

Cahier de TPs

Linux - Administration avancée

jour 3

François MICAUX - ACTILIS - 26.10

Table des matières

JOUR 3

Module 9 Le noyau : paramètres de boot et sysctl 3

Module 10 Booter autrement 6

Module 11 Démarrage du système et GRUB 2 12

Module 12 systemd : démarrage et services 15

Module 13 Les logs 19

Module 9

Le noyau : paramètres de boot et sysctl

TP : interroger le noyau, la ligne de boot, /proc et sysctl

Objectif et machine

Interroger le **noyau en place**, ajouter un **paramètre de boot**, puis lire, modifier et **rendre durables** des paramètres de `/proc/sys` avec `sysctl`.

La machine

Une VM AlmaLinux 10, avec un compte qui dispose de `sudo`. Tout se fait en SSH. Une étape demande un redémarrage : la session SSH est à rouvrir ensuite.

```
$ ssh admin@vm3-ks
```

> ⚠ Paramètres du noyau
>
> Un paramètre mal choisi change le comportement de toute la machine. Ce TP ne touche qu'à des valeurs sans risque et les remet à leur état initial à l'étape 8.

Étape 1 : Interroger le noyau

```
$ uname -r
uname -a
cat /proc/version
journalctl -k -b | head -3
```

Écrire un message dans le *Kernel Ring Buffer*, puis le retrouver :

```
$ echo "Salut du TP" | sudo tee /dev/kmsg
sudo dmesg | tail -2
journalctl -k -n 2 --no-pager
```

Questions :

1. Quelle version du noyau tourne ? Où se lit-elle en dehors de `uname` ?
2. Quelle commande donne les messages du noyau au-delà du *ring buffer* ?
3. Le message « Salut du TP » est-il visible dans `dmesg` et dans `journalctl -k` ?

Étape 2 : La ligne de commande du noyau

```
$ cat /proc/cmdline
cat /proc/sys/kernel/printk
```

Le premier chiffre de `kernel/printk` est le niveau de messages affichés sur la console. Ajouter le paramètre `loglevel=5` à tous les noyaux installés, puis vérifier :

```
$ sudo grubby --update-kernel=ALL --args="loglevel=5"
sudo grubby --info=DEFAULT | grep args
```

La ligne de `/proc/cmdline` n'a pas encore changé : elle décrit le démarrage **en cours**.

Questions :

1. Quels paramètres la ligne de commande contient-elle, et que fait `ro` ?
2. Le paramètre ajouté est-il visible dans `/proc/cmdline` tout de suite ? Pourquoi ?

Étape 3 : Redémarrer et vérifier

```
$ sudo systemctl reboot
```

Se reconnecter après une minute, puis :

```
$ cat /proc/cmdline
cat /proc/sys/kernel/printk
```

Questions :

1. Le paramètre est-il présent ? Quelle valeur a pris le premier chiffre de `kernel/printk` ?

Étape 4 : /proc

```
$ ls -l /proc/$$/cwd
ls -l /proc/meminfo
grep -E 'MemTotal|MemAvailable' /proc/meminfo
ls /proc | grep -c '^[0-9]'
```

Lire puis modifier un paramètre **directement** dans `/proc/sys` :

```
$ cat /proc/sys/net/ipv4/ip_forward
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
sysctl net.ipv4.ip_forward
```

Questions :

1. Quelle est la taille des fichiers de `/proc`, et que représentent les répertoires numériques ?
2. Le routage IPv4 est-il activé à la fin ? Résiste-t-il à un redémarrage ?

Étape 5 : Sysctl

Remettre le routage à `0`, essayer d'autres paramètres, et lister les classes :

```
$ sudo sysctl -w net.ipv4.ip_forward=0
sysctl vm.swappiness
sudo sysctl -w vm.swappiness=10
```

Questions :

1. Quelle était la valeur de `vm.swappiness`, et quel est son effet ?

```
$ sudo sysctl -w kernel.pid_max=131072
sudo sysctl -w kernel.pid_max=4194304
sudo sysctl -w kernel.osrelease=8.0
```

Questions :

2. Pourquoi `kernel.osrelease` est-il refusé, même avec `sudo` ?

```
$ sysctl -a 2>/dev/null | cut -d. -f1 | sort -u
sysctl -a 2>/dev/null | wc -l
```

Questions :

3. Quelles classes de paramètres existent, et laquelle est la plus peuplée ?

Étape 6 : Cache et surcharge mémoire

Vider les caches et mesurer :

```
$ free -m | head -2
sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
free -m | head -2
```

Relever les limites d'allocation :

```
$ grep -E 'MemTotal|SwapTotal|CommitLimit|Committed_AS' /proc/meminfo
sysctl vm.overcommit_memory vm.overcommit_ratio
```

Questions :

1. Quelle colonne de `free` a diminué après `drop_caches` ?
2. Avec `vm.overcommit_memory=2` et `vm.overcommit_ratio=50`, quelle limite d'allocation le noyau applique-t-il ? La retrouver dans `CommitLimit`.

Étape 7 : Rendre durable

Un fichier par sujet, dans `/etc/sysctl.d`. Créer `/etc/sysctl.d/90-tp.conf` avec un éditeur lancé en root (*`sudo`*) :

```
# TP : routage IPv4 et swap réduit
net.ipv4.ip_forward = 1
vm.swappiness = 10
```

Puis :

```
$ sysctl net.ipv4.ip_forward
sudo sysctl --system | tail -4
sysctl net.ipv4.ip_forward vm.swappiness
```

Redémarrer pour vérifier la persistance :

```
$ sudo systemctl reboot
sysctl net.ipv4.ip_forward vm.swappiness
```

Questions :

1. Que valait `net.ipv4.ip_forward` juste après la création du fichier, avant `sysctl --system` ?
2. Les valeurs résistent-elles au redémarrage ?

Étape 8 : Remettre à l'état initial

```
$ sudo rm /etc/sysctl.d/90-tp.conf
sudo sysctl -w net.ipv4.ip_forward=0 vm.swappiness=60
sudo grubby --update-kernel=ALL --remove-args="loglevel=5"
sudo grubby --info=DEFAULT | grep args
```

> **À savoir**
>
> Si `tuned` est installé et actif, son profil (par exemple `virtual-guest`) pose aussi des valeurs comme `vm.swappiness`. Un fichier de `/etc/sysctl.d` n'est pas forcément la dernière couche appliquée : `sysctl vm.swappiness` donne la valeur réelle.

Module 10

Booter autrement

TP : installer une VM par PXE, sauvegarder et cloner une VM avec ReaR sur NFS

Objectif et machines

Deux mises en oeuvre sur le réseau `default` de KVM (`192.168.122.0/24`) :

1. **Installer une VM par PXE**, avec le fichier kickstart du TP précédent.
2. **Sauvegarder une VM avec ReaR sur NFS** (hôte), puis la restaurer sur une VM vierge : un clone.

Les machines

Machine	Rôle
hôte AlmaLinux 9, `192.168.122.1`	KVM, DHCP/TFTP du réseau `default`, serveur NFS
`vm1-ks`	sert le fichier kickstart (TP précédent)
`vm-pxe`	VM vierge, installée par PXE
`vm3-ks`	**source** de la sauvegarde ReaR, installée par kickstart au TP précédent, **sans** ses disques supplémentaires (étape 7)
`vm-restore`	VM vierge de 10 Go, **cible** de la restauration

> **Le `dnsmasq` du réseau `default`**
>
> Ce réseau est déjà servi par un `dnsmasq` lancé par `libvirt`. Le TP active son TFTP et son BOOTP par le XML du réseau, sans second serveur DHCP.

Étape 1 : Activer TFTP et BOOTP sur le réseau `default`

Éteindre les VM, car le redémarrage du réseau leur coupe leur connexion :

```
# sudo virsh shutdown vm1-ks
sudo virsh shutdown vm3-ks
sudo virsh net-edit default
```

Dans la section `<ip>`, ajouter `<tftp>` et `<bootp>` :

```
<ip address='192.168.122.1' netmask='255.255.255.0'>
  <tftp root='/var/lib/tftpboot'/>
  <dhcp>
    <range start='192.168.122.2' end='192.168.122.254'/>
    <bootp file='pxelinux/pxelinux.0'/>
  </dhcp>
</ip>
```

Redémarrer le réseau, puis rallumer les VM :

```
# sudo virsh net-destroy default
sudo virsh net-start default
sudo virsh start vm1-ks
sudo virsh start vm3-ks
```

> **Équivalents `dnsmasq`**
>
> `<tftp root=...>` : `enable-tftp` et `tftp-root`. `<bootp file=...>` : `dhcp-boot`, annoncé à tous les clients du réseau.

Étape 2 : Les fichiers de démarrage

Sur l'hôte. `pxelinux.0` vient du paquet `syslinux-tftpboot` qui le dépose dans `/tftpboot/`. Le noyau et l'initrd de l'installateur se prennent dans le dépôt d'AlmaLinux 10 :

```
# sudo dnf install -y syslinux-tftpboot
sudo mkdir -p /var/lib/tftpboot/pxelinux/pxelinux.cfg \
  /var/lib/tftpboot/pxelinux/images/alma10
sudo cp /tftpboot/pxelinux.0 /tftpboot/ldlinux.c32 \
  /var/lib/tftpboot/pxelinux/
```

```
# D=/var/lib/tftpboot/pxelinux/images/alma10
U=https://repo.almalinux.org/almalinux/10/BaseOS/x86_64/os/images/pxeboot
sudo curl -o $D/vmlinuz $U/vmlinuz
sudo curl -o $D/initrd.img $U/initrd.img
sudo restorecon -R /var/lib/tftpboot
```

> **Pourquoi `pxelinux`**
>
> Les machines virtuelles du TP démarrent en BIOS : le chargeur est `pxelinux.0`. Une VM UEFI recevrait `BOOTX64.EFI` et un `grub.cfg` (voir slide « PXE : fichiers de boot et menus »).

Étape 3 : Le menu et le serveur kickstart

Sur l'hôte, le menu de démarrage. `IP1` est l'adresse de `vm1-ks` (`sudo virsh domifaddr vm1-ks`), qui peut avoir changé. Créer `/var/lib/tftpboot/pxelinux/pxelinux.cfg/default` avec un éditeur lancé en root (*`sudo`*) :

```
default alma10
label alma10
  kernel images/alma10/vmlinuz
  append initrd=images/alma10/initrd.img ip=dhcp
inst.repo=https://repo.almalinux.org/almalinux/10/BaseOS/x86_64/os/
inst.ks=http://IP1:8080/ks.cfg
```

Dans `vm1-ks`, remettre en service le serveur HTTP du TP précédent, arrêté avec la VM :

```
# sudo firewall-cmd --add-port=8080/tcp
```

```
$ python3 -m http.server 8080 --directory /srv/kickstart
```

> **Conteneur `httpd`**
>
> Si le TP précédent s'est terminé sur le conteneur, `podman start ks-httpd` remplace la commande Python.

Étape 4 : Démarrer la VM par le réseau

Sur l'hôte :

```
# sudo virt-install --name vm-pxe --memory 3072 --vcpus 2 --cpu \
    host-passthrough \
    --disk size=10,bus=virtio --network network=default \
    --pxe --os-variant almalinux9 --boot hd,network --noautoconsole
```

Ouvrir la console pour suivre le démarrage :

```
$ virt-viewer vm-pxe
```

La séquence : le BIOS de la VM obtient une adresse et le nom `pxelinux/pxelinux.0`, le télécharge par TFTP, puis `pxelinux` charge le noyau et l'initrd. L'installateur lit alors le kickstart sur `vm1-ks` et installe sans poser de question.

> ⚠ **Ordre de démarrage**
>
> `hd,network` essaie le disque d'abord : vide au départ, il laisse la main au réseau, puis il démarre le système installé sans relancer l'installation. Si la VM se réinstalle à son redémarrage, contrôler `sudo virsh dumpxml vm-pxe | grep boot`.

> ℹ **Mémoire de la VM**
>
> L'image de l'installateur se charge en mémoire : 3 Go de RAM laissent de la marge.

Étape 5 : Vérifier la VM installée

Relever l'adresse de `vm-pxe` et s'y connecter :

```
# sudo virsh domifaddr vm-pxe
```

```
$ ssh admin@IP-PXE 'sudo whoami; rpm -q bash-completion; hostname'
```

Questions :

1. Quel protocole a servi `pxelinux.0`, et lequel a servi le fichier kickstart ?
2. Quelle partie du menu dit à l'installateur de ne poser aucune question ?

Étape 6 : Le partage NFS sur l'hôte

```
# sudo dnf install -y nfs-utils
sudo mkdir -p /srv/rear
```

Déclarer le partage : créer `/etc/exports.d/rear.exports` avec un éditeur lancé en root (*`sudo`*) :

```
≡ /srv/rear 192.168.122.0/24(rw,sync,no_root_squash)
```

Puis l'activer :

```
# sudo systemctl enable --now nfs-server
sudo exportfs -v
```

Autoriser NFS depuis le réseau des VM :

```
# sudo firewall-cmd --zone=libvirt --add-service=nfs --add-service=mountd \
    --add-service=rpc-bind
```

> ⚠ `no_root_squash`
>
> ReaR s'exécute en `root` et doit écrire en `root` sur le partage : l'option laisse ce compte inchangé côté serveur. Le partage est limité au réseau `192.168.122.0/24`.

> ℹ Règle temporaire
>
> Sans `--permanent`, le service reste ouvert jusqu'au prochain rechargement du pare-feu. Cela suffit pour ce TP.

Étape 7 : Retirer les disques supplémentaires de la source

`vm3-ks` a reçu deux disques de 10 Go au TP « Stockage ». `vm-restore` n'a qu'un disque : ReaR enregistre le plan de disques de la source et recrée ce plan à la restauration, des disques en plus feraient échouer `rear recover`. Sur l'**hôte**, relever les fichiers des disques puis les détacher de la VM (les fichiers restent sur l'hôte) :

```
# sudo virsh domblklist vm3-ks
sudo virsh detach-disk vm3-ks vdb --persistent
sudo virsh detach-disk vm3-ks vdc --persistent
sudo virsh domblklist vm3-ks
```

Noter les chemins des deux fichiers de `vdb` et `vdc` : l'étape 13 les rattache.

> ✎ Disques vierges
>
> Le nettoyage du TP « Stockage » les a laissés vierges : rien n'est perdu en les détachant, et rien n'est monté dessus.

Étape 8 : Préparer la VM source

Dans `vm3-ks` (connexion `ssh admin@IP3`) :

```
# sudo dnf install -y rear nfs-utils
```

Créer `/etc/rear/local.conf` avec un éditeur lancé en root (*`sudo`*) :

```
OUTPUT=ISO
BACKUP=NETFS
BACKUP_URL=nfs://192.168.122.1/srv/rear
BACKUP_OPTIONS="nfsvers=3,nolock"
ISO_DEFAULT="manual"
```

Laisser un fichier repère, qu'on retrouvera après la restauration, et contrôler que le partage est visible :

```
$ echo "Sauvegardé par ReaR le $(date)" > ~/marqueur-rear.txt
showmount -e 192.168.122.1
```

Directive	Rôle
`BACKUP_URL=nfs://...`	l'archive `tar` **et** l'ISO sont rangées sur le partage
`BACKUP_OPTIONS`	options de montage NFS, aussi utilisées par le système de secours
`ISO_DEFAULT="manual"`	on se connecte et on lance soi-même `rear recover` (le défaut démarre le disque local, inutile sur une machine vierge)

Étape 9 : Sauvegarder

Dans `vm3-ks` :

```
# sudo rear -v mkbackup
```

La commande analyse les disques, prépare le système de secours, archive le système vers le partage, puis fabrique l'ISO.

Sur l'hôte, lister le partage :

```
# sudo ls -lh /srv/rear/*/
```

Questions :

1. Quel fichier contient les données, lequel est amorçable ?
2. Pourquoi l'ISO est-elle petite ?

Étape 10 : Démarrer la machine vierge

Sur l'hôte, copier l'ISO dans le dossier des images de `libvirt`, où `qemu` a le droit de la lire, puis créer `vm-restore` :

```
# sudo cp /srv/rear/*/rear-*.iso /var/lib/libvirt/images/rear-vm3.iso
sudo virt-install --name vm-restore --memory 2048 --vcpus 2 --cpu \
  host-passthrough \
  --disk size=10,bus=virtio --network network=default \
  --cdrom /var/lib/libvirt/images/rear-vm3.iso --os-variant almalinux9
```

Dans la console, le menu de ReaR s'affiche : choisir l'entrée **Recover** suivie du nom d'hôte.

> ⚠ Même taille de disque, au moins
>
> `rear recover` recrée le partitionnement sur le disque qu'il trouve. Un disque plus petit que celui de la source fait échouer la restauration.

Étape 11 : Restaurer

Dans la console de `vm-restore`, ouvrir une session en `root` (aucun mot de passe), puis :

```
# rear -v recover
```

Le système de secours obtient une adresse par DHCP sur le réseau `default`, monte le partage de l'hôte, recrée les partitions, `rootvg` et `lvroot`, restaure l'archive, puis réinstalle le chargeur de démarrage. Si une confirmation est demandée (correspondance des disques), la proposition par défaut convient ici.

À la fin, éteindre la VM, retirer l'ISO du lecteur et la rallumer :

```
# poweroff
```

```
# sudo virsh domblklist vm-restore
sudo virsh change-media vm-restore <cible-du-cdrom> --eject --config
sudo virsh start vm-restore
```

> ℹ Réétiquetage SELinux
>
> ReaR laisse un fichier `/.autorelabel` : au premier démarrage, SELinux réétiquette tous les fichiers, ce qui rallonge ce démarrage de quelques minutes.

Étape 12 : Vérifier le clone

Relever l'adresse de la nouvelle machine (sa carte réseau est neuve, l'adresse diffère) et s'y connecter avec le compte `admin` :

```
# sudo virsh domifaddr vm-restore
```

```
$ ssh admin@IP-RESTORE
cat ~/marqueur-rear.txt
hostname
lsblk
```

```
# sudo vgs
sudo lvs
```

Constater :

- le fichier repère est présent, avec sa date d'origine,
- le nom d'hôte et le partitionnement sont ceux de `vm3-ks`,
- `bash-completion` et `sudo` sans mot de passe, ajoutés par le kickstart, sont toujours là.

Questions :

1. Quelle commande a restauré les données, et d'où les a-t-elle lues ?
2. Qu'est-ce qui diffère entre la source et le clone, malgré la restauration complète ?

> **Pour garder les deux machines**
>
> Renouveler le nom d'hôte avec `sudo hostnamectl set-hostname <nom>`. `/etc/machine-id` et les clés SSH sont aussi ceux de la source.

Étape 13 : Nettoyer

Sur l'hôte :

```
# sudo virsh destroy vm-restore
sudo virsh undefine vm-restore --remove-all-storage
sudo virsh destroy vm-pxe
sudo virsh undefine vm-pxe --remove-all-storage
sudo rm /var/lib/libvirt/images/rear-vm3.iso
sudo rm -r /srv/rear/*
```

Rattacher à `vm3-ks` ses deux disques supplémentaires, que les TP suivants réutilisent (les chemins relevés à l'étape 7) :

```
# sudo virsh attach-disk vm3-ks <fichier-de-vdb> vdb --targetbus virtio \
  --persistent
sudo virsh attach-disk vm3-ks <fichier-de-vdc> vdc --targetbus virtio \
  --persistent
sudo virsh domblklist vm3-ks
```

Le partage NFS, le XML du réseau `default` et les fichiers de `/var/lib/tftpboot` peuvent rester en place pour la suite.

Module 11

Démarrage du système et GRUB 2

TP : modifier une entrée à la volée, démarrer à la main, retrouver un mot de passe

Objectif et machine

Observer ce que GRUB 2 charge, **changer le démarrage à la volée**, **démarrer sans menu** depuis sa ligne de commande, puis **retrouver l'accès** à un compte dont le mot de passe est perdu.

La machine

La VM `vm3-ks` (ou toute VM AlmaLinux 10 avec `/` sur LVM). GRUB 2 se manipule dans la **console** de la VM, pas en SSH : depuis le poste de travail,

```
$ virt-viewer vm3-ks
```

> ⚠ Le menu passe vite
>
> Le menu GRUB 2 s'affiche quelques secondes au démarrage. Une touche (la flèche vers le bas) arrête le compte à rebours.

Étape 1 : Observer

Dans la VM, noter le **mode de démarrage**, la **ligne de commande du noyau** (elle sert à l'étape 3) et le contenu de `/boot` :

```
$ [ -d /sys/firmware/efi ] && echo UEFI || echo BIOS
```

Questions :

1. Quelle commande indique si la machine a démarré en BIOS ou en UEFI ?

```
$ cat /proc/cmdline
ls /boot
ls /boot/loader/entries
```

Questions :

2. Quels fichiers de `/boot` GRUB 2 charge-t-il pour démarrer le noyau ?

```
# grubby --info=ALL
```

Questions :

3. Combien d'entrées de menu la machine propose-t-elle, et où sont-elles décrites ?

> UEFI
>
> Sur une VM UEFI, ce TP reste valable. Les chemins de l'ESP sont à la slide « UEFI : l'ESP, shim et GRUB ».

Étape 2 : Modifier une entrée à la volée

1. Redémarrer la VM : `sudo reboot`, puis arrêter le compte à rebours dans la console.

2. Sélectionner l'entrée par défaut, touche `e`.
3. À la fin de la ligne `linux`, **retirer** `rhgb quiet`, puis **Ctrl-x**.

Le démarrage affiche alors tous ses messages. Une fois connecté :

```
$ cat /proc/cmdline
```

Puis redémarrer **normalement**, sans toucher à GRUB, et relire `/proc/cmdline`.

Questions :

1. `rhgb` et `quiet` sont-ils présents dans la ligne de commande du premier démarrage ? Et du second ?
2. Les modifications faites avec `e` sont-elles conservées ?

Étape 3 : Démarrer depuis la ligne de commande

1. Redémarrer, arrêter le compte à rebours, puis `c` : l'invite `grub>` s'affiche.
2. Repérer les disques et partitions, puis le contenu de celle qui porte `/boot` :

```
ls
ls (hd0,gpt2)/
```

3. Charger les modules, désigner la partition, puis le noyau et l'initramfs. Les options de `linux` sont celles de `/proc/cmdline` notées à l'étape 1. La touche **Tabulation** complète les noms de fichiers :

```
insmod part_gpt
insmod ext2
set root=(hd0,gpt2)
linux  /vmlinuz-6.12.0-... root=/dev/mapper/rootvg-lvroot ro rd.lvm.lv=rootvg/lvroot
initrd /initramfs-6.12.0-....img
boot
```

> **Numéro de partition et système de fichiers**
>
> Le numéro (`gpt2`) est celui que `ls` a montré avec les fichiers `vmlinuz-...` et `initramfs-...`. Le module correspond au système de fichiers de `/boot` : `/boot` est en ext4 dans ce TP, que GRUB 2 lit avec le module `ext2` (il n'existe pas de module `ext4`). Un `/boot` en XFS demande `insmod xfs`.

Questions :

1. Quelle commande de GRUB 2 lit le contenu d'une partition ?
2. Quelle partie de la ligne `linux` désigne le système de fichiers racine ?

> **À savoir**
>
> `rd.lvm.lv=` demande à l'initramfs d'activer le volume LVM de la racine avant de la monter : sans lui, le noyau ne trouve pas `/` sur un volume logique.

Étape 4 : Un mot de passe perdu

Créer un compte de test, **puis oublier** son mot de passe :

```
# useradd tpgrub
echo 'oubli1' | passwd --stdin tpgrub
```

Redémarrer, arrêter le compte à rebours, `e`, ajouter `rd.break` à la fin de la ligne `linux`, **Ctrl-x**. L'invite de l'initramfs apparaît. Le disque est monté sous `/sysroot`, en lecture seule :

```
# mount -o remount,rw /sysroot
/sysroot/sbin/chroot /sysroot
passwd tpgrub
touch /.autorelabel
exit
exit
```

> **`chroot` du système**
>
> L'initramfs ne contient pas la commande `chroot` : on appelle celle du disque monté, `/sysroot/sbin/chroot`.

Le démarrage reprend, et reste plus long (réétiquetage SELinux). Vérifier le nouveau mot de passe depuis le compte `admin` :

```
$ su - tpgrub
```

Questions :

1. Quel paramètre arrête le démarrage dans l'initramfs ?
2. Où le disque de la machine est-il monté à ce moment ?
3. À quoi sert `touch /.autorelabel` ?

> **Autre méthode**
>
> `init=/bin/bash` à la place de `rd.break` donne un shell qui remplace `systemd` (voir slide « Reprendre la main : `init=/bin/bash` »). À essayer ensuite sur un autre compte de test.

Étape 5 : GRUB sans configuration (facultatif)

Mettre de côté le fichier de configuration, puis redémarrer :

```
# mv /boot/grub2/grub.cfg /boot/grub2/grub.cfg.sav
reboot
```

Sans `grub.cfg`, GRUB 2 n'a plus de menu. Démarrer **à la main** comme à l'étape 3, puis, connecté, **régénérer** la configuration :

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

Au redémarrage suivant, le menu est revenu.

> **Seulement en BIOS**
>
> Sur une VM UEFI, GRUB lit d'abord un petit fichier de l'ESP qui renvoie à `/boot/grub2/grub.cfg` : le comportement diffère, passer cette étape.

> **Nettoyage**
>
> Supprimer le compte de test et la sauvegarde : `userdel -r tpgrub`, `rm /boot/grub2/grub.cfg.sav`.

Module 12

systemd : démarrage et services

TP : cibles, premier service, surcharge et unité en échec

Objectif et machine

Lire l'**état** de systemd et sa **cible par défaut**, écrire puis activer **son propre service**, le **surcharger** sans toucher à son fichier, et réparer une **unité en échec**.

La machine

Une VM AlmaLinux 10, avec un compte qui dispose de `sudo`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

> ⚠ `isolate` en SSH
>
> `systemctl isolate rescue.target` arrête les services réseau et coupe la session SSH. Ce TP ne change pas de cible courante : à réserver à la console de la VM.

Étape 1 : Observer systemd

```
$ systemctl --version | head -1
systemctl get-default
systemctl list-units --type=target
systemctl is-system-running ; echo "code retour : $?"
systemctl status | head -6
```

Les fichiers de la cible par défaut :

```
$ ls -l $(find {/etc,/usr/lib}/systemd/system -name default.target)
```

Questions :

1. Quelle version de `systemd` la VM utilise-t-elle ?
2. Quelle est la cible par défaut, et vers quoi pointe le lien de `/usr/lib/systemd/system` ?
3. Quel code retour donne `is-system-running` quand le système est `running` ? Et quand il est `degraded` (étape 6) ?

Étape 2 : Changer la cible par défaut

Poser une autre cible par défaut, regarder les liens, puis revenir :

```
$ sudo systemctl set-default graphical.target
ls -l /etc/systemd/system/default.target
systemctl get-default
```

Questions :

1. Dans quel répertoire `set-default` écrit-il, et que contient le fichier créé ?

```
$ sudo systemctl set-default multi-user.target
  ls -l /etc/systemd/system/default.target
```

Questions :

2. Le fichier de `/usr/lib/systemd/system` a-t-il été modifié ?

Étape 3 : Écrire un service

Un script qui écrit une ligne toutes les 5 secondes, rangé dans `/usr/local/bin`. Créer `/usr/local/bin/mon-script` avec un éditeur lancé en root (`sudo`) :

```
#!/usr/bin/bash
while true; do
    echo "monservice : $(date +%T)"
    sleep 5
done
```

Le rendre exécutable :

```
$ sudo chmod +x /usr/local/bin/mon-script
  ls -Z /usr/local/bin/mon-script
```

L'unité : créer `/etc/systemd/system/monservice.service` avec un éditeur lancé en root (`sudo`) :

```
[Unit]
Description=Mon service

[Service]
ExecStart=/usr/local/bin/mon-script

[Install]
WantedBy=multi-user.target
```

Puis la vérifier :

```
$ systemd-analyze verify /etc/systemd/system/monservice.service
  systemctl cat monservice.service
```

Étape 4 : Activer et démarrer

```
$ sudo systemctl enable monservice.service
  systemctl is-enabled monservice.service
  systemctl is-active monservice.service
```

Questions :

1. Quel état affichent `is-enabled` et `is-active` avant le `start` ?

```
$ sudo systemctl start monservice.service
  systemctl status monservice.service
  journalctl -u monservice.service -n 5
  ls -l /etc/systemd/system/multi-user.target.wants/
```

Questions :

2. Quelle directive fait démarrer le service avec la cible `multi-user.target` ?
3. Que crée `enable`, et où ?

> **À savoir**
>
> Le script est étiqueté `bin_t` : SELinux autorise `systemd` à le lancer. Le même script posé dans un répertoire personnel serait refusé (voir slide « Installer un nouveau service »).

Étape 5 : Surcharger l'unité

Ajouter une relance automatique **sans modifier** `monservice.service`. L'éditeur s'ouvre sur un fichier vide : y écrire les trois lignes, enregistrer, quitter.

```
$ sudo systemctl edit monservice.service
```

```
[Service]
Restart=always
RestartSec=2
```

```
$ ls -l /etc/systemd/system/monservice.service.d/
systemctl cat monservice.service
```

Tuer le processus et regarder ce qui se passe :

```
$ systemctl show -p MainPID --value monservice.service
sudo kill $(systemctl show -p MainPID --value monservice.service)
sleep 4
systemctl show -p MainPID --value monservice.service
journalctl -u monservice.service -n 6
```

Questions :

1. Quel fichier `systemctl edit` a-t-il créé, et dans quel répertoire ?
2. Que montre `systemctl cat` en plus du fichier d'origine ?
3. Le numéro du processus principal a-t-il changé après le `kill` ? Que dit le journal ?

> **À savoir**
>
> Une modification d'un fichier d'unité à la main demande `sudo systemctl daemon-reload`. Tant qu'elle n'est pas faite, `systemctl status` signale `changed on disk`. `systemctl edit` recharge de lui-même.

Étape 6 : Une unité en échec

Une unité dont la commande échoue. Créer `/etc/systemd/system/panne.service` avec un éditeur lancé en root (*`sudo`*) :

```
[Unit]
Description=Service en échec

[Service]
Type=oneshot
ExecStart=/usr/bin/false
```

Puis la lancer :

```
$ sudo systemctl daemon-reload
sudo systemctl start panne.service
```

Relire l'état du système :

```
$ systemctl is-system-running ; echo "code retour : $?"
systemctl status | head -5
systemctl --failed
systemctl status panne.service
```

Remettre à zéro, puis relire l'état :

```
$ sudo systemctl reset-failed panne.service
systemctl is-system-running
```

Questions :

1. Quel état et quel code retour donne `is-system-running` avec l'unité en échec ?
2. Quelle ligne de `systemctl status` indique le nombre d'unités en échec ?
3. Quelle commande remet l'état à zéro une fois l'unité réparée ?

Étape 7 : Dépendances et nettoyage

Les dépendances de l'unité et d'une cible :

```
$ systemctl list-dependencies monservice.service
systemctl list-dependencies multi-user.target | head
```

Tout défaire, dans l'ordre :

```
$ sudo systemctl disable --now monservice.service
sudo rm -r /etc/systemd/system/monservice.service \
  /etc/systemd/system/monservice.service.d
sudo rm /etc/systemd/system/panne.service /usr/local/bin/mon-script
sudo systemctl daemon-reload
systemctl is-system-running
```

Questions :

1. `monservice.service` figure-t-il encore dans `list-dependencies multi-user.target` après `disable` ?

Module 13

Les logs

TP : émettre, lire, filtrer et conserver le journal

Objectif et machine

Suivre un message de bout en bout : l'**émettre**, le **retrouver** dans `journald`, le **filtrer** par ses champs, rendre le journal **persistant** puis mesurer et limiter sa **taille**.

La machine

Une VM AlmaLinux 10 en installation minimale, avec un compte qui dispose de `sudo`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

Le fil rouge

Une application fictive, `TP`, émet des messages sur la facility `local0`. On la suit dans le journal.

> **Un redémarrage au programme**
>
> L'étape 4 demande un redémarrage de la VM pour vérifier la persistance du journal.

Étape 1 : Émettre et retrouver un message

```
$ logger -p local0.info -t TP 'premier message'
echo 'ligne reçue sur stdin' | logger -p local0.debug -t TP
journalctl -t TP
journalctl -t TP -p info
```

Questions :

1. Combien de messages `journalctl -t TP` affiche-t-il, et combien avec `-p info` ? Pourquoi ?

```
$ journalctl -t TP -p info -n 1 -o verbose | grep -E 'PRIORITY|SYSLOG_'
```

Questions :

2. Quelles valeurs portent `PRIORITY` et `SYSLOG_FACILITY` ? Quel `PRI` en déduit-on ?

```
$ ls -l /dev/log
```

Questions :

3. Vers quoi pointe `/dev/log` ?

Étape 2 : Filtrer le journal

```
$ journalctl --since "10 minutes ago" -p warning
journalctl -u sshd -n 5
journalctl -k -n 5
journalctl -b -p err
journalctl -g 'Accepted'
```

Questions :

1. Quelle option sélectionne une unité ? Les messages du noyau ? Une plage horaire ? Une recherche dans le texte ?
2. Que renvoie `-p warning` : les avertissements seuls, ou aussi les messages plus graves ?

> **Combiner les filtres**
>
> Les filtres se cumulent : `journalctl -u sshd -p warning --since today`.

Étape 3 : Les champs et les transports

Chaque champ du journal peut servir de filtre :

```
$ journalctl _UID=$(id -u) -n 2
journalctl SYSLOG_IDENTIFIER=TP _TRANSPORT=syslog
journalctl -F _TRANSPORT
```

Questions :

1. Quels transports `journalctl -F _TRANSPORT` liste-t-il, et lequel a servi pour `logger` ?

```
$ echo "via stdout" | systemd-cat -t TP -p warning
journalctl -t TP -n 1 -o verbose | grep -E '_TRANSPORT|PRIORITY'
```

Questions :

2. Quelle priorité et quel transport le message de `systemd-cat` porte-t-il ?

```
$ journalctl -t TP -n 1 -o json-pretty | head -8
```

Questions :

3. Quelle option donne le message au format JSON ?

Étape 4 : Le journal est-il persistant ?

```
$ journalctl --list-boots
ls /var/log/journal
```

Questions :

1. Combien de démarrages le journal conserve-t-il ? Le répertoire `/var/log/journal` existe-t-il ?

```
$ systemd-analyze cat-config systemd/journald.conf | grep Storage
```

Questions :

2. Quel mode de stockage s'applique, et que fait-il dans ce cas ?

Étape 5 : Activer, redémarrer, vérifier

Un fichier de surcharge, puis le répertoire du journal :

```
$ sudo mkdir -p /etc/systemd/journald.conf.d
```

Créer `/etc/systemd/journald.conf.d/10-local.conf` avec un éditeur lancé en root (*`sudo`*) :

```
[Journal]
Storage=persistent
```

Puis créer le répertoire, relancer le service et redémarrer :

```
$ sudo mkdir /var/log/journal
sudo systemctl restart systemd-journald
sudo journalctl --flush
journalctl --disk-usage
sudo systemctl reboot
```

Après le redémarrage :

```
$ journalctl --list-boots
journalctl -t TP -b -1
```

Questions :

1. Combien de démarrages `--list-boots` liste-t-il maintenant ?
2. Le message `premier message` de l'étape 1 est-il encore accessible ?

Étape 6 : L'espace occupé

```
$ journalctl --disk-usage
systemd-analyze cat-config systemd/journald.conf | grep -E 'Max|KeepFree'
```

Questions :

1. Quels paramètres de `journald.conf` limitent l'espace ?
2. Quel mécanisme fait tourner le journal : `journald` lui-même ou `logrotate` ?

```
$ sudo journalctl --vacuum-size=20M
journalctl --disk-usage
```

Questions :

3. Quelle place le journal occupe-t-il avant et après `--vacuum-size` ?

Étape 7 : Bilan

Le journal reste persistant : ce réglage peut rester en place.

Questions

1. Comment émettre un message depuis le shell, et comment le retrouver par un champ précis ?
2. Où le journal est-il conservé par défaut, et que faut-il pour qu'il survive à un redémarrage ?
3. Qui limite la taille du journal : `journald` ou `logrotate` ?