# checks if certain directory exists
ping_dir_exists() {
    if ping -W 0.5 -c 2 "$1" &> /dev/null; then
        dir_exists "$2"
    else
        fail "$1 not reachable"
    fi
}
```

```
dir_exists_not_empty /home

dir_exists_not_empty /var/lib/postgresql


dir_exists /remote_homes
# just a few samples
ping_dir_exists 192.168.1.1 /remote_homes/schoe
ping_dir_exists 192.168.9.5 /remote_homes/songl
ping_dir_exists 192.168.3.1 /remote_homes/klaku
ping_dir_exists 192.168.7.35 /remote_homes/deike



echo "Tests failed: $failed_tests"
```

The second test is located on VM5, the webserver. It checks whether the files are reachable from outside the fileserver, whether SMV works and also verifies if the web server data is accessible. (We also recommend running the original webserver test to make sure all is well on top of this.)

```
#!/bin/bash

# < same helpers as other test>

smb_login() {
    # -L makes it list the shares, without it you don't get output
    if smbclient -L //192.168.10.1/$2 -U "$1"%"psa" >/dev/null 2>&1; then
        ok "SMB $1 can log in and access $2."
    else
        fail " SMB $1 can't log in"
    fi
}


smb_denied() {
    # -L makes it list the shares, without it you don't get output
    if smbclient -L //192.168.10.1/$2 -U "$1"%"psa" >/dev/null 2>&1; then
        ok "SMB $1 can't access $2."
    else
        fail " SMB $1 can access $2 even though it shouldn't"
    fi
}

dir_exists_not_empty /home

dir_exists_not_empty /home/schoe
dir_exists_not_empty /home/horva
dir_exists_not_empty /home/horva/.cgi-bin
dir_exists_not_empty /home/horva/.html-data

dir_exists_not_empty /var/www
dir_exists_not_empty /var/www/html
dir_exists_not_empty /var/www/nextcloud
```

```
dir_exists_not_empty /var/www/other_html
dir_exists_not_empty /var/www/vmpsateam10-05_html
dir_exists_not_empty /var/www/www_html


smb_login "horva" "home_dirs"
smb_login "grote" "home_dirs"
smb_login "meuse" "home_dirs"

# webserver smb
smb_login "horva" "www"
# users outside team10 get denied bc they lack appropriate file permissions
smb_denied "grote" "www"

echo "Tests failed: $failed_tests"
```