

Cahier de TPs

Linux - Administration avancée

François MICAUX - ACTILIS - 26.10

Table des matières

JOUR 1

Module 1 Installation d'AlmaLinux 10 3

Module 2 Installation automatisée avec Kickstart 7

Module 3 Mettre à jour le système 12

Module 4 Maîtriser la compilation des logiciels 18

JOUR 2

Module 5 Maîtriser la production de packages RPM 25

Module 6 Stockage, Swap, Systèmes de fichiers 32

Module 7 Périphériques, modules et udev 40

Module 8 Le noyau : compilation et installation 46

Module 9 Le noyau : paramètres de boot et sysctl 51

JOUR 3

Module 10 Booter autrement 55

Module 11 Démarrage du système et GRUB 2 62

Module 12 systemd : démarrage et services 66

Module 13 Les logs 71

JOUR 4

Module 14 Analyse de la charge et de l'activité système 75

Module 15 Optimiser les performances 80

Module 16 Collecter et grapher des métriques système 86

ANNEXES

Annexe 3 Gestion avancée et centralisation des logs 91

Annexe 7 Intégrité du système 98

Module 1

Installation d'AlmaLinux 10

TP : une installation manuelle, point de départ du kickstart

Objectif et machines

Installer à la main **une première VM AlmaLinux 10**, avec la racine sur LVM. Elle devient, au TP suivant, le **serveur du fichier kickstart** qui installera les autres machines.

Le poste de travail

Le système fourni est un **AlmaLinux 9** avec KVM et `libvirt`. Il héberge les VM et leur réseau `default` (`192.168.122.0/24`, NAT), sur lequel elles se voient entre elles.

La VM `vm1-ks`

2 Go de RAM, 2 cœurs, **un disque virtuel de 10 Go** (`vda`), réseau `default`, accès à Internet.

Partitionnement imposé

Partition	Taille	Type	Format
`/boot`	1 Gio	standard	ext4
`swap`	1 Gio	standard	swap
`/`	5 Gio	LV `lvroot` dans le VG `rootvg`	XFS
PV de `rootvg`	**tout le reste du disque**	partition LVM	

`/boot` reste hors LVM : le chargeur de démarrage doit pouvoir le lire sans passer par LVM (voir slide « Partitionnement : Quelles partitions créer ? »).

Étape 1 : Récupérer l'ISO

Sur le poste de travail, télécharger l'ISO **de démarrage** d'AlmaLinux 10 (`boot`, environ 1 Go), qui prend les paquets sur le réseau pendant l'installation, ainsi que le fichier des empreintes :

```
# cd /var/lib/libvirt/images
U=https://repo.almalinux.org/almalinux/10/isos/x86_64
curl -L -o AlmaLinux-10-boot.iso $U/AlmaLinux-10-latest-x86_64-boot.iso
curl -L -O $U/CHECKSUM
```

Contrôler l'empreinte de l'ISO, en cherchant dans `CHECKSUM` la ligne de `AlmaLinux-10-latest-x86_64-boot.iso` :

```
# grep 'AlmaLinux-10-latest-x86_64-boot.iso)' CHECKSUM
sha256sum AlmaLinux-10-boot.iso
```

Les deux empreintes doivent être **identiques**.

Étape 2 : Choisir l'ISO

> ⚠ Choisir l'ISO `x86_64`
>
> Les ISO AlmaLinux 10 existent aussi en variante `x86_64_v2`, pour les processeurs anciens. Avec elle, un kickstart lancé sous KVM peut rester bloqué au partitionnement (signalé sur le forum AlmaLinux). Le fichier `latest-x86_64` ci-dessus convient.

> ℹ Autres ISO
>
> `minimal` (1,6 Go) contient un système de base, `dvd` (10 Go) tous les paquets. Les deux servent à installer sans réseau : le TP garde `boot`.

Étape 3 : Créer la VM

Créer la VM et lancer l'installateur :

```
# sudo virt-install --name vm1-ks --memory 2048 --vcpus 2 --cpu \
    host-passthrough \
    --disk size=10,bus=virtio --network network=default \
    --cdrom /var/lib/libvirt/images/AlmaLinux-10-boot.iso --os-variant \
    almalinux9
```

La console graphique de la VM s'ouvre toute seule.

> ℹ Valeur de `--os-variant`
>
> La base de données de systèmes (`osinfo`) d'un AlmaLinux 9 ne connaît pas forcément `almalinux10` : `almalinux9` donne les mêmes réglages de VM.

Étape 4 : Installer, écrans de base

1. Choisir **Install AlmaLinux 10** au menu de démarrage, puis la langue **Français** et le clavier **Français (AZERTY)**.
2. **Réseau et nom d'hôte** : activer l'interface, nom d'hôte `srv-ks.exemple.lan`.
3. **Source d'installation** : laisser le **miroir le plus proche**, il est détecté sur le réseau.
4. **Sélection des logiciels** : **Installation minimale**, sans module complémentaire.
5. **Utilisateur** : nom `admin`, mot de passe de son choix, case **Faire de cet utilisateur un administrateur** cochée. **Mot de passe root** : compte désactivé.

Étape 5 : Installer, partitionnement

1. **Destination de l'installation** : sélectionner le disque de 10 Go, choisir **Personnalisée**, puis **Terminé**.
2. Dans l'écran de partitionnement, laisser le schéma sur **LVM**.
3. **+** `/boot`, `1 GiB` : type **Partition standard**, système de fichiers **ext4**, **Mettre à jour les paramètres**.
4. **+** `swap`, `1 GiB` : type **Partition standard**.
5. **+** `/`, `5 GiB` : type **LVM**, système de fichiers **XFS**, nom du volume `lvroot`. Pour « Groupe de volumes », **Modifier…** : nom `rootvg`, politique de taille **Aussi grand que possible**.
6. **Terminé**, relire la liste, **Accepter les modifications**, puis **Commencer l'installation**.

> ⚠ **Ordre de création**
>
> Créer les partitions dans l'ordre `/boot`, `swap`, `/` : la partition LVM se place alors en dernier, sur tout l'espace restant.

> ❗ **BIOS : créer d'abord la partition `biosboot`**
>
> Sur une VM en BIOS, AlmaLinux 10 partitionne en GPT, et GRUB n'a alors aucun endroit où s'installer sans une partition `biosboot` (type **BIOS Boot**, taille minimale, sans système de fichiers).
> La créer **avant** la partition LVM de `/` (**+** `biosboot`) : une fois le groupe de volumes réglé sur **Aussi grand que possible**, il occupe tout l'espace restant et l'installation échoue faute de place.

Après la copie, **redémarrer** la VM (l'ISO n'est plus utilisée) et ouvrir une session avec `admin`.

Étape 6 : Contrôler l'installation

```
$ lsblk
findmnt /boot
ip -br address
```

```
# sudo vgs
sudo lvs
```

Questions :

1. Sur quel disque et quelle partition se trouve le PV de `rootvg` ?
2. Quel espace reste libre dans `rootvg` ?
3. Quelle adresse IP la VM a-t-elle reçue ? La noter, elle sert au TP suivant.

> ℹ **Réponses**
>
> Le PV se lit avec `lsblk` ou `pvs`. L'espace libre est la colonne `VFree` de `vgs`. L'adresse est celle de l'interface `enp1s0` (nom donné par KVM, à adapter) dans `192.168.122.0/24`.

Étape 7 : Lire le fichier kickstart déposé

L'installateur a écrit les réponses données dans `/root/anaconda-ks.cfg`, lisible seulement par `root` :

```
# sudo cat /root/anaconda-ks.cfg
```

Repérer, dans ce fichier :

Ce qui a été choisi	Où le trouver
Source des paquets	ligne `url` ou `cdrom`, selon l'ISO et le choix de la source
Disque et partitions	`ignoredisk`, `part`, `volgroup`, `logvol`
Compte `admin`	ligne `user` (mot de passe chiffré, `--iscrypted`)
Compte root désactivé	ligne `rootpw --lock`
Paquets installés	section `%packages`

Questions :

1. Quelle commande décrit le volume logique `lvroot`, et quelle taille porte-t-elle ?
2. Le mot de passe de `admin` apparaît-il en clair ?

> **À savoir**
>
> Ce fichier reflète **cette** installation, avec le nom de la VM et le disque `vda`. Il sert de point de départ : il se copie, s'édite et se rejoue au TP suivant.

Étape 8 : La complétion de `bash`

Vérifier si l'extension de complétion de `bash` est déjà là, le résultat sert au TP suivant :

```
$ rpm -q bash-completion
```

Module 2

Installation automatisée avec Kickstart

TP : servir un fichier kickstart et installer N machines

Objectif et prérequis

Faire de `vm1-ks` le **serveur de son propre fichier kickstart**, l'améliorer, puis installer plusieurs machines identiques **sans répondre à aucune question**.

Ce qui est déjà en place

La VM `vm1-ks` du TP « Installation d'AlmaLinux 10 », avec son `/root/anaconda-ks.cfg`, et l'ISO `AlmaLinux-10-boot.iso` sur le poste de travail (AlmaLinux 9, KVM).

Les trois améliorations

1. le paquet `bash-completion` installé d'office,
2. un traitement `%post` : `sudo` sans mot de passe pour les administrateurs (`NOPASSWD:`),
3. un traitement `%post` : l'adresse IP affichée dans `/etc/issue`, au-dessus de l'invite de connexion (séquence `\4`).

> ⚠ Les deux machines ne jouent pas le même rôle
>
> Les commandes `virt-install` et `virsh` se tapent sur le **poste de travail**. Tout le reste se tape dans `vm1-ks`, sauf mention contraire. L'adresse de `vm1-ks`, relevée au TP précédent, est notée `IP1` dans les commandes.

Étape 1 : Préparer `/srv/kickstart`

Dans `vm1-ks` :

```
# sudo mkdir -p /srv/kickstart
sudo cp /root/anaconda-ks.cfg /srv/kickstart/ks.cfg
sudo chown -R admin: /srv/kickstart
chmod 644 /srv/kickstart/ks.cfg
```

`anaconda-ks.cfg` n'est lisible que par `root` : la copie reçoit les droits qui permettent au serveur HTTP de la lire.

> ⛔ Le mot de passe chiffré est publié
>
> Le fichier contient le **hachage** du mot de passe de `admin`. Quiconque atteint le serveur HTTP peut le récupérer et chercher à le casser. Le TP se déroule sur un réseau fermé, un déploiement réel utiliserait un hachage dédié aux installations et un réseau restreint.

Étape 2 : Le servir en HTTP

Dans `vm1-ks`, deux commandes :

```
# sudo dnf install -y python3
sudo firewall-cmd --add-port=8080/tcp
```

Lancer le serveur dans un second terminal de la VM, il tourne au premier plan :

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

Depuis le **poste de travail**, contrôler que le fichier est servi :

```
$ curl -sI http://IP1:8080/ks.cfg | head -1
```

`HTTP/1.0 200 OK` confirme que le fichier est accessible. Une réponse refusée ou muette désigne le pare-feu de `vm1-ks`.

> **La règle `firewall-cmd` est temporaire**
>
> Sans `--permanent`, le port s'ouvre jusqu'au prochain rechargement ou redémarrage du pare-feu. Cela suffit pour ce TP.

Étape 3 : Adapter la source des paquets

Ouvrir `/srv/kickstart/ks.cfg` et regarder la ligne de **source d'installation** :

- une ligne `url --url=...` ou `url --mirrorlist=...` : elle pointe vers le réseau, **la garder**,
- une ligne `cdrom` : elle suppose un média dans la machine installée, la remplacer par les deux dépôts réseau ci-dessous.

```
url --url="https://repo.almalinux.org/almalinux/10/BaseOS/x86_64/os/"
repo --name="AppStream" --
baseurl="https://repo.almalinux.org/almalinux/10/AppStream/x86_64/os/"
```

Contrôler aussi que le disque est bien `vda` (`ignoredisk`, `part`) et que l'installation est en mode `text`, ce qui évite d'ouvrir une console graphique par machine. Une installation faite en mode graphique a écrit `graphical` : le remplacer par `text`.

Étape 4 : Tester le fichier tel quel

Créer une VM `vm2-ks` sur le poste de travail. L'ISO ne sert qu'à lancer l'installateur, les paquets viennent du réseau.

```
# sudo virt-install --name vm2-ks --memory 2048 --vcpus 2 --cpu \
  host-passthrough \
  --disk size=10,bus=virtio --network network=default \
  --cdrom /var/lib/libvirt/images/AlmaLinux-10-boot.iso --os-variant \
  almalinux9
```

Dans la console, au menu de démarrage :

1. sélectionner **Install AlmaLinux 10**,
2. appuyer sur **e** (BIOS) ou **Tab** (UEFI) : la ligne de commande du noyau s'affiche,
3. ajouter à la fin `inst.ks=http://IP1:8080/ks.cfg`, puis **Entrée**.

L'installateur ne pose plus de question, il lit le fichier, partitionne, installe et redémarre.

> **Machine en UEFI**
>
> Sur un menu de type GRUB (UEFI), la touche **e** remplace **Tab** : ajouter `inst.ks=...` à la ligne qui commence par `linuxefi` ou `linux`, puis **Ctrl+x**.

Dans le terminal qui sert HTTP, les requêtes de `vm2-ks` apparaissent : c'est la preuve que le fichier a été lu.

Étape 5 : Améliorer le fichier

Éditer `/srv/kickstart/ks.cfg` dans `vm1-ks`. Dans la section `%packages`, ajouter une ligne :

```
%packages
@^minimal-environment
bash-completion
%end
```

Tout à la fin du fichier, après les autres sections, ajouter :

```
%post --log=/root/ks-post.log
echo "%wheel ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/10-wheel-nopasswd
chmod 0440 /etc/sudoers.d/10-wheel-nopasswd
visudo -cf /etc/sudoers.d/10-wheel-nopasswd
echo 'Adresse IP : \4' >> /etc/issue
%end
```

Étape 6 : Ce que fait le `%post`

Ce que fait chaque ligne

Le fichier de `/etc/sudoers.d/` donne à tout le groupe `wheel` le droit de lancer des commandes sans mot de passe. Il garde les droits `0440` attendus par `sudo`, et `visudo -cf` en contrôle la syntaxe : son verdict (`parsed OK`) se lit dans `/root/ks-post.log`. Les guillemets simples de la dernière ligne gardent `\4` tel quel dans le fichier : c'est `agetty`, au moment d'afficher l'invite, qui le remplace par l'adresse IP de l'interface.

`NOPASSWD` ne convient pas partout

Le droit d'élévation sans mot de passe sert ici pour automatiser et pour le TP. Sur un serveur de production, il augmente les conséquences d'un compte `admin` compromis.

Étape 7 : Installer N machines

Sur le poste de travail, `virt-install` ajoute lui-même `inst.ks` à la ligne de commande du noyau : plus rien à taper dans la console. Trois machines à la suite :

```
# for i in 3 4 5; do
    sudo virt-install --name vm$i-ks --memory 2048 --vcpus 2 --cpu \
    host-passthrough \
      --disk size=10,bus=virtio --network network=default \
      --location /var/lib/libvirt/images/AlmaLinux-10-boot.iso \
    --os-variant almalinux9 \
      --extra-args "inst.ks=http://IP1:8080/ks.cfg" --noautoconsole
done
```

Suivre l'avancement, l'installation prend quelques minutes :

```
$ sudo virsh list --all
```

Chaque installation s'arrête ou redémarre à la fin du fichier. Une machine restée éteinte se relance par `sudo virsh start vm3-ks`.

`--location` ou `--cdrom`

Avec `--location`, `virt-install` extrait lui-même le noyau et l'initrd de l'ISO pour y mettre ses arguments. Avec `--cdrom`, comme à l'étape 4, il faut taper `inst.ks` dans le menu de démarrage.

Étape 8 : Vérifier les machines

Relever l'adresse de chaque machine, puis tester depuis le poste de travail :

```
# sudo virsh domifaddr vm3-ks
```

```
$ ssh admin@IP3 'sudo whoami; rpm -q bash-completion; hostname'
```

Constater :

- `sudo` n'a pas demandé de mot de passe, et répond `root`,
- `bash-completion` est installé, alors qu'il manquait sur `vm1-ks` si le TP précédent l'a constaté,
- l'adresse IP s'affiche dans la console de la VM, au-dessus de l'invite de connexion `login:`,
- `/etc/issue` contient `\4` tel quel : `cat /etc/issue`.

Questions :

1. Quel est le nom d'hôte de chaque machine installée ? Est-ce un souci ?
2. Où trouver la trace de l'exécution du `%post` sur la machine installée ?

> **Réponses**
>
> Les trois machines portent le nom écrit dans la ligne `network --hostname=` du fichier. Le journal du `%post` est `/root/ks-post.log`, demandé par `--log=`.

Étape 9 : Un fichier par machine (facultatif)

Pour donner son propre nom à chaque machine, générer une copie du fichier par machine, dans `vm1-ks` :

```
$ for i in 6 7; do
    sed "s/--hostname=.*/--hostname=srv-ks$i.exemple.lan/" \
    /srv/kickstart/ks.cfg > /srv/kickstart/ks-$i.cfg
  done
```

Puis installer `vm6-ks` et `vm7-ks` comme à l'étape 7, avec `inst.ks=http://IP1:8080/ks-$i.cfg`. Contrôler le nom d'hôte de chacune avec `hostname`.

Étape 10 : Un conteneur `httpd` à la place de Python (facultatif)

Dans `vm1-ks`, arrêter le serveur Python (**Ctrl+c**), puis :

```
$ sudo dnf install -y podman
  podman run -d --name ks-httpd -p 8080:80 -v \
    /srv/kickstart:/usr/local/apache2/htdocs:ro,Z \
    docker.io/library/httpd:2.4
  curl -sI http://localhost:8080/ks.cfg | head -1
```

Le même `inst.ks=http://IP1:8080/ks.cfg` fonctionne. L'option `:Z` donne aux fichiers le contexte SELinux que le conteneur peut lire.

> **À savoir**
>
> `/usr/local/apache2/htdocs` est le répertoire servi par l'image officielle `httpd`. Un conteneur lancé par `admin` n'a pas besoin de `sudo`, et le port 8080 dépasse la limite des ports privilégiés.

Étape 11 : Un ISO qui démarre sur le kickstart (facultatif)

Le fichier reste sur `vm1-ks`, servi en HTTP. Sur le poste de travail, `mkksiso` ajoute `inst.ks=` aux lignes de

démarrage d'une copie de l'ISO :

```
# dnf install -y lorax
D=/var/lib/libvirt/images
mkksiso --cmdline "inst.ks=http://IP1:8080/ks.cfg" \
  $D/AlmaLinux-10-boot.iso $D/AlmaLinux-10-ks.iso
```

Installer `vm8-ks` à partir de ce nouvel ISO :

```
# sudo virt-install --name vm8-ks --memory 2048 --vcpus 2 --cpu \
  host-passthrough \
  --disk size=10,bus=virtio --network network=default \
  --cdrom /var/lib/libvirt/images/AlmaLinux-10-ks.iso --os-variant \
  almalinux9
```

Au menu de démarrage, choisir **Install AlmaLinux 10** : la ligne de commande contient déjà `inst.ks=`, rien n'est à compléter.

> **Le fichier est lu à l'installation**
>
> L'ISO ne contient que l'adresse : modifier `ks.cfg` sur `vm1-ks` suffit, sans refaire l'ISO. Le serveur HTTP de `vm1-ks` doit répondre pendant l'installation.

> **Toutes les entrées du menu**
>
> `mkksiso` ajoute l'argument à **toutes** les entrées de démarrage de l'ISO (BIOS et UEFI), il ne crée pas d'entrée séparée « Installation KS ». Une entrée supplémentaire demande de modifier à la main `isolinux.cfg` et `grub.cfg` dans l'ISO.

Étape 12 : Nettoyer

Sur le poste de travail, supprimer les VM de test et leurs disques, en gardant `vm1-ks` et `vm3-ks` (source du TP « Booter autrement ») :

```
# for v in vm2-ks vm4-ks vm5-ks vm6-ks vm7-ks vm8-ks; do
  sudo virsh destroy $v 2>/dev/null
  sudo virsh undefine $v --remove-all-storage 2>/dev/null
done
```

Supprimer aussi `/var/lib/libvirt/images/AlmaLinux-10-ks.iso`. Dans `vm1-ks`, arrêter le service de fichiers : **Ctrl+c** pour Python, `podman rm -f ks-httpd` pour le conteneur.

Module 3

Mettre à jour le système

TP : rpm, dépôts, recherche, historique et mises à jour avec dnf

Objectif et machine

Interroger la base **RPM**, ouvrir un paquet **sans l'installer**, lire la configuration de **DNF**, puis chercher, installer, annuler et **mettre à jour** avec `dnf`.

La machine

Une VM AlmaLinux 10 avec accès à Internet et un compte qui dispose de `sudo`. Tout se fait en SSH.

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

> ⚠ **Premier appel de `dnf`**
>
> Le premier appel télécharge les méta-données des dépôts : plusieurs minutes selon la connexion. Les suivants utilisent le cache.

Étape 1 : Interroger la base RPM

```
$ rpm -q bash
rpm -qi bash | head -8
rpm -ql bash | head -3
rpm -qc bash
```

Questions :

1. Comment lister seulement les fichiers de configuration d'un paquet ?

> ℹ **Réponses**
>
> Avec `rpm -qc` (voir slide « Les requêtes : `rpm -q` »).

```
$ rpm -qf /etc/passwd
```

Questions :

2. Quel paquet a installé `/etc/passwd` ?

> ℹ **Réponses**
>
> `setup`.

```
$ rpm -q --requires bash | head -3
rpm -q --scripts systemd | head -4
```

Questions :

3. Que contient la section `preinstall scriptlet` de `systemd` ?

> **Réponses**
>
> Un script exécuté avant l'installation, qui crée l'utilisateur et le groupe `systemd-oom` s'ils n'existent pas.

Étape 2 : Ouvrir un paquet sans l'installer

Télécharger le paquet `tree` sans l'installer, puis l'examiner :

```
$ mkdir ~/tp-rpm && cd ~/tp-rpm
dnf download tree
rpm -qpi tree-*.rpm | head -3
rpm -qpl tree-*.rpm
rpm --checksig tree-*.rpm
```

Extraire son contenu avec `rpm2cpio` :

```
$ rpm2cpio tree-*.rpm | cpio -idm
find . -type f
```

Questions :

1. Que signifie `digests signatures OK` ?
2. Où `tree` serait-il installé, et où est-il extrait ?

> **Réponses**
>
> Les empreintes et la signature GPG du paquet sont valides (voir slide « Le format RPM »). Il serait dans `/usr/bin/tree`. `rpm2cpio` l'extrait sous `./usr/bin/tree`, dans le répertoire courant, sans rien installer (voir slide « La commande `rpm` »).

Étape 3 : Configuration et dépôts

```
$ ls -l /etc/yum.conf
cat /etc/dnf/dnf.conf
```

Questions :

1. Vers quoi pointe `/etc/yum.conf` ?

> **Réponses**
>
> Vers `dnf/dnf.conf` (voir slide « Fichiers de configuration »).

```
$ ls /etc/yum.repos.d
dnf repolist
dnf repolist all | head -12
```

Questions :

2. Quels dépôts sont actifs ? Le dépôt `crb` l'est-il ?

> **Réponses**
>
> `appstream`, `baseos` et `extras` sont actifs. `crb`, `*-debuginfo` et `*-source` sont désactivés.

```
$ dnf repoinfo baseos | head -14
```

Questions :

3. Quelle durée de vie des méta-données le dépôt `baseos` annonce-t-il (`Repo-expire`) ?

> **Réponses**
>
> `Repo-expire` vaut 86400 secondes, soit 24 h : la valeur de `metadata_expire` du fichier `.repo` (voir slide « Paramètres de la section `[main]` »).

Étape 4 : Chercher

```
$ dnf provides '*/lspci'
```

Questions :

1. Quel paquet fournit `lspci` ?
2. Que signifie `Repo : @System` dans la réponse de `dnf provides` ?

> **Réponses**
>
> `pciutils`. `@System` indique que le paquet est déjà **installé** : la ligne vient de la base RPM locale, pas d'un dépôt (voir slide « Chercher des paquets »).

```
$ dnf info tree | head -8
dnf search tree | head -5
dnf list --showduplicates tree
dnf repoquery --requires tree
```

Questions :

3. Dans quel dépôt se trouve `tree` ?

> **Réponses**
>
> `tree` est dans `baseos`, colonne de droite de `dnf list`.

Étape 5 : Installer, annuler

Installer le RPM téléchargé à l'étape 2, qui passe par `dnf` pour les dépendances :

```
$ sudo dnf -y install ./tree-*.rpm
tree --version
sudo dnf history | head -4
sudo dnf history info last | head -14
```

Annuler la transaction, puis installer depuis le dépôt :

```
$ sudo dnf -y history undo last
rpm -q tree
sudo dnf -y install tree
sudo dnf -y remove tree
```

Questions :

1. Quelle ligne de `dnf history info` indique d'où vient le paquet installé ?
2. Que fait `history undo last` d'une transaction d'installation ?

> **Réponses**
>
> `Install tree-... @@commandline` : le paquet vient d'un fichier local et non d'un dépôt. L'`undo` d'une installation désinstalle le paquet : `rpm -q tree` répond « package tree is not installed » (voir slide « Installer, supprimer, annuler »).

Étape 6 : Ajouter un dépôt : installer `micro`

`micro` est un éditeur de texte en terminal. Il ne figure dans aucun dépôt d'AlmaLinux, mais dans **EPEL** (*Extra Packages for Enterprise Linux*), un dépôt communautaire du projet Fedora :

```
$ dnf info micro
```

Questions :

1. Que répond `dnf info micro` ?

> **Réponses**
>
> `Error: No matching Packages to list` : aucun des dépôts actifs ne fournit `micro`.

Le paquet `epel-release` (dans `extras`) dépose le fichier `.repo` d'EPEL et sa clé de signature :

```
# sudo dnf install epel-release
rpm -ql epel-release
dnf repolist
```

Questions :

2. Quels fichiers `epel-release` installe-t-il pour déclarer le dépôt et vérifier ses paquets ?
3. Quel dépôt est apparu dans `dnf repolist` ?

> **Réponses**
>
> `/etc/yum.repos.d/epel.repo` (et `epel-testing.repo`, désactivé) pour la déclaration, `/etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-10` pour la clé. Le dépôt `epel` est maintenant actif (voir slide « Lister et explorer les dépôts »).

```
$ dnf info micro | head -8
```

```
# sudo dnf install micro
```

La clé d'EPEL est proposée à l'import au premier paquet installé depuis ce dépôt.

Questions :

4. Dans quel dépôt `dnf info` trouve-t-il `micro` ?
5. Que demande `dnf` avant d'installer, et pourquoi ?

> **Réponses**
>
> Dans `epel`. Il demande de confirmer l'import de la clé GPG d'EPEL (empreinte affichée) : sans elle, `dnf` ne peut pas vérifier la signature des paquets du dépôt.

```
$ micro -version
rpm -qf /usr/bin/micro
dnf repoquery --installed --qf '%{name} %{from_repo}' micro
```

Ouvrir un fichier avec `micro` (Ctrl-S enregistre, Ctrl-Q quitte) :

```
$ micro ~/essai.txt
```

Questions :

6. Quelle version de `micro` est installée, et de quel dépôt provient-elle ?

> **Réponses**
>
> La version se lit avec `micro -version` (2.0.15 au moment de la rédaction). `dnf repoquery` indique `epel` comme dépôt d'origine.

> **Dépôts tiers**
>
> EPEL n'est pas maintenu par AlmaLinux et ne bénéficie pas de son support. Ses paquets suivent le rythme du projet Fedora, avec leurs mises à jour. Chaque dépôt ajouté s'ajoute à ce qu'il faut surveiller en production.

Retirer ensuite `micro`, puis désactiver le dépôt sans le supprimer :

```
# sudo dnf remove micro
sudo dnf config-manager --set-disabled epel
dnf repolist
```

Questions :

7. Quelle commande réactive EPEL pour une seule commande, sans modifier le fichier `.repo` ?

> **Réponses**
>
> `sudo dnf --enablerepo=epel install paquet`.

Étape 7 : Repérer les mises à jour

```
$ dnf check-update | tail -3 ; echo "code retour : ${PIPESTATUS[0]}"
dnf updateinfo list | head -5
dnf list updates | head -4
sudo dnf upgrade --security --assumeno | head -12
```

`--assumeno` répond « non » à la confirmation : rien n'est installé.

Questions :

1. Quel code retour `check-update` donne-t-il quand des mises à jour sont disponibles ?
2. Que signifie `ALSA-2026:...` dans `updateinfo list` ?
3. Quelle est la différence entre `dnf upgrade` et `dnf upgrade --security` ?

> **Réponses**
>
> `100` (voir slide « Mettre à jour le système »). Un avis (*advisory*) publié par AlmaLinux : sa gravité et son type (`Sec.`, `Low`, `Moderate`...) s'affichent à côté. `--security` ne retient que les paquets liés à un avis de sécurité.

> **À savoir**
>
> `ALSA` signifie *AlmaLinux Security Advisory*. Une mise à jour du noyau s'**installe** à côté du noyau en cours (« Installing: kernel ») : le nouveau noyau ne tourne qu'après un redémarrage.

Étape 8 : Modules et groupes

```
$ dnf module list
```

Questions :

1. Que renvoie `dnf module list` sur AlmaLinux 10 ?

Réponses

Rien : aucun contenu modulaire n'est distribué sur la version 10 (voir slide « Modules et streams »).

```
$ dnf grouplist | head -8
dnf groupinfo "Development Tools" | head -12
```

Questions :

2. Quels paquets `groupinfo` classe-t-il en « Mandatory » ?

Réponses

Les paquets obligatoires d'un groupe (`gcc`, `binutils`, `gdb`…) sont toujours installés avec lui, les optionnels jamais par défaut (voir slide « Traiter les paquets par groupes »).

Étape 9 : Le cache

```
$ du -sh /var/cache/dnf
sudo dnf clean expire-cache
```

Questions :

1. Que fait `dnf clean expire-cache` : supprime-t-il les méta-données ?

Réponses

Non, il les marque comme expirées : elles sont revérifiées au prochain appel (voir slide « Gérer le cache »).

```
$ dnf repolist | head -2
grep -h metadata_expire /etc/yum.repos.d/almalinux-baseos.repo
```

Questions :

2. Pourquoi éviter `dnf clean all` en réflexe ?

Réponses

`clean all` supprime tout le cache : le prochain appel retélécharge toutes les méta-données (plusieurs dizaines de Mo, plusieurs minutes sur la VM de test).

Nettoyer

`rm -r ~/tp-rpm` supprime le dossier de travail de l'étape 2.

Module 4

Maîtriser la compilation des logiciels

TP : compiler GNU Hello, écrire un Makefile, publier une bibliothèque partagée

Objectif et machine

Compiler un logiciel **depuis son archive de sources**, l'installer dans `/opt`, écrire un **Makefile** à la main, puis **publier une bibliothèque partagée** pour que le chargeur la trouve.

La machine

La VM `vm3-ks` (ou toute VM AlmaLinux 10 reliée à Internet), avec un compte qui dispose de `sudo`. Tout se fait en SSH.

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

> **Le logiciel**
>
> **GNU Hello** : un programme de quelques lignes qui affiche un message, mais dont l'archive suit toute la démarche GNU (`configure`, `make`, tests, pages de manuel, traductions). Il n'a aucune dépendance au-delà du compilateur.

Étape 1 : Préparer la machine

Le compilateur et `make` viennent du groupe `Development Tools`. Le paquet `man-db` sert à l'étape 4.

```
$ sudo dnf group install "Development Tools"
sudo dnf install man-db
```

Vérifier :

```
$ gcc --version | head -1
make --version | head -1
```

Questions :

1. Quelle commande installe d'un coup `gcc`, `make`, `autoconf` et `patch` ?
2. Que signale un message `configure: error: ... not found` et comment retrouver le paquet à installer ?

> **Réponses**
>
> Le groupe `Development Tools`. Le message désigne en général un paquet `-devel` manquant : `dnf provides '*/fichier.h'` retrouve lequel (voir slide « Préparer la machine de compilation »).

Étape 2 : Récupérer et vérifier l'archive

Travailler dans un répertoire de sources, avec un **utilisateur ordinaire** :

```
$ mkdir -p ~/src && cd ~/src
curl -O https://mirrors.kernel.org/gnu/hello/hello-2.12.3.tar.gz
```

Contrôler la somme de l'archive. Elle a été relevée sur le miroir le 2026-10-06. Créer le fichier `hello.sha256` avec un éditeur, contenant la somme, deux espaces, puis le nom de l'archive :

```
0d5f60154382fee10b114a1c34e785d8b1f492073ae2d3a6f7b147687b366aa0  hello-2.12.3.tar.gz
```

Puis :

```
$ sha256sum -c hello.sha256
```

Étape 3 : Extraire et lire

Lister le contenu **avant** de l'extraire, puis extraire et lire la documentation d'accueil :

```
$ tar tf hello-2.12.3.tar.gz | head
tar xf hello-2.12.3.tar.gz
cd hello-2.12.3
less README INSTALL
```

Questions :

1. Que répond `sha256sum -c` si l'archive est intacte ?
2. Pourquoi lister le contenu de l'archive avant de l'extraire ?

> **Réponses**
>
> `hello-2.12.3.tar.gz: OK`. Pour vérifier que tout est dans un seul répertoire (voir slide « Démarche : du tarball au binaire »).

> **À savoir**
>
> Le projet GNU publie aussi une signature (`.sig`) de chaque archive, vérifiable avec `gpg`. C'est un contrôle plus fort qu'une somme relevée sur le même serveur que l'archive.

Étape 4 : Configurer, compiler, tester

Lister les options de `configure` qui concernent l'emplacement de l'installation :

```
$ ./configure --help | grep -E -- '^ +--(prefix|sysconfdir|mandir)='
```

Configurer pour une installation dans `/opt/hello`, puis compiler **en parallèle** :

```
$ ./configure --prefix=/opt/hello
```

Questions :

1. Quelle option de `configure` fixe le répertoire d'installation ?

> **Réponses**
>
> Le répertoire est fixé par `--prefix`.

```
$ make -j $(nproc)
```

Questions :

2. Quel utilisateur a lancé `configure`, `make` et `make check` ?

> **Réponses**
>
> Un utilisateur ordinaire : seule l'installation demande `root`.

```
$ make check
```

Questions :

3. Combien de tests ont réussi, combien ont échoué ?

> **Réponses**
>
> Le bilan se lit dans les lignes `# PASS` et `# FAIL` de la sortie de `make check`.

Étape 5 : Installer et finaliser

Seule cette étape demande `root` :

```
$ sudo make install
```

Observer ce qui a été installé et par qui :

```
$ find /opt/hello -maxdepth 3 | sort
ls -l /opt/hello/bin
/opt/hello/bin/hello
```

Étape 6 : Finaliser

`hello` n'est pas encore dans le `PATH`, et `man hello` ne trouve rien. **Finaliser à la main**, comme le dit la slide « Compilation, tests, installation » :

Créer `/etc/profile.d/hello.sh` avec un éditeur lancé en root (`sudo`) :

```
≡ export PATH=/opt/hello/bin:$PATH
```

Ajouter une ligne à la fin de `/etc/man_db.conf` avec un éditeur lancé en root (`sudo`) :

```
≡ MANDATORY_MANPATH /opt/hello/share/man
```

Puis :

```
$ source /etc/profile.d/hello.sh
command -v hello
man hello | head -5
```

Questions :

1. Quels sont les propriétaires des fichiers de `/opt/hello` ?
2. Quels fichiers modifie-t-on pour que le `PATH` et `man` connaissent le logiciel ?
3. Que faudrait-il faire en plus si `hello` installait des bibliothèques partagées ?

> **Réponses**
>
> `root`, puisque `make install` s'est exécuté avec `sudo`. `/etc/profile.d/` pour le `PATH`, `/etc/man_db.conf` pour les pages de manuel. Déclarer le répertoire dans `/etc/ld.so.conf.d/` puis relancer `ldconfig` (étape 10).

> **À conserver**
>
> Le répertoire `~/src/hello-2.12.3` et l'installation dans `/opt/hello` sont à garder sur la VM.

Étape 7 : Un Makefile à la main

Dans un nouveau répertoire, trois fichiers : un programme, une fonction, son en-tête.

```
$ mkdir -p ~/src/salut && cd ~/src/salut
```

```
/* salut.h */
void salut(const char *nom);
```

```
/* salut.c */
#include <stdio.h>
#include "salut.h"

void salut(const char *nom)
{
    printf("Bonjour, %s !\n", nom);
}
```

```
/* main.c */
#include "salut.h"

int main(void)
{
    salut("stagiaire");
    return 0;
}
```

Étape 8 : Le Makefile

À taper avec un éditeur. Les lignes de commande commencent par **une tabulation**, pas par des espaces :

```
CC     = gcc
CFLAGS = -O2 -Wall
EXEC   = monprog
OBJS   = main.o salut.o

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

main.o salut.o: salut.h

clean:
        @rm -f $(OBJS) $(EXEC)
```

> **À savoir**
>
> Un copier-coller depuis un PDF remplace les tabulations par des espaces. `make` répond alors `*** missing separator. Arrêt.` : la ligne désignée doit recommencer par une tabulation.

Étape 9 : Observer les reconstructions

```
$ make
./monprog
make
```

Questions :

1. Que répond le deuxième `make`, lancé sans rien avoir modifié ?
2. Quelle est la première cible, et que lance `make` sans argument ?

> **Réponses**
>
> `make: Nothing to be done for 'all'.` (ou `Rien à faire pour « all »` si la VM est en français). La première cible est `all`, celle que `make` exécute sans argument (voir slide « La commande `make` »).

```
$ touch salut.c
make
```

Questions :

3. Quelle(s) commande(s) `gcc` s'exécutent après `touch salut.c` ?

> **Réponses**
>
> Après `touch salut.c`, seul `salut.c` est recompilé, puis l'édition de liens refait `monprog`.

```
$ touch salut.h
make
make clean
```

Questions :

4. Et après `touch salut.h` ? Quelle ligne du Makefile l'explique ?

> **Réponses**
>
> Après `touch salut.h`, les deux `.c` sont recompilés, grâce à la ligne `main.o salut.o: salut.h`.

Étape 10 : Une bibliothèque hors des répertoires standard

Construire `salut.c` en **bibliothèque partagée**, l'installer dans `/opt/salut/lib`, puis lier le programme dessus :

```
$ make clean
gcc -fPIC -shared -o libsalut.so salut.c
sudo install -D libsalut.so /opt/salut/lib/libsalut.so
gcc -O2 -Wall -c main.c -o main.o
gcc -o monprog-dyn main.o -L/opt/salut/lib -lsalut
./monprog-dyn
```

Le programme ne démarre pas. Chercher pourquoi :

```
$ objdump -p monprog-dyn | grep NEEDED
ldd ./monprog-dyn
```

Questions :

1. Quel message d'erreur le chargeur affiche-t-il ?
2. Quelle bibliothèque `ldd` marque-t-il `not found` ?
3. Pourquoi `-L/opt/salut/lib` a-t-il suffi à la compilation et pas à l'exécution ?

> **Réponses**
>
> `error while loading shared libraries: libsalut.so: cannot open shared object file`, et `libsalut.so => not found`. `-L` ne sert qu'à l'édition de liens. À l'exécution, le chargeur applique l'ordre de la slide « Ordre de recherche des bibliothèques » et `/opt/salut/lib` n'y figure pas.

Étape 11 : Rendre la bibliothèque visible

Essai **temporaire**, pour ce seul lancement :

```
$ LD_LIBRARY_PATH=/opt/salut/lib ./monprog-dyn
```

Solution **durable** : déclarer le répertoire, puis reconstruire le cache.
Créer `/etc/ld.so.conf.d/salut.conf` avec un éditeur lancé en root (`sudo`), contenant le répertoire :

```
≡ /opt/salut/lib
```

Puis reconstruire le cache :

```
$ sudo ldconfig
```

```
$ ldconfig -p | grep salut
ldd ./monprog-dyn
./monprog-dyn
```

Questions :

1. Quelle commande reconstruit le cache lu par le chargeur, et dans quel fichier est-il ?
2. Que se passe-t-il si on oublie de la relancer après avoir créé `salut.conf` ?

> **Réponses**
>
> `ldconfig` reconstruit `/etc/ld.so.cache`. Sans elle, le chargeur ne connaît pas encore le nouveau répertoire et l'erreur de l'étape précédente subsiste (voir slide « `ldconfig` et `/etc/ld.so.conf` »).

Étape 12 : `RPATH` et `RUNPATH` *(facultative)*

Intégrer le chemin au binaire, sans toucher à la configuration du système :

```
$ gcc -o monprog-rp main.o -L/opt/salut/lib -lsalut \
    -Wl,-rpath,/opt/salut/lib
readelf -d monprog-rp | grep -E 'RPATH|RUNPATH'
gcc -o monprog-rp2 main.o -L/opt/salut/lib -lsalut \
    -Wl,--enable-new-dtags,-rpath,/opt/salut/lib
readelf -d monprog-rp2 | grep -E 'RPATH|RUNPATH'
```

Questions :

1. Quel attribut ELF `readelf` montre-t-il pour chacun des deux binaires ?
2. Lequel est consulté avant `LD_LIBRARY_PATH` ?

> **Réponses**
>
> `RPATH` pour le premier : l'éditeur de liens de RHEL et dérivées l'écrit par défaut, et `--enable-new-dtags` donne `RUNPATH` pour le second. `RPATH` est consulté en premier, `RUNPATH` après `LD_LIBRARY_PATH` et avant le cache (voir slide « Ordre de recherche des bibliothèques »).

Étape 13 : `LD_PRELOAD` *(facultative)*

Détourner une fonction de la bibliothèque :

```c
/* hook.c */
#include <stdio.h>

void salut(const char *nom)
{
    printf("Salut détourné pour %s\n", nom);
}
```

```
$ gcc -fPIC -shared -o libhook.so hook.c
LD_PRELOAD=./libhook.so ./monprog-dyn
./monprog-dyn
```

Questions :

1. Quelle fonction s'exécute avec `LD_PRELOAD`, puis sans la variable ?

> **Réponse**
>
> Celle de `libhook.so`, chargée en premier. Sans la variable, celle de `libsalut.so`.

> **À savoir**
>
> Un binaire où `salut.o` est lié directement, comme `monprog` de l'étape 7 (à reconstruire avec `make`), n'est pas affecté : `LD_PRELOAD=./libhook.so ./monprog` affiche le message d'origine.

Module 5

Maîtriser la production de packages RPM

TP : empaqueter GNU Hello, servir et signer un dépôt

Objectif et machine

Transformer l'archive de **GNU Hello** en **paquet RPM**, publier ce paquet dans un **dépôt** que `dnf` consulte en HTTP, puis **signer** paquets et métadonnées pour que le client refuse tout ce qui ne l'est pas.

La machine

La VM `vm3-ks` (ou toute VM AlmaLinux 10 reliée à Internet), avec un compte qui dispose de `sudo`. Tout se fait en SSH, en utilisateur ordinaire.

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

> **Le logiciel**
>
> Ce TP reprend l'archive `hello-2.12.3.tar.gz` du TP de compilation, rangée dans `~/src`. Le compilateur et `make` (groupe `Development Tools`) doivent être installés.

Sans l'archive, la récupérer :

```
$ mkdir -p ~/src
curl -o ~/src/hello-2.12.3.tar.gz \
  https://mirrors.kernel.org/gnu/hello/hello-2.12.3.tar.gz
```

Étape 1 : Préparer l'environnement

```
$ sudo dnf install rpm-build rpmdevtools createrepo_c rpm-sign gnupg2
rpmdev-setuptree
ls ~/rpmbuild
```

Questions :

1. Quels sont les six répertoires créés par `rpmdev-setuptree` ?
2. Pourquoi construit-on un paquet sans être `root` ?

> **Réponses**
>
> `BUILD`, `BUILDROOT`, `RPMS`, `SOURCES`, `SPECS`, `SRPMS`. Parce que la phase `%install` n'écrit que dans `BUILDROOT/` et ne risque pas de déborder sur le système réel (voir slides « La commande `rpmbuild` » et « Arborescence `rpmbuild` et macros personnelles »).

Étape 2 : Le fichier SPEC

Ranger l'archive dans `SOURCES/` :

```
$ cp ~/src/hello-2.12.3.tar.gz ~/rpmbuild/SOURCES/
```

Puis écrire `~/rpmbuild/SPECS/hello.spec` avec un éditeur :

```
Name:           hello
Version:        2.12.3
Release:        1%{?dist}
Summary:        Programme d'exemple de la distribution GNU
License:        GPL-3.0-or-later
URL:            https://www.gnu.org/software/hello/
Source0:        https://mirrors.kernel.org/gnu/hello/%{name}-%{version}.tar.gz
BuildRequires:  gcc, make

%description
GNU Hello affiche un message d'accueil. Il sert d'exemple
de logiciel construit avec les outils GNU.
```

Étape 3 : Le fichier SPEC (suite)

À la suite du fichier :

```
%prep
%autosetup

%build
%configure
%make_build

%install
%make_install
rm -f %{buildroot}%{_infodir}/dir
%find_lang %{name}

%check
make check

%files -f %{name}.lang
%license COPYING
%doc README NEWS
%{_bindir}/hello
%{_mandir}/man1/hello.1*
%{_infodir}/hello.info*

%changelog
* Tue Oct 06 2026 Prénom Nom <adresse@exemple.org> - 2.12.3-1
- Première version packagée
```

Questions :

1. Que remplacent `%configure`, `%make_build` et `%make_install` ?
2. Quelle commande donne la valeur de `%{_bindir}` ?
3. Quelle section décrit la liste des fichiers du paquet ?

> **Réponses**
>
> `./configure` avec les chemins standards, `make` avec les options du système, et `make install DESTDIR=%{buildroot}`. La valeur se lit avec `rpm --eval '%{_bindir}'` (`/usr/bin`). La section `%files` (voir slide « Les macros du SPEC »).

> **À savoir**
>
> Trois lignes sortent du modèle de base. `rm -f ...dir` retire le fichier d'index `info` partagé, qui appartient à un autre paquet. `%find_lang` réunit les traductions du programme dans une liste, que `%files -f` ajoute. `%check` lance les tests du logiciel pendant la construction.

Étape 4 : Construire par étapes

Extraire seulement, regarder l'arbre, puis construire le RPM binaire et le RPM source :

```
$ cd ~/rpmbuild/SPECS
rpmbuild -bp hello.spec
ls ~/rpmbuild/BUILD
```

Questions :

1. Quelle étape produit l'arbre qui apparaît dans `BUILD/` ?

> **Réponses**
>
> `-bp` exécute `%prep` : extraction et patches.

```
$ rpmbuild -ba hello.spec
ls ~/rpmbuild/RPMS/*/ ~/rpmbuild/SRPMS
```

Questions :

2. Quels fichiers `rpmbuild -ba` a-t-il produits, et dans quels répertoires ?
3. Que se passe-t-il si une dépendance de construction manque ?

> **Réponses**
>
> `-ba` produit les RPM binaires dans `RPMS/x86_64/` et le RPM source `hello-2.12.3-1.el10.src.rpm` dans `SRPMS/`. Un `BuildRequires` absent arrête la construction (`Failed build dependencies`) : installer le paquet cité, puis relancer.

> **À savoir**
>
> `hello-debuginfo` et `hello-debugsource` (symboles de débogage) sont produits automatiquement. Le dépôt n'en a pas besoin.

Étape 5 : Inspecter le paquet

```
$ cd ~/rpmbuild/RPMS/x86_64
rpm -qpl hello-2.12.3-1.el10.x86_64.rpm | grep -v locale
```

Questions :

1. Où le paquet installe-t-il le binaire et la page de manuel ?

> **Réponses**
>
> `/usr/bin/hello` et `/usr/share/man/man1/hello.1.gz`.

```
$ rpm -qpi hello-2.12.3-1.el10.x86_64.rpm
```

Questions :

2. Que donne le champ `Signature` du paquet ?

> **Réponses**
>
> Le champ `Signature` vaut `(none)` : le paquet n'est pas signé (étape 9).

```
$ rpm -qp --requires hello-2.12.3-1.el10.x86_64.rpm
```

Questions :

3. Le SPEC ne contient aucune ligne `Requires:`. D'où viennent les dépendances affichées ?

> **Réponses**
>
> RPM a détecté seul la dépendance à `libc.so.6` : il n'y a pas à l'écrire (voir slide « Les macros du SPEC »).

Étape 6 : Installer le paquet

```
$ sudo dnf install ./hello-2.12.3-1.el10.x86_64.rpm
rpm -qf /usr/bin/hello
rpm -ql hello | wc -l
```

Questions :

1. Quelle commande donne le paquet qui possède `/usr/bin/hello` ?

> **Réponses**
>
> `rpm -qf`.

```
$ type -a hello
```

Questions :

2. Si le TP de compilation a été fait : pourquoi `hello` lance-t-il encore la version de `/opt/hello` ?

> **Réponses**
>
> Le `PATH` place `/opt/hello/bin` avant `/usr/bin` (fichier `/etc/profile.d/hello.sh` créé au TP précédent) : `type -a hello` montre les deux, `/usr/bin/hello` lance celui du paquet.

> **Les deux installations**
>
> `hello` compilé à la main (`/opt/hello`, inconnu de `rpm`) et le paquet (`/usr/bin`, suivi par `rpm`) coexistent. Seul le second se désinstalle avec `sudo dnf remove hello`.

Étape 7 : Créer le dépôt

Le dépôt ne reçoit que le paquet binaire :

```
$ sudo mkdir -p /srv/repo/myrepo/Packages
sudo chown -R $USER /srv/repo
cp ~/rpmbuild/RPMS/x86_64/hello-2.12.3-1.el10.x86_64.rpm \
  /srv/repo/myrepo/Packages/
createrepo_c /srv/repo/myrepo/
ls /srv/repo/myrepo/repodata
```

Questions :

1. Quel fichier de `repodata/` est l'index des autres ?
2. Quelle extension portent les fichiers de métadonnées produits ?
3. Quelle option de `createrepo_c` choisit la compression ?

> **Réponses**
>
> `repomd.xml`. Les fichiers (`primary`, `filelists`, `other`) se terminent en `.xml.zst` : c'est le choix par défaut de `createrepo_c` 1.1.2 sur AlmaLinux 10. L'option est `--compress-type` (voir slide « Contenu de `repodata/` »).

Étape 8 : Servir le dépôt et l'utiliser

Un serveur HTTP minimal, lancé en arrière-plan, suffit pour le TP :

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

Déclarer le dépôt côté client (pour l'instant sans signature), dans `/etc/yum.repos.d/myrepo.repo` créé avec un éditeur lancé en root (*`sudo`*) :

```
[myrepo]
name=Mon dépôt local
baseurl=http://localhost:8080/myrepo/
enabled=1
gpgcheck=0
```

```
$ sudo dnf remove hello
sudo dnf --disablerepo='*' --enablerepo=myrepo list available
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Questions :

1. Quel est le nom du dépôt affiché par `dnf` pour la commande `list available` ?
2. Pourquoi ajouter `--disablerepo='*' --enablerepo=myrepo` ?

> **Réponses**
>
> `myrepo` (colonne de droite de la liste). Pour que `dnf` n'interroge que le dépôt du TP, sans les dépôts AlmaLinux.

> **Un serveur de TP**
>
> `python3 -m http.server` sert tout le répertoire sans contrôle d'accès ni HTTPS. Il remplace ici un vrai serveur web, avec un dépôt en lecture seule servi en HTTPS.

Étape 9 : Créer la clé et signer le paquet

Créer une clé sans phrase secrète (pour le TP), l'exporter dans le dépôt et la déclarer à `rpmsign`. Le script de génération est un fichier à créer avec un éditeur, `/tmp/cle.batch` :

```
%no-protection
Key-Type: RSA
Key-Length: 3072
Name-Real: ACTILIS TP
Name-Email: tp@actilis.net
Expire-Date: 1y
%commit
```

Puis générer la clé, l'exporter et la déclarer :

```
$ gpg --batch --gen-key /tmp/cle.batch
gpg --armor --export "ACTILIS TP" > \
  /srv/repo/myrepo/RPM-GPG-KEY-actilis-tp
echo '%_gpg_name ACTILIS TP' >> ~/.rpmmacros
```

Signer le paquet **dans le dépôt** et le vérifier :

```
$ rpmsign --addsign /srv/repo/myrepo/Packages/hello-2.12.3-1.el10.x86_64.rpm
sudo rpm --import /srv/repo/myrepo/RPM-GPG-KEY-actilis-tp
rpm -K /srv/repo/myrepo/Packages/hello-2.12.3-1.el10.x86_64.rpm
```

Étape 10 : Exiger les signatures et provoquer les refus

Côté client, passer le dépôt en `gpgcheck=1` et `repo_gpgcheck=1`, avec la clé publique :

```
[myrepo]
name=Mon dépôt local
baseurl=http://localhost:8080/myrepo/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=http://localhost:8080/myrepo/RPM-GPG-KEY-actilis-tp
```

```
$ sudo dnf clean all
sudo dnf remove hello
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Lire l'erreur, puis **signer les métadonnées** et recommencer. Le paquet est signé, mais les métadonnées du dépôt ne le sont pas encore (et `createrepo_c --update` les reconstruit à chaque ajout) :

```
$ createrepo_c --update /srv/repo/myrepo/
gpg --detach-sign --armor /srv/repo/myrepo/repodata/repomd.xml
sudo dnf clean all
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Questions :

1. Que dit `dnf` au premier essai, avant la signature de `repomd.xml` ?
2. Quelle option du fichier `.repo` contrôle la signature des paquets, et laquelle celle des métadonnées ?
3. Quelle commande contrôle localement la signature d'un fichier RPM, comme à l'étape précédente ?

> **Réponses**
>
> `GPG signature is not available` : il cherche `repomd.xml.asc`, absent du dépôt. `gpgcheck` pour les paquets, `repo_gpgcheck` pour les métadonnées (voir slide « Config côté client »). `rpm -K fichier.rpm`, qui répond `digests signatures OK`.

> **À savoir**
>
> Dès qu'un paquet est ajouté ou modifié, `createrepo_c` réécrit `repomd.xml` : l'ancien `repomd.xml.asc` ne correspond plus et `dnf` répond `Bad PGP signature`. Il faut signer de nouveau après chaque génération. Un paquet publié sans `rpmsign` est refusé avec `Package ... is not signed`.

Étape 11 : Publier une nouvelle version *(facultative)*

Passer le paquet en `2.12.3-2`, le reconstruire, le publier et mettre le client à niveau :

```
$ cd ~/rpmbuild/SPECS
rpmdev-bumpspec -c "Reconstruction du paquet" hello.spec
```

Questions :

1. Que modifie `rpmdev-bumpspec` dans le SPEC ?

Réponses

Le `Release` passe à `2` et une entrée est ajoutée au `%changelog`.

```
$ rpmbuild -bb hello.spec
cp ~/rpmbuild/RPMS/x86_64/hello-2.12.3-2.el10.x86_64.rpm \
  /srv/repo/myrepo/Packages/
rpmsign --addsign /srv/repo/myrepo/Packages/hello-2.12.3-2.el10.x86_64.rpm
createrepo_c --update /srv/repo/myrepo/
gpg --batch --yes --detach-sign --armor \
  /srv/repo/myrepo/repodata/repomd.xml
sudo dnf clean expire-cache
sudo dnf --disablerepo='*' --enablerepo=myrepo upgrade
```

Questions :

2. Quelle commande confirme la version installée après la mise à niveau ?

Réponses

`rpm -q hello` affiche `hello-2.12.3-2.el10.x86_64`.

Arrêter le serveur de TP

`kill %1` arrête le `python3 -m http.server` lancé en arrière-plan dans cette session.

Module 6

Stockage, Swap, Systèmes de fichiers

TP : partitionner, formater, monter, puis LVM, snapshot et RAID 1

Objectif et machine

Préparer **deux disques vierges** : partitions, systèmes de fichiers, montages permanents et swap, puis un **VG LVM** (extension à chaud, réduction, snapshot, déplacement de données) et, en dernier, un **RAID 1** logiciel.

La machine

Une VM AlmaLinux 10 avec un compte `sudo`, en SSH, et **deux disques VirtIO vierges de 10 Go**, `/dev/vdb` et `/dev/vdc`, en plus du disque système `vda`.

> ⚠ Matériel à ajouter avant ce TP
>
> C'est le premier TP du cahier qui demande des disques supplémentaires : **ajouter à la VM deux disques VirtIO de 10 Go** *(dans l'hyperviseur, VM éteinte ou à chaud)*. Le nettoyage de fin de TP les laisse vierges, les TP « noyau » et « optimisation » les réutilisent.

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

> ❗ Ne toucher que `vdb` et `vdc`
>
> Les commandes de ce TP écrasent des données. Le disque `vda` porte le système : il n'apparaît dans aucune commande d'écriture.

Étape 1 : Identifier les disques

```
$ cat /proc/partitions
lsblk
```

Questions :

1. Quel préfixe de nom portent les disques de cette VM, et quel pilote les sert ?

> ℹ Réponses
>
> `/dev/vd?`, servi par `virtio-blk` (voir slide « Block devices et interfaces disques »).

```
$ sudo fdisk -l /dev/vdb
```

Questions :

2. Quelle ligne de `fdisk -l` indique le type de table de partitions de `vda` ? Et celui de `vdb` ?

> ℹ Réponses
>
> `Disklabel type: gpt` pour `vda` : la ligne n'existe pas pour `vdb`, qui n'a aucune table (voir slide « GPT : la table de partitions actuelle »).

Étape 2 : Partitionner en GPT

`sfdisk` lit un script de partitionnement. Trois partitions sur `vdb` (système de fichiers, swap, LVM) : créer le fichier `vdb.sfdisk` dans le répertoire personnel avec un éditeur, et y écrire :

```
label: gpt
,2G,L
,1G,S
,,V
```

Une seule partition sur `vdc` (LVM), dans le fichier `vdc.sfdisk` :

```
label: gpt
,,V
```

Appliquer les deux scripts :

```
$ sudo sfdisk /dev/vdb < ~/vdb.sfdisk
sudo sfdisk /dev/vdc < ~/vdc.sfdisk
```

```
$ sudo partprobe /dev/vdb /dev/vdc
lsblk -o NAME,SIZE,PARTTYPENAME,PARTUUID /dev/vdb /dev/vdc
sudo parted /dev/vdb print
```

Questions :

1. Quel type porte chacune des trois partitions de `vdb` ?
2. Quelle commande demande au noyau de relire la table ?
3. Dans quel cas la relecture immédiate n'est-elle pas possible ?

> **Réponses**
>
> `Linux filesystem`, `Linux swap`, `Linux LVM`. `partprobe` (ou `hdparm -z`). Quand une partition du disque est en cours d'utilisation : montée, swap actif (voir slide « Outils de partitionnement »).

Étape 3 : Formater et monter

```
$ sudo mkfs.ext4 -L data /dev/vdb1
sudo tune2fs -l /dev/vdb1 | grep -E \
  'volume name|UUID|Reserved block count|Block count'
sudo mkdir -p /data
sudo mount /dev/vdb1 /data
findmnt /data
df -h /data
sudo umount /data
```

Rendre le montage **permanent** avec l'UUID, puis le tester **sans redémarrer** :

```
$ sudo blkid -s UUID -o value /dev/vdb1
```

Ajouter une ligne à la fin de `/etc/fstab`, avec un éditeur lancé en root (*`sudo`*), en remplaçant l'UUID par celui relevé :

```
UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX  /data  ext4  defaults,nofail  0 2
```

Puis :

```
$ sudo systemctl daemon-reload
sudo mount -a
findmnt /data
```

Questions :

1. Pourquoi `UUID=` plutôt que `/dev/vdb1` dans `fstab` ?
2. À quoi sert `nofail` ?
3. Que vaut la valeur `pass` de `/data`, et pourquoi ?

> **Réponses**
>
> L'UUID et le label sont préférés pour leur stabilité : le nom `/dev/vd?` peut changer. `nofail` empêche l'absence du disque de bloquer le démarrage. `2` : `1` est réservé à `/`, `2` aux autres systèmes de fichiers (voir slide « Le fichier `/etc/fstab` »).

Étape 4 : Un démontage bloqué

Faire tenir `/data` par un processus, puis tenter de le démonter :

```
$ sudo chown $USER /data
(cd /data && exec sleep 600) &
sudo umount /data
sudo fuser -v /data
```

Libérer le point de montage, puis le remonter :

```
$ sudo fuser -km /data
sudo umount /data
sudo mount -a
```

Questions :

1. Quel message `umount` affiche-t-il ?
2. Quel code figure dans la colonne `ACCESS` du processus `sleep`, et que signifie-t-il ?
3. Que fait `fuser -km` ?

> **Réponses**
>
> `target is busy`. `c` : répertoire courant du processus. `-km` envoie `SIGKILL` à tous les processus qui utilisent `/data` (voir slide « Identifier le processus qui bloque un démontage »).

Étape 5 : Un swap

```
$ sudo mkswap /dev/vdb2
sudo swapon /dev/vdb2
swapon --show
free -h
```

Le rendre permanent, puis relire la configuration :

```
$ sudo blkid -s UUID -o value /dev/vdb2
```

Ajouter une ligne à la fin de `/etc/fstab`, avec un éditeur lancé en root (*`sudo`*), en remplaçant l'UUID par celui relevé :

```
≡ UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX  none  swap  defaults,nofail  0 0
```

Puis :

```
$ sudo swapoff /dev/vdb2
sudo systemctl daemon-reload
sudo swapon -a
swapon --show
```

Questions :

1. Quelle commande prépare la partition, et laquelle l'active ?
2. Combien de zones de swap `swapon --show` affiche-t-il, avec quelles priorités ?

> **Réponses**
>
> `mkswap` puis `swapon` (voir slide « Que faire avec une partition ? »). Deux zones : celle du système (`vda3`) et `vdb2`, dont la priorité est plus faible (colonne `PRIO`).

Étape 6 : Un groupe de volumes et deux volumes logiques

Les partitions `vdb3` et `vdc1` deviennent des **PV**, réunis dans un **VG** :

```
$ sudo pvcreate /dev/vdb3 /dev/vdc1
sudo vgcreate vg_tp /dev/vdb3 /dev/vdc1
sudo vgs vg_tp
sudo lvcreate -L 2G -n lv_www  vg_tp
sudo lvcreate -L 1G -n lv_logs vg_tp
```

Formater (XFS et ext4), monter et inscrire dans `fstab` :

```
$ sudo mkfs.xfs -q /dev/vg_tp/lv_www
sudo mkfs.ext4 -q -L logs /dev/vg_tp/lv_logs
sudo mkdir -p /data/www /data/logs
```

Ajouter les deux lignes suivantes à la fin de `/etc/fstab`, avec un éditeur lancé en root (*`sudo`*) :

```
≡ /dev/vg_tp/lv_www   /data/www   xfs   defaults,nofail  0  0
/dev/vg_tp/lv_logs  /data/logs  ext4  defaults,nofail  0  2
```

Puis :

```
$ sudo systemctl daemon-reload
sudo mount -a
df -h /data/www /data/logs
ls -l /dev/mapper | grep vg_tp
```

> **Disque déjà utilisé**
>
> Si le disque a servi à un TP précédent, `lvcreate` propose d'effacer l'ancienne signature (`Wipe it? [y/n]`) : répondre `y`.

Questions :

1. Quelle taille a le VG, et d'où vient-elle ?
2. Sous quels deux chemins retrouve-t-on le volume `lv_www` ?

> **Réponses**
>
> Environ 17 Go : la somme des deux PV (`vdb3` et `vdc1`). `/dev/vg_tp/lv_www` et `/dev/mapper/vg_tp-lv_www` (voir slide « LVM : concepts »).

Étape 7 : Étendre à chaud

Les deux volumes sont montés et **restent utilisables** pendant l'opération :

```
$ sudo lvextend -L +1G    --resizefs /dev/vg_tp/lv_www
sudo lvextend -L +512M  --resizefs /dev/vg_tp/lv_logs
df -h /data/www /data/logs
sudo lvs vg_tp
```

Questions :

1. Quel outil agrandit le système de fichiers XFS, et lequel l'ext4 ?
2. Quelle option de `lvextend` agrandit le système de fichiers dans la foulée ?

> **Réponses**
>
> `xfs_growfs` et `resize2fs`, appelés par `--resizefs` (messages `xfs_growfs done` et `resize2fs done`). `--resizefs`, ou `fsadm` seul (voir slides « Redimensionner un système de fichiers » et « LVM : maintenance courante »).

Étape 8 : Réduire un volume ext4

Un volume ext4 se **réduit** à froid. Une première tentative, volume monté, montre ce qui manque : répondre `n` à la question posée.

```
$ sudo lvreduce -L -500M --resizefs /dev/vg_tp/lv_logs
```

Démonter, puis réduire pour de bon et remonter :

```
$ sudo umount /data/logs
sudo lvreduce -y -L -500M --resizefs /dev/vg_tp/lv_logs
sudo mount /data/logs
df -h /data/logs
```

Questions :

1. Que demande `fsadm` quand le volume est monté ?
2. Dans quel ordre le système de fichiers et le volume sont-ils réduits ?

> **Réponses**
>
> Démonter, contrôler (`fsck`) puis lancer `resize2fs` : `File system unmount is needed for reduce`. D'abord le système de fichiers, puis le volume (voir slide « Redimensionner un système de fichiers »).

Étape 9 : Pas de réduction sur XFS

> **Pas de réduction sur `lv_www`**
>
> XFS ne se réduit pas. Sur un XFS agrandi à chaud (étape 7), `lvreduce --resizefs` a pourtant réduit le **volume** sans toucher au système de fichiers, qui dépasse alors son support et ne se monte plus (AlmaLinux 10.0, `lvm2` 2.03.28).

Étape 10 : Un snapshot

Un fichier, une photo du volume, puis une modification. Créer `/data/www/index.html` avec un éditeur lancé en root (*`sudo`*), contenant `version 1`, puis prendre la photo :

```
$ sudo lvcreate -s -L 500M -n snap_www /dev/vg_tp/lv_www
```

Modifier le fichier (`version 2`) avec l'éditeur, puis :

```
$ sudo lvs vg_tp
```

Lire la photo et le volume d'origine :

```
$ sudo mkdir -p /mnt/snap
sudo mount -o ro,nouuid /dev/vg_tp/snap_www /mnt/snap
```

Questions :

1. Pourquoi l'option `nouuid` au montage ?

> **Réponses**
>
> `nouuid` est requis sur XFS : le snapshot a le même UUID que l'origine.

```
$ cat /mnt/snap/index.html /data/www/index.html
sudo umount /mnt/snap
```

Questions :

2. Que contient `index.html` dans le snapshot, et dans le volume d'origine ?
3. Que devient un snapshot dont les 500 Mo réservés sont saturés ?

> **Réponses**
>
> `version 1` dans le snapshot, `version 2` dans l'origine. Un snapshot dont les 500 Mo réservés sont saturés devient **invalide** (l'origine reste intacte). Voir slide « Snapshots LVM ».

Étape 11 : Revenir en arrière

Fusionner le snapshot dans l'origine (retour à `version 1`) :

```
$ sudo lvconvert --merge /dev/vg_tp/snap_www
```

La fusion attend que l'origine soit **réactivée** :

```
$ sudo umount /data/www
sudo lvchange -an vg_tp/lv_www
sudo lvchange -ay vg_tp/lv_www
sudo mount /data/www
cat /data/www/index.html
sudo lvs vg_tp
```

Questions :

1. Que montrent `cat` et `lvs` après la réactivation ?

> **Réponse**
>
> `version 1`, et le snapshot a disparu de la liste : il est fusionné dans l'origine (voir slide « Snapshots LVM »).

> **À savoir**
>
> Avec l'origine montée, `lvconvert --merge` répond `Delaying merge since origin is open`. La fusion a lieu à la prochaine activation du volume et se termine en arrière-plan : si `lvs` montre encore `snap_www`, attendre quelques secondes puis lancer `sudo lvchange --refresh vg_tp/lv_www`.

Étape 12 : Libérer un disque *(facultative)*

Avant de retirer le disque `vdb` du groupe, déplacer ce qu'il contient :

```
$ sudo lvs -o lv_name,devices vg_tp
sudo pvmove /dev/vdb3
sudo lvs -o lv_name,devices vg_tp
```

Questions :

1. Sur quel(s) PV se trouvent les volumes avant et après `pvmove` ?

> **Réponses**
>
> Colonne `Devices` : `vdb3` (et `vdc1`) avant, `vdc1` seul après.

```
$ sudo vgreduce vg_tp /dev/vdb3
sudo pvremove /dev/vdb3
sudo vgs vg_tp
```

Questions :

2. Dans quel ordre les commandes `pvmove`, `vgreduce` et `pvremove` s'enchaînent-elles ?

> **Réponses**
>
> `pvmove` libère le PV, `vgreduce` le retire du VG, `pvremove` le démarque (voir slide « LVM : maintenance courante »).

Étape 13 : Remettre les disques à zéro

Retirer d'abord les lignes `fstab` ajoutées (`/data`, le swap `vdb2` et les deux volumes), puis tout défaire **dans l'ordre inverse** de la création :

Éditer `/etc/fstab` avec un éditeur lancé en root (*`sudo`*) pour retirer ces lignes, puis :

```
$ sudo umount /data/www /data/logs /data
sudo swapoff /dev/vdb2
sudo lvremove -y vg_tp
sudo vgremove -y vg_tp
sudo pvremove -y /dev/vdc1
sudo wipefs -a /dev/vdb1 /dev/vdb2 /dev/vdb
sudo wipefs -a /dev/vdc1 /dev/vdc
sudo systemctl daemon-reload
lsblk /dev/vdb /dev/vdc
```

> **`fstab` avant les disques**
>
> Une ligne de `fstab` sans `nofail` qui désigne un disque disparu bloque le démarrage de la VM. Les lignes de ce TP portent `nofail`, mais les retirer reste préférable.

> **À savoir**
>
> `wipefs -a` efface les signatures (système de fichiers, LVM, table de partitions). `rmdir /data/www /data/logs /data` retire les points de montage devenus inutiles.

Étape 14 : Un RAID 1 logiciel *(facultative)*

Une partition de 1 Go de type RAID sur chaque disque, assemblées en miroir :

```
$ sudo dnf install mdadm
for d in vdb vdc; do echo ',1G,R' | sudo sfdisk --label gpt /dev/$d; done
sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/vdb1 \
  /dev/vdc1
cat /proc/mdstat
```

Formater, monter, écrire un fichier, puis **déclarer un disque en panne** :

```
$ sudo mkfs.xfs -q /dev/md0
sudo mkdir -p /mnt/raid && sudo mount /dev/md0 /mnt/raid
```

Créer `/mnt/raid/fichier` avec un éditeur lancé en root (`sudo`), contenant `ok`, puis :

```
$ sudo mdadm /dev/md0 --fail /dev/vdc1
sudo mdadm --detail /dev/md0 | grep -E 'State|Working|Failed'
cat /mnt/raid/fichier
```

Questions :

1. Que lit-on dans `/proc/mdstat` sur le niveau, les membres et l'état de `md0` ?
2. Combien de disques le RAID 1 tolère-t-il de perdre ? Que montre `mdadm --detail` après la panne simulée, et le fichier est-il lisible ?

Réponses

`raid1`, les membres `vdb1` et `vdc1`, l'état de synchronisation (`resync`). Le RAID 1 tolère la perte d'**un** disque (voir slide « RAID : compromis sécurité / performances / coût »). L'état devient `degraded` et le fichier reste lisible.

À savoir

Dans `/proc/mdstat`, `[UU]` désigne deux membres actifs, `[U_]` un membre manquant ou en panne.

Défaire

`sudo umount /mnt/raid`, `sudo mdadm --stop /dev/md0`, `sudo mdadm --zero-superblock /dev/vdb1 /dev/vdc1`, puis `wipefs -a` sur les partitions et sur les deux disques.

Module 7

Périphériques, modules et udev

TP : lire le matériel, manipuler des modules, écrire une règle udev

Objectif et machine

Lire le **matériel** vu par le noyau, retrouver le **module** d'un périphérique, charger et configurer un module, créer un **fichier spécial** à la main, puis laisser **udev** le faire avec une règle écrite par soi.

La machine

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

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

> **Un module « sans matériel »**
>
> La VM n'a pas de périphérique rare à manipuler. Le module `dummy` (interfaces réseau factices) sert de cobaye à l'étape 3, et un fichier monté en disque (`loop`) sert de « clé USB » à l'étape 8.

Étape 1 : Détecter le matériel

`lspci` et `lsusb` viennent des paquets `pciutils` et `usbutils`, absents d'une installation minimale :

```
$ sudo dnf install pciutils usbutils
lspci
lspci -n
lspci -nn | grep -i ethernet
lsusb
```

Les noms viennent de la base `pci.ids` :

```
$ ls -l /usr/share/hwdata/
grep -m1 -A3 '^1af4' /usr/share/hwdata/pci.ids
```

Questions :

1. Quels sont l'emplacement, la classe, le constructeur et le périphérique du contrôleur Ethernet dans la sortie de `lspci -n` ?
2. Quel fichier `lspci` consulte-t-il pour afficher des noms, et que devient le contrôleur Ethernet sans lui ?
3. Quelle commande met à jour cette base ?

> **Réponses**
>
> La ligne `01:00.0 0200: 1af4:1041` (valeurs de la VM de test, à relire sur la sortie) donne : bus 01, slot 00, fonction 0, classe `0200` (Ethernet), constructeur `1af4`, périphérique `1041`. `/usr/share/hwdata/pci.ids` : sans lui, seuls les nombres s'affichent. `update-pciids` (voir slides « Détecter le matériel : lspci et lsusb » et « Les bases de noms : pci.ids et usb.ids »).

Étape 2 : Quel module pour quel périphérique ?

Le contrôleur SATA de la VM sert d'exemple. Relever son identifiant, puis chercher le module dans `modules.alias` :

```
$ lspci -nn | grep -i sata
grep -i 'pci:v00008086d00002922' /lib/modules/$(uname -r)/modules.alias
modinfo ahci | head -5
lsmod | grep ahci
```

Questions :

1. Quel module `modules.alias` associe-t-il à ce contrôleur ?
2. Où se trouve le fichier du module, et dans quel format ?
3. Ce module est-il chargé ?

Réponses

`ahci`, dans la dernière colonne de la ligne `alias`. Le fichier `ahci.ko.xz` est dans `/lib/modules/<version>/kernel/drivers/ata/` : un module compressé en xz (voir slides « Quel module pour quel périphérique ? » et « Configuration : /etc/modprobe.d et /lib/modules »). `lsmod` le montre chargé, avec ses dépendances `libahci` et `libata`.

À savoir

Certains pilotes ne figurent pas dans `modules.alias` parce qu'ils sont intégrés au noyau : c'est le cas du pilote `virtio-pci` des autres périphériques de cette VM. `lspci -k` affiche le pilote utilisé (`Kernel driver in use`).

Étape 3 : Charger et décharger

Le module `dummy` crée des interfaces réseau factices :

```
$ modinfo dummy | head -5
sudo modprobe dummy
lsmod | grep dummy
ip -br link | grep dummy
sudo modprobe -r dummy
```

Le même, avec un paramètre sur la ligne de commande :

```
$ sudo modprobe dummy numdummies=2
ip -br link | grep dummy
sudo modprobe -r dummy
```

Questions :

1. Quelle commande donne le chemin du fichier d'un module, ses dépendances et ses paramètres ?
2. `modprobe dummy` seul crée-t-il une interface ? Et avec `numdummies=2` ?
3. Quelle commande décharge le module ?

Réponses

`modinfo dummy`. Sans paramètre, aucune interface : le module est chargé (`lsmod`) mais ne crée rien. Avec `numdummies=2`, deux interfaces `dummy0` et `dummy1`. `modprobe -r` (ou `rmmod`).

Étape 4 : La configuration dans `/etc/modprobe.d`

Rendre le paramètre permanent. Créer `/etc/modprobe.d/tp-dummy.conf` avec un éditeur lancé en root (*`sudo`*) :

```
options dummy numdummies=2
```

Puis charger le module sans rien préciser :

```
$ sudo modprobe dummy
ip -br link | grep dummy
```

Questions :

1. Que fait le fichier `tp-dummy.conf` au chargement du module ?

> **Réponses**
>
> Il passe `numdummies=2` à chaque chargement, quel que soit l'appel.

```
$ sudo modprobe -c | grep 'options dummy'
sudo modprobe -r dummy
```

Questions :

2. Combien de lignes `options dummy` la configuration contient-elle, et laquelle l'emporte ?

> **Réponses**
>
> Deux lignes : `numdummies=0` (posée par le paquet `systemd-udev`) et `numdummies=2`. La dernière lue l'emporte, ici celle du fichier de ce TP.

> **L'ordre des fichiers**
>
> Les fichiers sont lus dans l'ordre alphabétique de leur nom, tous répertoires confondus. Un fichier nommé `dummy.conf` passerait **avant** celui de `systemd.conf` et verrait sa valeur écrasée : aucune interface ne serait créée.

Étape 5 : `blacklist` et `install`

Remplacer le contenu de `/etc/modprobe.d/tp-dummy.conf` avec un éditeur lancé en root (*`sudo`*) par :

```
blacklist dummy
```

Puis :

```
$ sudo modprobe dummy
lsmod | grep dummy
sudo modprobe -r dummy
```

Questions :

1. Le module `dummy` est-il chargé malgré `blacklist` ?

> **Réponses**
>
> Oui : `blacklist` n'empêche pas un `modprobe` explicite.

Remplacer le contenu du même fichier par :

```
install dummy /bin/false
```

Puis :

```
$ sudo modprobe dummy
echo "code retour : $?"
lsmod | grep -c dummy
```

Questions :

2. Que répond `modprobe` avec la ligne `install` ?

> **Réponses**
>
> Avec `install dummy /bin/false`, `modprobe` échoue (`Error running install command '/bin/false'`, code retour 1) et le module n'est pas chargé : c'est ce qui bloque vraiment (voir slide « Configuration : /etc/modprobe.d et /lib/modules »).

Retirer ensuite le fichier :

```
$ sudo rm /etc/modprobe.d/tp-dummy.conf
```

Étape 6 : Les fichiers spéciaux

```
$ ls -l /dev/vda /dev/null /dev/tty0
```

Questions :

1. Quel est le type, le majeur et le mineur de `/dev/vda` et de `/dev/null` ?

> **Réponses**
>
> `b` pour `/dev/vda` (disque), `c` pour `/dev/null` (majeur 1, mineur 3).

```
$ head -12 /proc/devices
```

Questions :

2. Que lit-on dans `/proc/devices` ?

> **Réponses**
>
> `/proc/devices` liste les majeurs que le noyau actuel peut gérer, en caractère puis en bloc.

Créer son propre fichier spécial avec le couple de `/dev/null` :

```
$ sudo mknod -m 666 /dev/monnull c 1 3
ls -l /dev/monnull
echo test > /dev/monnull
echo "code retour : $?"
sudo rm /dev/monnull
```

Questions :

3. Que se passe-t-il quand on écrit dans `/dev/monnull` ?

> **Réponses**
>
> L'écriture réussit sans rien afficher : `/dev/monnull` est un second accès au même pilote que `/dev/null` (voir slides « Fichiers spéciaux : majeur et mineur » et « Créer un fichier spécial : mknod »).

Étape 7 : Observer udev

Suivre les évènements pendant qu'un module crée une interface. Ouvrir une **seconde session SSH** pour le moniteur :

```
$ sudo udevadm monitor --udev --subsystem-match=net
```

Dans la première session :

```
$ sudo modprobe dummy numdummies=1
sudo modprobe -r dummy
```

Les informations et attributs d'un disque :

```
$ udevadm info -q all -n vda | head -12
udevadm info -a -p /sys/block/vda | head -14
```

Questions :

1. Quels évènements `udevadm monitor` affiche-t-il pour `modprobe dummy` puis `modprobe -r dummy` ?
2. Quels liens `S:` udev a-t-il créés pour `vda` ?
3. Pour quoi servent les lignes `ATTR{...}` d'`udevadm info -a` ?

> **Réponses**
>
> `add` puis `remove` du périphérique `dummy0` (sous-système `net`). Des liens dans `/dev/disk/by-path` et `/dev/disk/by-diskseq`. Ce sont les attributs (`ATTR`, `ENV`) sur lesquels s'appuient les règles (voir slide « udevadm : observer et tester »).

> **À savoir**
>
> Arrêter le moniteur avec Ctrl-C.

Étape 8 : Une règle `SYMLINK`

Un fichier de 50 Mo, formaté avec un **label** et présenté comme un disque, joue la clé USB :

```
$ cd /tmp
truncate -s 50M tpudev.img
mkfs.ext4 -q -L TPDISK tpudev.img
L=$(sudo losetup -f --show tpudev.img)
echo $L
sleep 3
udevadm info -q property -n $L | grep ID_FS_LABEL
ls -l /dev/disk/by-label/
```

La règle : un lien `/dev/mondisque` pour ce label, et une trace dans le journal.
Créer `/etc/udev/rules.d/99-tp.rules` avec un éditeur lancé en root (*`sudo`*) :

```
ENV{ID_FS_LABEL}=="TPDISK", SYMLINK+="mondisque"
ACTION=="change", ENV{ID_FS_LABEL}=="TPDISK", RUN+="/usr/bin/logger -t udev TP : %k"
```

Étape 9 : Tester la règle

Simuler, puis **rejouer l'évènement** du disque : il est traité quelques secondes plus tard.

```
$ sudo udevadm test /sys/class/block/$(basename $L) 2>&1 | grep mondisque
sudo udevadm trigger --action=change --name-match=$L
sleep 3
ls -l /dev/mondisque
sudo journalctl -t udev -n 3
```

Détacher le disque, vérifier que le lien disparaît, puis nettoyer :

```
$ sudo losetup -d $L
sleep 2
ls /dev/mondisque
sudo rm /etc/udev/rules.d/99-tp.rules /tmp/tpudev.img
```

Questions :

1. Quelle clé de la règle compare le label, et quelle clé crée le lien ?
2. Vers quel périphérique `/dev/mondisque` pointe-t-il, et que devient-il quand le disque est détaché ?
3. Quelle commande rejoue les évènements et laquelle teste les règles sans toucher au périphérique ?

Réponses

`ENV{ID_FS_LABEL}==` compare, `SYMLINK+=` crée le lien. Il pointe vers `loop0` (ou le premier `loop` libre) et disparaît avec lui, car udev supprime les liens des périphériques retirés. `udevadm trigger` rejoue les évènements, `udevadm test` simule les règles (voir slide « udevadm : observer et tester »).

À savoir

Un disque `loop` ne produit que des évènements `change`, d'où `ACTION=="change"` dans la deuxième règle. Pour une vraie clé USB, la règle de la slide « Autres exemples de règles » (`ACTION=="add"`) convient. Les règles sont relues sans `udevadm control --reload`, mais les évènements sont traités en arrière-plan : d'où les `sleep`.

Module 8

Le noyau : compilation et installation

TP : compiler un noyau réduit, l'installer, ranger les fichiers

Objectif et machine

Compiler un noyau **depuis les sources de kernel.org** avec une configuration **réduite** (seulement ce que la VM utilise), l'installer, constater ce que `make install` ajoute à `/boot`, puis **tout défaire**.

La machine

Une VM AlmaLinux 10 avec un compte `sudo`, **2 CPU ou plus**, et un **disque supplémentaire de 10 Go** (`/dev/vdb`, vierge : celui du TP « Stockage », laissé vierge par son nettoyage, ou un disque ajouté à la VM) pour `/usr/src`. Tout se fait en SSH.

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

Secure Boot

Un noyau compilé n'est pas signé : si Secure Boot est actif, le firmware refuse de le démarrer. L'étape 1 relève l'état de la VM, l'étape 7 en tient compte.

Durée

La compilation seule demande une quinzaine de minutes avec 2 CPU. Elle se lance en arrière-plan et se suit dans un journal.

Étape 1 : Le noyau en place

```
$ uname -r
ls -l /boot
```

Questions :

1. Quels fichiers de `/boot` portent la version du noyau, et à quoi sert chacun ?

Réponses

`vmlinuz-` (l'image compressée), `initramfs-` (le ramdisk construit à l'installation), `config-` (la configuration ayant servi à le compiler) et `System.map-` (adresses et noms de symboles), voir slide « Le noyau standard ».

```
$ sudo grubby --default-kernel
grep -c '=m$' /boot/config-$(uname -r)
grep '^CONFIG_MODULES=' /boot/config-$(uname -r)
```

Questions :

2. Combien de modules le noyau de la distribution compile-t-il ?

Réponses

Environ 2400 modules, relevés par `grep -c`.

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

Questions :

3. La VM démarre-t-elle en UEFI ou en BIOS ? Secure Boot est-il actif ?

> **Réponses**
>
> `mokutil` répond `SecureBoot enabled` ou `disabled` sur une VM UEFI. Sur une VM BIOS, `mokutil` signale que les variables EFI ne sont pas gérées (ou est absent) : Secure Boot n'existe pas, le noyau compilé démarre.

Étape 2 : Le disque et les paquets

Un système de fichiers pour `/usr/src` sur le disque supplémentaire :

```
$ echo ',,L' | sudo sfdisk --label gpt /dev/vdb
sudo mkfs.ext4 -q -L usrsrc /dev/vdb1
sudo mount /dev/vdb1 /usr/src
sudo chown $USER /usr/src
df -h /usr/src
```

Les paquets de la slide « Prérequis de compilation » :

```
$ sudo dnf group install "Development Tools"
sudo dnf install ncurses-devel openssl openssl-devel \
  elfutils-libelf-devel bc flex bison perl dwarves rsync
```

> **Sans la commande `openssl`**
>
> La compilation s'arrête à l'étape `GENKEY` avec `openssl: command not found` si seul `openssl-devel` est installé.

Étape 3 : Les sources

```
$ cd /usr/src
curl -O https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.12.112.tar.xz
```

Contrôler la somme (relevée sur kernel.org le 2026-10-06). Créer le fichier `linux.sha256` dans `/usr/src` avec un éditeur lancé en root (*`sudo`*), contenant la somme, deux espaces, puis le nom de l'archive :

```
164dc9d1f6c93c61a15e1f071c48379b467f2b17c469cce7223471968208ed03  linux-6.12.112.tar.xz
```

Puis contrôler et extraire :

```
$ sha256sum -c linux.sha256
tar xJf linux-6.12.112.tar.xz
du -sh linux-6.12.112
```

Questions :

1. De quelle branche de kernel.org la version `6.12.x` fait-elle partie, et pourquoi la choisir pour un serveur ?
2. Quelle taille occupent les sources extraites ?

> **Réponses**
>
> La branche **longterm**, à support étendu : le noyau de RHEL 10 repose sur la 6.12 (voir slide « Récupérer les sources du noyau »). Environ 1,6 Go avant compilation, et plus du double après.

> **À savoir**
>
> Le numéro de la dernière 6.12 évolue : la page kernel.org donne la version courante, et le nom d'archive comme la somme sont à adapter.

Étape 4 : Configurer

Repartir de la configuration en place, la réduire aux modules chargés, puis suffixer la version pour ne pas confondre ce noyau avec celui de la distribution :

```
$ cd /usr/src/linux-6.12.112
make mrproper
cp /boot/config-$(uname -r) .config
yes "" | make localmodconfig
scripts/config --set-str LOCALVERSION "-tp"
scripts/config --disable DEBUG_INFO_BTF --disable \
  DEBUG_INFO_DWARF_TOOLCHAIN_DEFAULT
make olddefconfig
```

Comparer :

```
$ grep -c '=m$' /boot/config-$(uname -r) .config
make -s kernelrelease
```

Questions :

1. Combien de modules la configuration contient-elle avant et après `localmodconfig` ?
2. Quelle version `make kernelrelease` annonce-t-il ?
3. Que produit `make mrproper`, et que remet-il à zéro ?

> **Réponses**
>
> Environ 2400 avant, une centaine après : seuls les modules **chargés** au moment de la commande sont gardés. `6.12.112-tp`. `mrproper` nettoie tout, y compris `.config` (voir slide « Étapes 1 et 2 : nettoyer et configurer »).

> **À savoir**
>
> `yes ""` accepte la valeur par défaut des options apparues depuis la version de la distribution. Les deux options `--disable` retirent les informations de débogage (`DEBUG_INFO_NONE`), qui alourdissent surtout la dernière étape de la compilation.

Étape 5 : Compiler

En arrière-plan, pour survivre à une coupure SSH :

```
$ cd /usr/src/linux-6.12.112
date +%T
nohup make -j$(nproc) > ~/build.log 2>&1 &
tail -f ~/build.log
```

Arrêter `tail` avec Ctrl-C. À la fin :

```
$ grep 'is ready' ~/build.log
date +%T
ls -lh arch/x86/boot/bzImage /boot/vmlinuz-$(uname -r)
ls /lib/modules
```

Questions :

1. Quel fichier est le noyau bootable, et de quelle taille par rapport à celui de la distribution ?
2. Les modules compilés sont-ils déjà dans `/lib/modules` ?

> **Réponses**
>
> `arch/x86/boot/bzImage` : environ 14 Mo contre 16 Mo. Non : ils sont compilés mais pas installés, c'est l'étape suivante (voir l'étape 4).

> **À savoir**
>
> Si `make` s'arrête, le message d'erreur est dans `~/build.log` : `grep -n -B3 'Error' ~/build.log`. Relancer `make` reprend où il s'était arrêté.

Étape 6 : Installer les modules, puis le noyau

Dans cet ordre, la cible `install` supposant les modules déjà installés :

```
$ cd /usr/src/linux-6.12.112
sudo make modules_install INSTALL_MOD_STRIP=1
du -sh /lib/modules/6.12.112-tp /lib/modules/$(uname -r)
sudo make install
```

Ce que `make install` a ajouté :

```
$ ls -l /boot | grep tp
sudo sh -c 'cat /boot/loader/entries/*6.12.112-tp.conf'
sudo grubby --default-kernel
sudo grubby --info=ALL | grep -E '^(index|title)'
```

Questions :

1. Quelle taille occupent les modules installés, et celle des modules de la distribution ? Que fait `INSTALL_MOD_STRIP=1` ?
2. Quels fichiers `make install` a-t-il créés dans `/boot` ?
3. Quel est maintenant le noyau par défaut ?

> **Réponses**
>
> Une vingtaine de Mo contre une centaine : peu de modules, et `INSTALL_MOD_STRIP=1` les allège. `vmlinuz-6.12.112-tp` et `initramfs-6.12.112-tp.img`, avec une entrée de menu dans `/boot/loader/entries/` : c'est ce que fait `make install`, sans `dracut` ni `grubby` à la main (voir slide « Étapes 5 et 6 : installer »). Le nouveau : `make install` en fait l'entrée par défaut.

> **Noyau par défaut**
>
> Avec Secure Boot actif, un redémarrage sans autre précaution tomberait sur un noyau refusé. L'étape suivante remet l'ancien noyau par défaut.

Étape 7 : Démarrer ce noyau, ou pas

Secure Boot actif (réponse de l'étape 1, VM UEFI seulement) : ne pas redémarrer sur le nouveau noyau. Remettre celui de la distribution par défaut :

```
$ sudo grubby --set-default /boot/vmlinuz-$(uname -r)
sudo grubby --default-kernel
```

Secure Boot inactif : redémarrer sur le noyau compilé, puis lire la version et les modules chargés.

```
$ sudo reboot
```

```
$ uname -r
lsmod | wc -l
ls /boot/vmlinuz-*
sudo grubby --set-default /boot/vmlinuz-<version de la distribution>
```

Questions :

1. Que répond `uname -r` après le redémarrage sur le noyau compilé ?
2. Pourquoi remettre le noyau de la distribution par défaut avant de continuer ?

> **Réponses**
>
> `6.12.112-tp`. Ce noyau n'est pas signé et ne reçoit aucun correctif de sécurité de la distribution : l'ancien reste le repli (voir slide « Bonnes pratiques »).

> **À savoir**
>
> La version de la distribution est celle notée à l'étape 1 (`uname -r` avant le redémarrage). Au menu GRUB, une touche choisit un autre noyau pour ce seul démarrage.

Étape 8 : Tout défaire

Revenir sur le noyau de la distribution s'il ne l'est pas, puis retirer le noyau de test, ses modules et les sources :

```
$ uname -r
sudo kernel-install remove 6.12.112-tp
sudo rm -r /lib/modules/6.12.112-tp
ls /boot /lib/modules
sudo grubby --default-kernel
rm ~/build.log
cd ~
sudo umount /usr/src
sudo wipefs -a /dev/vdb1 /dev/vdb
```

Questions :

1. Quelle commande retire le noyau, son initramfs et son entrée de menu ?
2. Que reste-t-il à supprimer à la main, et pourquoi ?

> **Réponses**
>
> `kernel-install remove 6.12.112-tp`. Le répertoire `/lib/modules/6.12.112-tp` : la commande ne le supprime pas.

> **Ne pas retirer le noyau en cours**
>
> `kernel-install remove` sur le noyau sur lequel la machine tourne laisse un système sans `/boot` cohérent. `uname -r` donne le noyau actif : il ne doit pas être celui qu'on retire.

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` ?

> ℹ **Réponses**
>
> `uname -r`, aussi dans `/proc/version`. `journalctl -k` conserve les messages du noyau au-delà du *ring buffer* de `dmesg` (voir slide « Commandes et fichiers d'information »). Le message apparaît dans les deux. `dmesg` demande `sudo` (`kernel.dmesg_restrict=1`).

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 ?

> **Réponses**
>
> `root=` désigne le volume racine et `ro` le monte en lecture seule au départ (voir slide « La ligne de commande du noyau »). Le paramètre n'apparaît dans `/proc/cmdline` qu'après un redémarrage : `grubby` modifie les entrées de démarrage, pas le noyau qui tourne.

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` ?

> **Réponse**
>
> `loglevel=5` figure en fin de ligne et le premier chiffre de `kernel/printk` est passé à `5` (7 auparavant). Le paramètre est persistant : `grubby` l'a écrit dans les entrées de démarrage (voir slide « La ligne de commande du noyau »).

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 ?

> **Réponses**
>
> Les fichiers de `/proc` ont une taille nulle : le noyau les produit à la lecture. Un répertoire numérique correspond à un processus (`$$` est celui du shell courant). Le routage est activé (`1`), mais seulement jusqu'au redémarrage : `echo` vers `/proc/sys` est volatile (voir slide « /proc et /proc/sys »).

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 ?

> **Réponses**
>
> `vm.swappiness` vaut 60 par défaut : plus la valeur est basse, moins le noyau swappe (voir slide « Swap, cache et écritures disque »).

```
$ 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` ?

> **Réponses**
>
> `kernel.osrelease` fait partie des informations non modifiables (voir slide « Quelques paramètres kernel utiles »).

```
$ 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 ?

> **Réponses**
>
> Les classes vont de `abi` à `vm` (dont `kernel`, `net`, `fs`). `net` est la plus peuplée (voir slide « Panorama des classes de paramètres »).

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`.

Réponses

`buff/cache` baisse : `drop_caches` est un déclencheur ponctuel, utile pour des mesures (voir slide « Swap, cache et écritures disque »). En mode `2`, la limite est `swap + overcommit_ratio % de la RAM` (voir slide « La surcharge mémoire : OOM killer et overcommit ») : 1 Go de swap + 50 % de 1,6 Go de RAM, soit environ 1,8 Go (affiché dans tous les modes, appliqué seulement en mode `2`).

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 ?

Réponses

`0` : le fichier n'est lu qu'au démarrage ou par `sysctl --system` (voir slide « sysctl : l'interface propre vers /proc/sys »). Après le redémarrage, `ip_forward` vaut `1` et `vm.swappiness` vaut `10`.

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 ?

> ℹ **Réponses**
>
> `pxelinux.0` est servi par TFTP, le kickstart par HTTP. C'est l'argument `inst.ks=` de la ligne `append`.

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 ?

> **Réponses**
>
> L'archive est `backup.tar.gz`, l'ISO s'appelle `rear-` suivi du nom d'hôte. L'ISO ne contient que le système de secours : les données sont sur le partage et le système de secours les y reprend.

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
```

> **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 ?

> **Réponses**
>
> `rear recover` depuis le support de secours, à partir du partage NFS. L'adresse IP et l'adresse MAC changent : le nom d'hôte, lui, est identique, donc deux machines de même nom coexistent si la source reste allumée.

> **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 ?

> ℹ Réponses
>
> Le test `/sys/firmware/efi` donne le mode.

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

Questions :

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

> ℹ Réponses
>
> GRUB charge `vmlinuz-...` et `initramfs-...img`.

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

Questions :

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

> **Réponses**
>
> Les entrées sont des fichiers BLS, un par noyau, dans `/boot/loader/entries/`.

> **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 ?

> **Réponses**
>
> Absents au premier démarrage, présents au second : l'édition ne dure que pour ce démarrage et n'est pas enregistrée dans le fichier de configuration. Pour une modification durable : `grubby`, voir slide « Modifier une option de démarrage du noyau ».

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 ?

Réponses

`ls (hd0,gptN)/`. Le paramètre `root=` (voir slide « La chaîne de démarrage »).

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` ?

Réponses

`rd.break`. Sous `/sysroot`. Il demande à SELinux de réétiqueter les fichiers au démarrage suivant : sans lui, la connexion avec le nouveau mot de passe peut échouer.

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.

> ℹ **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) ?

> ℹ Réponses
>
> `systemd 257`. Le lien de `/usr/lib` pointe vers `graphical.target`, la cible par défaut du paquet. Celle de la VM est `multi-user.target` : un lien dans `/etc/systemd/system` la surcharge. `0` quand le système est `running`, un code différent de 0 sinon (voir slides « Les targets, sortes de runlevels » et « Connaître l'état du système »).

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 ?

> **Réponses**
>
> Dans `/etc/systemd/system` : `default.target` est un lien symbolique vers la cible choisie.

```
$ 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 ?

> **Réponses**
>
> Le fichier de `/usr/lib` n'a pas bougé : `set-default` le surcharge, et un paquet peut donc se mettre à jour sans écraser ce choix (voir slide « Les targets, sortes de runlevels »).

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` ?

> **Réponses**
>
> `enabled` et `inactive` : activé au démarrage, mais pas encore lancé.

```
$ 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 ?

> **Réponses**
>
> `WantedBy=` de la section `[Install]`. `enable` crée un lien dans le répertoire `multi-user.target.wants` (voir slide « Dépendances entre unités »).

> **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 ?

> **Réponses**
>
> `override.conf` dans `/etc/systemd/system/monservice.service.d/` (voir slide « Surcharge d'une unité »). `systemctl cat` affiche aussi le fichier de surcharge, à la suite de l'unité. Oui : `Restart=` relance le processus mort après `RestartSec=`. Le journal indique `Scheduled restart job`.

> **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 ?

> **Réponses**
>
> `degraded`, avec un code différent de 0. La ligne `Failed: 1 units`. `systemctl reset-failed` (voir slide « Connaître l'état du système »).

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` ?

> **Réponse**
>
> Non : `disable` supprime le lien de `multi-user.target.wants`, et `--now` arrête aussi le service (voir slide « Équivalences systemctl / service / chkconfig »).

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 ?

> **Réponses**
>
> Deux messages, puis un seul : `-p info` retient les niveaux jusqu'à `info` et écarte le message `debug`.

```
$ 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 ?

> **Réponses**
>
> `PRIORITY=6` (`info`) et `SYSLOG_FACILITY=16` (`local0`) donnent `PRI = 16 × 8 + 6 = 134` (voir slide « Syslog : la convention commune »).

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

Questions :

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

Réponses

`/dev/log` pointe vers `/run/systemd/journal/dev-log` : c'est `journald` qui reçoit les messages de `syslog()` (voir slide « `/dev/log` : la porte d'entrée »).

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 ?

Réponses

`-u` (`--unit`), `-k`, `--since` et `--until`, `-g` (voir slide « `journalctl` : lire le journal »). `-p warning` renvoie aussi les niveaux plus graves (`err`, `crit`…), comme `-p info` à la question précédente.

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` ?

Réponses

`syslog`, `stdout`, `journal`, `driver` et `kernel` : `logger` passe par `/dev/log`, donc `syslog`.

```
$ 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 ?

Réponses

`systemd-cat -p warning` donne `PRIORITY=4` avec le transport `stdout` (voir slide « Les champs d'un message »).

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

Questions :

3. Quelle option donne le message au format JSON ?

> **Réponses**
>
> `-o json-pretty`.

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 ?

> **Réponses**
>
> Un seul démarrage : `/var/log/journal` n'existe pas et `Storage` n'est pas défini.

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

Questions :

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

> **Réponses**
>
> La valeur par défaut est `auto`, donc le stockage est **volatile**, dans `/run/log/journal` (voir slide « Persistance du journal »).

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 ?

> **Réponses**
>
> Deux démarrages. Le message est retrouvé avec `-b -1`, le démarrage précédent : le journal est écrit sur disque. `journalctl --flush` y a copié le journal en mémoire : sans lui, les messages précédant le redémarrage seraient perdus.

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` ?

> **Réponses**
>
> `SystemMaxUse`, `SystemKeepFree`, `SystemMaxFileSize` et `MaxRetentionSec` (voir slide « Limiter l'espace occupé par le journal »). C'est `journald` lui-même, d'après ces paramètres.

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

Questions :

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

> **Réponses**
>
> Sur une VM récente, le journal est plus petit que 20 Mo : `--vacuum-size` ne supprime rien.

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` ?

Module 14

Analyse de la charge et de l'activité système

TP : charge, outils instantanés et sysstat

Objectif et machine

Produire une charge **CPU**, puis une charge d'**I/O**, et les lire avec les outils instantanés, la **PSI** et `sysstat`.

La machine

Une VM AlmaLinux 10 avec au moins 2 CPU, un accès à Internet et un compte qui dispose de `sudo`. Tout se fait en SSH, avec **deux sessions** ouvertes en parallèle : l'une produit la charge, l'autre l'observe.

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

> **Un redémarrage au programme**
>
> L'étape 1 active la PSI : elle demande un redémarrage de la VM.

Étape 1 : Activer PSI

```
$ ls /proc/pressure
sudo grubby --update-kernel=ALL --args=psi=1
sudo grubby --info=DEFAULT | grep -o 'psi=1'
sudo systemctl reboot
```

Après le redémarrage :

```
$ ls /proc/pressure
cat /proc/pressure/cpu /proc/pressure/io /proc/pressure/memory
```

Questions :

1. Que répond `ls /proc/pressure` avant et après ?
2. Quelles lignes donnent `/proc/pressure/io` ? Que signifient `some` et `full` ?

> **Réponses**
>
> Avant : « No such file or directory » (PSI est compilée mais désactivée). Après : `cpu`, `io` et `memory`. Les lignes `some` (au moins une tâche bloquée) et `full` (toutes les tâches bloquées) donnent des moyennes sur 10, 60 et 300 secondes (voir slide « Interpréter la charge »).

```
$ nproc
cat /proc/loadavg
```

Questions :

3. Combien de CPU la VM a-t-elle ?

> **Réponses**
>
> `nproc` donne le nombre de CPU, à comparer à la charge.

Étape 2 : Une charge CPU

Dans la **session 1**, lancer trois calculs en arrière-plan :

```
$ for i in 1 2 3; do sha256sum /dev/zero > /dev/null & done
jobs
```

Dans la **session 2**, après une minute :

```
$ uptime
```

Questions :

1. Quelle charge `uptime` affiche-t-il, face au nombre de CPU ?

> **Réponses**
>
> Une charge proche de 3 pour 2 CPU : plus de tâches runnable que de CPU (voir slide « uptime et /proc/loadavg »).

```
$ top -b -n1 | head -12
vmstat -w 1 3
```

Questions :

2. Que valent `r`, `id` et `wa` dans `vmstat` ?

> **Réponses**
>
> `r` vaut 3, `id` et `wa` sont à 0 : le CPU est saturé.

```
$ mpstat -P ALL 1 1
pidstat -u 1 1
```

Questions :

3. Quel outil désigne les processus responsables ?

> **Réponses**
>
> `top` et `pidstat -u` désignent les trois `sha256sum` (voir slides « top, htop, btop » et « pidstat »).

Étape 3 : Par CPU, puis arrêter la charge

`mpstat -P ALL` détaille l'usage de chaque coeur. Dans la session 1, arrêter les calculs :

```
$ kill $(jobs -p)
jobs
```

Dans la session 2, avant puis après l'arrêt :

```
$ cat /proc/pressure/cpu
uptime
```

Questions :

1. `mpstat` montre-t-il un CPU plus chargé que l'autre ?
2. Que donne `some` dans `/proc/pressure/cpu` pendant la charge ?
3. La charge retombe-t-elle tout de suite après l'arrêt ? Pourquoi ?

> **Réponses**
>
> Les deux CPU sont saturés à environ 98 % en `%usr` (voir slide « mpstat (par CPU) et free (mémoire) »). `some` est supérieur à 0 : des tâches attendent un CPU. La charge à 1 minute redescend progressivement : c'est une moyenne glissante, comme le `avg10` de PSI, qui retombe plus vite que les moyennes de 5 et 15 minutes.

Étape 4 : Charge élevée, CPU libre

Session 1 : 15 000 blocs de 4 Kio synchronisés sur le disque. Sans `oflag=dsync`, `dd` écrit dans le cache de l'hôte et aucune attente n'apparaît.

```
$ dd if=/dev/zero of=/var/tmp/tp-io bs=4k count=15000 oflag=dsync
rm /var/tmp/tp-io
```

Session 2, pendant l'écriture :

```
$ vmstat -w 1 3
sar -u 1 3
```

Questions :

1. Quelles colonnes de `vmstat` et de `sar -u` montrent l'attente d'I/O ?

> **Réponses**
>
> `wa` dans `vmstat`, `%iowait` dans `sar -u` : environ 45 % avec un CPU à moitié libre.

```
$ iostat -x -d -k vda 1 2
```

Questions :

2. Que valent `%util` et `w_await` dans `iostat` ?

> **Réponses**
>
> `%util` proche de 90 %, `w_await` autour de 1 ms.

```
$ pidstat -d 1 2
```

Questions :

3. Quel processus engendre les écritures ? Quel outil le montre ?

> **Réponses**
>
> `pidstat -d` désigne `dd` (voir slide « pidstat »).

```
$ cat /proc/pressure/io
```

Questions :

4. Que valent `some` et `full` dans `/proc/pressure/io` ?

> **Réponses**
>
> `some` et `full` sont élevés.

Étape 5 : La mémoire

```
$ free -h
grep -E 'MemTotal|MemAvailable|^Cached' /proc/meminfo
```

Questions :

1. Quelle colonne de `free` indique la mémoire réellement allouable ?

> **Réponses**
>
> `available` (voir slide « mpstat (par CPU) et free (mémoire) »).

```
$ sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
free -h
vmstat -s | head -8
```

Questions :

2. La colonne `buff/cache` est-elle de la mémoire perdue ? Que montre-t-on en la vidant ?

> **Réponses**
>
> Non : le noyau rend le cache dès qu'un programme a besoin de mémoire. Après `drop_caches`, `buff/cache` baisse et `free` monte, mais `available` varie peu : le cache en faisait déjà partie.

Étape 6 : Enregistrer et rejouer

Vérifier la collecte, puis enregistrer soi-même :

```
$ rpm -q sysstat
sudo dnf install -y sysstat
sudo systemctl enable --now sysstat
systemctl list-timers 'sysstat*' --no-pager
ls /var/log/sa
```

Produire de la charge pendant un enregistrement :

```
$ sha256sum /dev/zero > /dev/null &
sar -o /var/tmp/tp.sa 1 5 > /dev/null
kill %1
sar -u -f /var/tmp/tp.sa
sar -q -f /var/tmp/tp.sa
```

Questions :

1. Quel mécanisme déclenche la collecte de `sysstat` ? Où sont les fichiers `saXX` ?
2. Que gagne-t-on à enregistrer plutôt qu'à observer en direct ?

> **Réponses**
>
> Des *timers* systemd (`sysstat-collect.timer`, `sysstat-summary.timer`), et non plus cron. Les fichiers sont dans `/var/log/sa/` (voir slide « sysstat : enregistrer pour rejouer »). On peut analyser un incident passé, comme une charge de nuit : `sar -f` rejoue le fichier avec ses horaires.

Étape 7 : Exporter avec `sadf`

```
$ sadf -d /var/tmp/tp.sa -- -u | head -3
sadf -j /var/tmp/tp.sa -- -u | head -6
```

Questions :

1. Quels formats d'export `sadf` propose-t-il ?

> **Réponses**
>
> CSV avec `-d`, JSON avec `-j`, SVG avec `-g` : la passerelle vers un outil de graphe (voir slide « sar : exploiter l'historique »).

```
$ sar -q PSI 1 3
sar -n DEV 1 2
rm /var/tmp/tp.sa
```

Questions :

2. Quelle option de `sar` affiche la PSI ? Que faut-il pour qu'elle fonctionne ?

> **Réponses**
>
> `sar -q PSI`, à condition que la PSI soit activée (`psi=1`, étape 1).

> **Nettoyer**
>
> Retirer `psi=1` si la PSI n'est plus utile : `sudo grubby --update-kernel=ALL --remove-args=psi=1`, puis redémarrer. Le TP d'optimisation qui suit peut s'en servir.

Module 15

Optimiser les performances

TP : mesurer, régler le disque, les processus et le réseau

Objectif et machine

Appliquer la méthode « **mesurer, régler, re-mesurer** » : tester un disque, changer le scheduler d'I/O, régler un montage, appliquer un profil `tuned`, hiérarchiser des processus, lire le démarrage, tester le réseau.

La machine

La VM du TP précédent (AlmaLinux 10, 2 CPU, accès à Internet, compte avec `sudo`), en SSH. Les outils de ce TP viennent d'AppStream, de BaseOS et d'EPEL :

```
$ sudo dnf install -y epel-release
sudo dnf install -y fio ioping ps_mem iperf3 mtr time tuned
```

> ⚠ Un seul réglage à la fois
>
> Chaque réglage est noté avec sa valeur d'origine et remis à l'état initial à l'étape 12.

Étape 1 : Identifier le support

```
$ lsblk -d -o NAME,ROTA,DISC-GRAN,MODEL
```

Questions :

1. Que signifie la colonne `ROTA` ? Quelle valeur la VM annonce-t-elle ?

> ℹ Réponses
>
> `ROTA=1` : disque rotatif, `0` : SSD ou NVMe (voir slide « Tester les performances du disque »). La VM annonce `1` pour ses disques virtuels : cette valeur ne dit rien du support réel.

```
$ for d in vda vdb vdc; do echo $d $(cat /sys/block/$d/queue/scheduler); \
  done
```

Questions :

2. Quel scheduler est actif sur chaque disque, et que signifient les crochets ?

> ℹ Réponses
>
> Le scheduler actif est entre crochets, ici `none` : un disque virtuel laisse l'ordonnancement à l'hyperviseur (voir slide « Choisir le scheduler d'I/O (blk-mq) »).

Étape 2 : Mesurer avec `fio` et `ioping`

Un fichier de test dans `/var/tmp`, pas un périphérique :

```
$ df -h /var/tmp
fio --name=test --filename=/var/tmp/fio-test --size=256M --direct=1 \
    --rw=randrw --bs=4k --numjobs=2 --runtime=10 --time_based \
  --group_reporting
rm /var/tmp/fio-test
```

Questions :

1. Quels débits et IOPS `fio` annonce-t-il en lecture et en écriture ?

> **Réponses**
>
> Plusieurs dizaines de milliers d'IOPS et plus de 100 Mo/s dans la VM de test : les accès passent par le cache de l'hôte, ces chiffres sont sans rapport avec un disque physique.

```
$ ioping -c 5 /var/tmp
```

Questions :

2. Quelle latence `ioping` mesure-t-il (`min/avg/max`) ?
3. Pourquoi ne pas lancer `fio` sur `/dev/vda` ?

> **Réponses**
>
> `ioping` mesure la latence par requête, de l'ordre de la milliseconde ou moins (voir slide « Tester les performances du disque »). Écrire sur le périphérique détruirait ce qu'il contient : le test se fait sur un fichier (voir slide « Bonnes pratiques »).

Étape 3 : Changer le scheduler d'I/O

Sur `vdb`, un disque sans données *(disque supplémentaire du TP « Stockage », laissé vierge, rattaché à la VM à la fin du TP « Booter autrement »)* :

```
$ echo mq-deadline | sudo tee /sys/block/vdb/queue/scheduler
cat /sys/block/vdb/queue/scheduler
```

Rendre le réglage **durable** avec une règle `udev`. Créer `/etc/udev/rules.d/60-tp-scheduler.rules` avec un éditeur lancé en root *(`sudo`)* :

```
≡ ACTION=="add|change", KERNEL=="vdb", ATTR{queue/scheduler}="none"
```

Puis la tester :

```
$ sudo udevadm control --reload
sudo udevadm trigger --subsystem-match=block --action=change
cat /sys/block/vdb/queue/scheduler
```

Questions :

1. Le changement par `echo` survit-il à un redémarrage ?
2. Que fait la règle `udev` ?

> **Réponses**
>
> Non : il est volatile. La règle `udev` repose sur le disque (`KERNEL=="vdb"`) et pose `none` à chaque détection du disque : après `trigger`, `none` redevient actif (voir slide « Choisir le scheduler d'I/O (blk-mq) »). Un profil `tuned` est l'autre voie.

Étape 4 : Options de montage et TRIM

```
$ findmnt -no OPTIONS /var
sudo mount -o remount,noatime /var
findmnt -no OPTIONS /var
sudo mount -o remount,relatime /var
```

Questions :

1. Quelle option de date d'accès `/var` utilise-t-il au départ ?

> **Réponses**
>
> `relatime` : la date d'accès n'est mise à jour que de temps en temps. `noatime` ne l'écrit jamais, ce qui limite les écritures (voir slide « Régler le filesystem »).

```
$ systemctl list-timers fstrim.timer --no-pager
sudo fstrim -v /var
```

Questions :

2. Quel mécanisme lance `fstrim` périodiquement ?

> **Réponses**
>
> Le *timer* `fstrim.timer` lance `fstrim`, préférable au `discard` continu. Pour rendre `noatime` durable, l'ajouter dans `/etc/fstab`.

Étape 5 : Un profil `tuned`

```
$ sudo systemctl enable --now tuned
tuned-adm active
tuned-adm recommend
```

Questions :

1. Quel profil est actif, et lequel `tuned` recommande-t-il ?

> **Réponses**
>
> `virtual-guest`, aussi le profil recommandé pour une VM.

```
$ sysctl vm.swappiness vm.dirty_ratio
sudo tuned-adm profile throughput-performance
sysctl vm.swappiness vm.dirty_ratio
```

Questions :

2. Quelles valeurs de `vm.swappiness` et `vm.dirty_ratio` avant et après le changement de profil ?

> **Réponses**
>
> Avec `virtual-guest` : `vm.swappiness=30` et `vm.dirty_ratio=30`. Avec `throughput-performance` : `10` et `40`. Un profil règle d'un coup les paramètres `vm`, le scheduler et le CPU (voir slide « Régler le filesystem »). Les valeurs du noyau sans `tuned` sont `60` et `20`.

Étape 6 : Où est la mémoire ?

```
$ free -h
sudo ps_mem | tail -6
```

Questions :

1. Quel programme occupe le plus de mémoire selon `ps_mem` ? Quelle grandeur additionne-t-il ?

> **Réponses**
>
> `ps_mem` additionne la mémoire privée et une part de la mémoire partagée, par programme (le dernier de la liste est le plus gros).

```
$ sudo slabtop -o | sed -n 7,11p
```

Questions :

2. Que contient `slabtop` ?

> **Réponses**
>
> `slabtop` liste les structures du noyau en mémoire : `buffer_head`, `dentry`, `inode_cache`… (voir slide « Analyse détaillée de la mémoire »).

> **À savoir**
>
> `smem`, cité dans d'anciennes documentations, n'existe pas dans EPEL 10 : `ps_mem` en tient lieu.

Étape 7 : Hiérarchiser avec `nice`

Trois calculs lancés ensemble, puis un renice :

```
$ sha256sum /dev/zero > /dev/null & P1=$!
sha256sum /dev/zero > /dev/null & P2=$!
sha256sum /dev/zero > /dev/null & P3=$!
sleep 6; ps -o pid,ni,pcpu,comm -p $P1,$P2,$P3
renice -n 19 -p $P3
sleep 6; ps -o pid,ni,pcpu,comm -p $P1,$P2,$P3
kill $P1 $P2 $P3
```

Questions :

1. Quelle part de CPU chaque calcul reçoit-il avant et après `renice` ?
2. Un utilisateur peut-il remettre `P3` à une priorité plus élevée ?

> **Réponses**
>
> Environ 66 % chacun avant (3 calculs pour 2 CPU), puis 83 % pour les deux premiers et 33 % pour `P3` : `nice 19` lui laisse ce que les autres n'utilisent pas (voir slide « Identifier les processus inutiles et consommateurs »). Non : `renice` vers une valeur plus basse demande root (« Permission denied »).

Étape 8 : Mesurer une commande

```
$ /usr/bin/time -v tar -czf /dev/null /usr/lib64 2>&1 | \
    grep -E 'Elapsed|Maximum resident|Percent'
  nice -n 19 /usr/bin/time -v tar -czf /dev/null /usr/lib64 2>&1 | \
    grep -E 'Elapsed|Maximum resident'
```

Questions :

1. Quels sont le temps réel et le pic de mémoire (`Maximum resident set size`) de la commande ?
2. Qu'ajoute `/usr/bin/time` à la commande `time` du shell ?

> **Réponses**
>
> Les valeurs s'affichent dans la sortie (`Elapsed` en secondes, le pic en kilo-octets). `/usr/bin/time -v` détaille le pic de mémoire (RSS max) (voir slide « Identifier les processus inutiles et consommateurs »). Sans autre charge, `nice` ne change presque rien : il ne départage que des processus en concurrence.

Étape 9 : Analyser le démarrage

```
$ systemd-analyze
```

Questions :

1. Quels temps `systemd-analyze` donne-t-il pour le noyau, l'initrd et l'espace utilisateur ?

> **Réponses**
>
> Les trois durées s'additionnent sur la première ligne (voir slide « Booter rapidement son système »).

```
$ systemd-analyze blame | head -6
  systemd-analyze critical-chain | head -12
```

Questions :

2. Le service le plus long de `blame` est-il sur la chaîne critique ?

> **Réponses**
>
> Un service long dans `blame` ne gêne que s'il est sur la chaîne critique : `critical-chain` désigne ce qui bloque réellement (`multi-user.target` et ses dépendances).

> **À savoir**
>
> Avec `blame | grep -v '.device'`, les unités de périphériques (virtio, qemu-guest-agent…) ne masquent plus les services.

Étape 10 : Tester le réseau

```
$ ping -c 3 127.0.0.1
  ss -s
  iperf3 -s -D
  iperf3 -c 127.0.0.1 -t 3
  pkill iperf3
```

Entre **deux VM**, remplacer `127.0.0.1` par l'adresse de l'autre VM, `iperf3 -s` y étant lancé.

Questions :

1. Quel débit `iperf3` mesure-t-il en bouclage ? Que vaut-il comme mesure du réseau ?
2. Quelle commande donne le nombre de connexions TCP ?

> **Réponses**
>
> Plusieurs dizaines de Gbit/s : le bouclage ne traverse aucune carte réseau, il mesure la machine et non le réseau. Il faut deux hôtes pour mesurer un lien (voir slide « Tester les performances du réseau »). `ss -s` résume les sockets, dont `TCP:`.

Étape 11 : L'algorithme TCP

```
$ sysctl net.ipv4.tcp_congestion_control \
    net.ipv4.tcp_available_congestion_control
sudo modprobe tcp_bbr
sysctl net.ipv4.tcp_available_congestion_control
```

Questions :

1. Quel algorithme est actif au départ ? Lesquels sont disponibles avant et après `modprobe` ?

> **Réponses**
>
> `cubic`, avec `reno` et `cubic` disponibles. `bbr` apparaît une fois le module `tcp_bbr` chargé.

```
$ sudo sysctl -w net.ipv4.tcp_congestion_control=bbr
sysctl net.ipv4.tcp_congestion_control
```

Questions :

2. Que faut-il pour garder `bbr` après un redémarrage ?

> **Réponses**
>
> Pour le garder, un fichier de `/etc/sysctl.d/` (et le module chargé au démarrage, avec `net.core.default_qdisc = fq` conseillé) (voir slide « MTU, fenêtres et files d'émission »).

Étape 12 : Remettre à l'état initial

```
$ sudo tuned-adm profile virtual-guest
sudo sysctl -w net.ipv4.tcp_congestion_control=cubic
sudo rm -f /etc/udev/rules.d/60-tp-scheduler.rules
sudo udevadm control --reload
```

Questions :

1. Quelle règle de méthode le TP a-t-il appliquée à chaque réglage ?
2. Quel réglage a eu un effet **mesuré** dans ce TP ?

> **Réponses**
>
> Une mesure avant, un seul changement, une mesure après (voir slide « Mesurer avant d'optimiser »). Le `renice`, visible dans la part de CPU de chaque calcul. Le scheduler, les options de montage et `bbr` n'ont pas corrigé de mesure sur cette VM : ils ne se gardent pas (voir slide « Bonnes pratiques »).

Module 16

Collecter et grapher des métriques système

TP : Cockpit et PCP, puis Prometheus, une alerte et Grafana

Objectif et machine

Observer une machine avec la solution **native** (Cockpit + PCP), puis monter la pile **Prometheus** : exporter, collecte, requêtes, alerte et tableau de bord Grafana.

La machine

Une VM AlmaLinux 10 avec accès à Internet, un compte qui dispose de `sudo` et **au moins 1 Go libre** sur `/` (Grafana en demande environ 650 Mo). Les pages web s'ouvrent depuis le navigateur du poste de travail, avec l'adresse de la VM.

```
$ ssh admin@vm3-ks
ip -4 -br addr show
df -h /
```

> ⚠ Une seule machine pour toute la pile
>
> Cockpit et Prometheus écoutent tous deux sur le port `9090` : l'étape 3 déplace Prometheus sur `9091`.

Étape 1 : Cockpit et PCP

```
$ sudo dnf install -y cockpit pcp-zeroconf
sudo systemctl enable --now pmcd pmlogger cockpit.socket
systemctl is-active pmcd pmlogger cockpit.socket
pminfo -f kernel.all.load
pmrep -s 3 kernel.all.load mem.util.available
```

Dans le navigateur : `https://<adresse de la VM>:9090`, connexion avec le compte `admin`, page **Metrics**.

Questions :

1. Quelles valeurs `pminfo` donne-t-il pour la charge, et sur quelles durées ?
2. Quels services collectent l'historique que la page **Metrics** affiche ?
3. Quelle page de Cockpit permet de relier une courbe à un service ou à un journal ?

> ℹ Réponses
>
> Les moyennes de charge à 1, 5 et 15 minutes. `pmcd` collecte et `pmlogger` archive : la page **Metrics** relit cet historique (voir slide « Cockpit + PCP : le tableau de bord intégré »). Les pages **Services** et **Logs** (même slide).

> ℹ À savoir
>
> Le paquet `cockpit-pcp` n'existe plus : la page **Metrics** est fournie par `cockpit-system`.

Étape 2 : L'exporter

```
$ sudo dnf install -y epel-release
sudo dnf install -y node-exporter
sudo systemctl enable --now node_exporter
sudo ss -tlnp | grep 9100
```

Questions :

1. Quel port `node_exporter` écoute-t-il, et quel chemin expose les métriques ?
2. D'où vient le paquet `node-exporter` ?

> **Réponses**
>
> Le port `9100`, chemin `/metrics` (voir slide « Déployer `node_exporter` sur la cible »). D'**EPEL 10**, pas d'AlmaLinux (voir slide « Architecture Prometheus »).

```
$ curl -s localhost:9100/metrics | grep -E '^node_memory_MemAvailable'
curl -s localhost:9100/metrics | grep -E '^node_cpu.*cpu="0",mode="idle"'
curl -s localhost:9100/metrics | wc -l
```

Questions :

3. Pourquoi `node_cpu_seconds_total` croît-il sans cesse ?

> **Réponses**
>
> C'est un compteur cumulatif : on en dérive un taux avec `rate()` (voir slide « PromQL : du compteur à l'indicateur »).

Étape 3 : Prometheus

```
$ sudo dnf install -y prometheus
```

Remplacer le contenu de `/etc/prometheus/prometheus.yml` avec un éditeur lancé en root (*`sudo`*) par :

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "noeuds"
    static_configs:
      - targets:
          - "localhost:9100"
```

Puis contrôler la syntaxe :

```
$ sudo promtool check config /etc/prometheus/prometheus.yml
```

Étape 4 : Port, rétention et démarrage

Éditer `/etc/default/prometheus` avec un éditeur lancé en root (*`sudo`*) et remplacer la ligne `ARGS=''` par :

```
ARGS='--web.listen-address=0.0.0.0:9091 --storage.tsdb.retention.time=30d'
```

```
$ grep ^ARGS /etc/default/prometheus
sudo systemctl enable --now prometheus
curl -s localhost:9091/-/ready
curl -s localhost:9091/api/v1/targets | python3 -m json.tool | \
  grep -E 'health|scrapeUrl'
```

Questions :

1. Que contrôle `promtool check config` ?
2. Pourquoi `--web.listen-address` ? Que se passerait-il sans lui ?
3. Quel état la cible `noeuds` affiche-t-elle ?

> **Réponses**
>
> La syntaxe du fichier, sans démarrer le service. Cockpit occupe déjà `9090` : sans ce paramètre, Prometheus ne peut pas écouter (voir slide « Configurer la collecte »). La cible est `up` : Prometheus a scruté `localhost:9100` (la page `http://<VM>:9091/targets` donne la même information).

Étape 5 : Interroger avec PromQL

Dans le navigateur (`http://<VM>:9091`, onglet **Graph**) ou avec `curl` :

```
$ q() { curl -s -G localhost:9091/api/v1/query --data-urlencode "query=$1" |
  python3 -c 'import sys,json;print(json.load(sys.stdin)["data"]["result"][0]["value"]
[1])'; }
q 'node_memory_MemAvailable_bytes'
q '100 * node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes'
q '100 * (1 - avg by (instance)(rate(node_cpu_seconds_total{mode="idle"}[5m])))'
q 'node_filesystem_avail_bytes{mountpoint="/"}
   / node_filesystem_size_bytes{mountpoint="/"}'
```

Pour le CPU, `sha256sum /dev/zero > /dev/null &` produit de la charge (`kill %1` l'arrête). Ouvrir l'interface demande peut-être `sudo firewall-cmd --add-port=9091/tcp`.

Questions :

1. Quel pourcentage de mémoire est disponible ? Quel pourcentage de CPU est utilisé (attendre 5 minutes de données, ou réduire la fenêtre à `[1m]`) ?
2. Quelle fraction du système de fichiers `/` reste libre ?
3. Que fait `rate(...[5m])`, et `avg by (instance)` ?

> **Réponses**
>
> Les valeurs se lisent dans les sorties. `rate()` calcule la pente d'un compteur sur la fenêtre, `avg by (label)` agrège en gardant une dimension : c'est l'équivalent dynamique des graphes figés de Cacti (voir slide « PromQL : du compteur à l'indicateur »).

Étape 6 : Une règle d'alerte

Un seuil volontairement haut pour déclencher l'alerte :

Créer `/etc/prometheus/rules.yml` avec un éditeur lancé en root (*`sudo`*) :

```
groups:
  - name: noeud
    rules:
      - alert: DisqueRacinePresquePlein
        expr: node_filesystem_avail_bytes{mountpoint="/"}
              / node_filesystem_size_bytes{mountpoint="/"} < 0.95
        for: 30s
        labels:
          severity: warning
        annotations:
          summary: "Moins de 95 % libres sur / ({{ $labels.instance }})"
```

Déclarer ce fichier dans `/etc/prometheus/prometheus.yml`, avec un éditeur lancé en root (*`sudo`*), avant `scrape_configs:` :

```
rule_files:
  - "rules.yml"
```

```
$ sudo promtool check config /etc/prometheus/prometheus.yml
sudo systemctl reload prometheus
```

Étape 7 : Observer l'alerte

Après une à deux minutes :

```
$ curl -s localhost:9091/api/v1/alerts | python3 -m json.tool | \
  grep -E 'alertname|state'
```

Questions :

1. Quels états l'alerte traverse-t-elle, et quel rôle joue `for: 30s` ?
2. Quel composant route ensuite l'alerte vers un mail ou Slack ?

> **Réponses**
>
> `pending` (la condition est vraie, la durée `for` n'est pas écoulée) puis `firing`. `for` évite les fausses alertes sur un pic transitoire (voir slide « Des règles d'alerte »). C'est **Alertmanager**, qui n'est pas installé dans ce TP : Prometheus affiche l'alerte, mais n'envoie rien.

> **Dans un cas réel**
>
> Le seuil serait `< 0.10` et `for: 10m`, comme dans le support.

Étape 8 : Grafana et le dashboard 1860

```
$ sudo dnf install -y grafana
sudo systemctl enable --now grafana-server
curl -s localhost:3000/api/health
sudo firewall-cmd --add-port=3000/tcp
```

Dans le navigateur, `http://<VM>:3000`, connexion `admin` / `admin` (changer le mot de passe) :

1. **Connections** puis **Add data source** : Prometheus, URL `http://localhost:9091`, **Save & test**.
2. **Dashboards** puis **New**, **Import**, ID **1860**, **Load**, choisir la source Prometheus, **Import** (le navigateur doit joindre `grafana.com`).

Questions :

1. Quel port Grafana utilise-t-il, et d'où vient son paquet ?

2. Quels panneaux le dashboard 1860 affiche-t-il sans qu'aucun ne soit construit ?

> **Réponses**
>
> Le port `3000`. Le paquet `grafana` est dans AppStream (version 10.2) (voir slides « Architecture Prometheus » et « Grafana et le dashboard 1860 »). CPU, mémoire, disque et réseau de la machine : le dashboard « Node Exporter Full » lit les métriques de `node_exporter`.

Étape 9 : Nettoyage et bilan

```
$ sudo systemctl disable --now grafana-server prometheus node_exporter
sudo systemctl disable --now cockpit.socket pmlogger pmcd
sudo dnf remove -y grafana prometheus node-exporter cockpit pcp-zeroconf
sudo rm -rf /var/lib/prometheus /var/lib/grafana
sudo rm -rf /etc/prometheus /etc/default/prometheus.rpmsave
```

Questions

1. Quelle différence de principe entre Cockpit + PCP et la pile Prometheus ?
2. Qu'est-ce qui distingue la collecte par *pull* de Prometheus de l'ancien poll SNMP de Cacti ?

> **Réponses**
>
> Cockpit + PCP suffit pour une machine ou un petit parc, sans serveur dédié. Prometheus est une base de séries temporelles centrale qui scrute plusieurs cibles (voir slide « Quelle stack pour quel besoin ? »). Avec Prometheus, les métriques portent des **labels** et se requêtent à la demande. Dans Cacti, chaque courbe était définie à l'avance dans un fichier RRD (voir slide « Métrique ? »).

Annexe 3

Gestion avancée et centralisation des logs

TP : rsyslog, logrotate, centralisation vers l'hôte (TCP, RELP, journal-remote)

Objectif et machines

Écrire les messages dans un fichier avec `rsyslog`, les faire **tourner** avec `logrotate`, puis **centraliser** les logs de la VM vers l'hôte KVM : en TCP, avec **RELP**, et avec `systemd-journal-upload`.

Les machines

La **VM** AlmaLinux 10 des TP précédents, en installation minimale, avec un compte qui dispose de `sudo`, en SSH. Les étapes 1 à 3 se font sur elle seule.

Pour les étapes 5 à 10, le **récepteur** est l'**hôte KVM** (le poste de travail AlmaLinux 9) : la VM émet vers l'adresse de l'hôte sur le réseau `default` de `libvirt`, `192.168.122.1`. Les commandes de l'hôte se tapent sur le poste de travail, celles de la VM en SSH.

```
$ ssh admin@vm3-ks
ip -4 -br addr show
```

> **SELinux et pare-feu**
>
> SELinux reste en mode `Enforcing` et le pare-feu actif, sur l'hôte comme sur la VM : ils font partie de l'exercice. Les VM atteignent l'hôte par la zone `libvirt` de `firewalld`.

Étape 1 : Installer `rsyslog`

```
$ rpm -q rsyslog
ls /var/log
sudo dnf install -y rsyslog
sudo systemctl enable --now rsyslog
logger -t TP 'message pour rsyslog'
ls /var/log
```

Questions :

1. Quels fichiers apparaissent dans `/var/log` après l'installation ?

> **Réponses**
>
> `messages`, `secure`, `maillog`, `cron` et `spooler`.

```
$ sudo tail -n 3 /var/log/messages
```

Questions :

2. Que contient `messages` ?

Réponses

`messages` contient les messages du système, dont `message pour rsyslog` et ceux du démarrage du service (voir slide « `rsyslog` : quand l'installer ? »).

À savoir

Sur une installation minimale, `rpm -q rsyslog` répond « package rsyslog is not installed » et `/var/log` ne contient presque rien (voir slide « `rsyslog` : quand l'installer ? »).

Étape 2 : Une règle personnelle

Créer `/etc/rsyslog.d/50-tp.conf` avec un éditeur lancé en root (*`sudo`*) :

```
local0.*  action(type="omfile" file="/var/log/tp.log")
```

Puis contrôler la syntaxe, redémarrer et tester :

```
$ sudo rsyslogd -N1
```

Questions :

1. Que contrôle `rsyslogd -N1` ?
2. Que signifie le sélecteur `local0.*` ?

Réponses

La syntaxe de la configuration, sans démarrer le service. `local0.*` : la facility `local0`, toutes les sévérités (voir slide « Règles : sélecteur ⇒ action »).

```
$ sudo systemctl restart rsyslog
logger -p local0.info -t TP 'test de la règle'
sudo tail -n 1 /var/log/tp.log
sudo grep -c 'test de la règle' /var/log/messages
```

Questions :

3. Que contient `tp.log` ? Le message figure-t-il aussi dans `messages` ?

Réponses

`tp.log` contient le message, qui figure aussi dans `messages` (`grep -c` affiche `1`) : une règle ajoutée n'en remplace pas une autre.

Étape 3 : Écrire la règle de rotation

Créer `/etc/logrotate.d/tp` avec un éditeur lancé en root (*`sudo`*) :

```
/var/log/tp.log {
    rotate 3
    compress
    delaycompress
    missingok
    notifempty
    create 0640 root root
    postrotate
        /usr/bin/systemctl reload rsyslog.service >/dev/null 2>&1 || true
    endscript
}
```

```
$ for i in $(seq 20); do logger -p local0.info -t TP "ligne $i"; done
sudo logrotate -d -f /etc/logrotate.d/tp
```

Questions :

1. Que décrit la simulation `-d` ?

> **Réponses**
>
> Ce que `logrotate` ferait (voir slide « `logrotate` : tester sa règle »).

```
$ systemctl list-timers logrotate.timer --no-pager
```

Questions :

2. Quel mécanisme lance `logrotate` chaque jour ?

> **Réponses**
>
> Le timer `logrotate.timer` (voir slide « `logrotate` : faire tourner les logs »).

Étape 4 : Forcer la rotation

```
$ sudo logrotate -f /etc/logrotate.d/tp
ls -l /var/log/tp.log*
logger -p local0.info -t TP 'après la rotation'
sudo logrotate -f /etc/logrotate.d/tp
ls -l /var/log/tp.log*
sudo grep tp.log /var/lib/logrotate/logrotate.status
```

Questions :

1. Quels fichiers existent après la première rotation ? Après la seconde ? Lesquels sont compressés ?
2. Que fait `delaycompress` ? Que contient le `tp.log` courant ?
3. Quel est le rôle du bloc `postrotate` ?

> **Réponses**
>
> Après la première : `tp.log` (vide) et `tp.log.1`. Après la seconde : `tp.log`, `tp.log.1` et `tp.log.2.gz`. `delaycompress` retarde la compression d'une rotation, d'où un `.1` non compressé. `create` recrée un `tp.log` vide. `postrotate` demande à `rsyslog` de rouvrir son fichier.

> **À savoir**
>
> Sans `-f`, `logrotate -d` répond « does not need rotating » : la fréquence par défaut de `/etc/logrotate.conf` s'applique.

Étape 5 : L'hôte écoute

Sur l'**hôte**, un récepteur TCP simple et un récepteur RELP dans `/etc/rsyslog.d/00-recepteur.conf` :

```
module(load="imtcp")
input(type="imtcp" port="514")
module(load="imrelp")
input(type="imrelp" port="2514" maxDataSize="10k")
```

```
$ sudo dnf install -y rsyslog rsyslog-relp policycoreutils-python-utils
sudo systemctl enable --now rsyslog
sudo rsyslogd -N1
sudo systemctl restart rsyslog
journalctl -u rsyslog -n 4 --no-pager
ss -tln | grep -E ':514|:2514'
```

Questions :

1. Quel récepteur ne démarre pas, et que dit le journal de `rsyslog` ?
2. Quelle commande montre le refus de SELinux ?

> **Réponses**
>
> RELP : « error while binding relp tcp socket on port 2514 ». `sudo ausearch -m avc -ts recent` montre un `name_bind` refusé : le port `2514` n'est pas un port syslog pour SELinux (voir slide « RELP : SELinux et pare-feu »). Le port `514/tcp` est autorisé d'origine.

Étape 6 : Autoriser, ouvrir, vérifier

Suite sur l'**hôte** :

```
$ sudo semanage port -a -t syslogd_port_t -p tcp 2514
```

Questions :

1. Que fait `semanage port -a` ?

> **Réponses**
>
> Il associe le port `2514/tcp` au type `syslogd_port_t`.

```
$ firewall-cmd --get-zone-of-interface=virbr0
sudo firewall-cmd --zone=libvirt --add-port=514/tcp --add-port=2514/tcp
```

Questions :

2. Dans quelle zone se trouve le pont des VM, et que reste-t-il à faire pour que l'ouverture survive à un redémarrage ?

> **Réponses**
>
> Le pont `virbr0` est dans la zone `libvirt` : l'ouverture se fait dans cette zone, et l'option `--permanent` (puis `firewall-cmd --reload`) la rend durable (voir slide « RELP : SELinux et pare-feu »).

```
$ sudo systemctl restart rsyslog
ss -tln | grep -E ':514|:2514'
```

Questions :

3. Quels ports écoutent maintenant ?

> **Réponses**
>
> Les ports `514` et `2514`.

Étape 7 : La VM envoie

Sur la **VM**, deux envois selon le tag, dans `/etc/rsyslog.d/90-vers-hote.conf` (adresse de l'hôte :

`192.168.122.1`) :

```
module(load="omrelp")
if $programname == 'TCPTEST' then {
    action(type="omfwd" target="192.168.122.1" port="514" protocol="tcp")
}
if $programname == 'RELPTEST' then {
    action(type="omrelp" target="192.168.122.1" port="2514"
           queue.type="LinkedList" queue.filename="relp_queue"
           queue.spoolDirectory="/var/spool/rsyslog" queue.saveOnShutdown="on"
           action.resumeRetryCount="-1" action.resumeInterval="5")
}
```

Étape 8 : Envoyer des messages

Sur la **VM** :

```
$ sudo dnf install -y rsyslog rsyslog-relp
sudo systemctl enable --now rsyslog
sudo rsyslogd -N1
sudo systemctl restart rsyslog
logger -t TCPTEST 'tcp : récepteur actif'
logger -t RELPTEST 'relp : récepteur actif'
```

Sur l'**hôte** : `sudo grep -E 'TCPTEST|RELPTEST' /var/log/messages`.

Questions :

1. Quel module envoie en TCP, lequel envoie en RELP ?
2. Pourquoi filtrer sur `$programname` ?

> **Réponses**
>
> `omfwd` pour TCP, `omrelp` pour RELP (voir slides « Centraliser : envoyer vers un serveur » et « Centralisation fiable avec RELP »). Chaque tag prend un seul chemin : les deux envois se comparent sans se mélanger.

Étape 9 : Le récepteur s'arrête

L'hôte ne s'éteint pas : on arrête seulement son service `rsyslog` le temps d'émettre depuis la VM.

```
$ sudo systemctl stop rsyslog                  # hôte
logger -t TCPTEST 'tcp : récepteur arrêté'     # VM
logger -t RELPTEST 'relp : récepteur arrêté'   # VM
sudo systemctl start rsyslog                   # hôte
```

Attendre une trentaine de secondes, puis sur l'**hôte** :

```
$ sudo grep -E 'récepteur arrêté' /var/log/messages
```

Questions :

1. Quels messages l'hôte a-t-il reçus ?
2. Qu'est-ce qui explique la différence ?

> **Réponses**
>
> Seul le message `relp : récepteur arrêté` arrive. Le client RELP l'a gardé dans sa file d'attente (`queue.*`) et l'a renvoyé quand le récepteur est revenu, avec `action.resumeRetryCount="-1"`. Le message TCP, envoyé pendant l'arrêt, est perdu (voir slides « Formats et transports de syslog » et « Centralisation fiable avec RELP »).

Étape 10 : L'hôte reçoit le journal *(facultatif)*

Sur l'**hôte**, recevoir en HTTP dans `/etc/systemd/system/systemd-journal-remote.service.d/override.conf` :

```
[Service]
ExecStart=
ExecStart=/usr/lib/systemd/systemd-journal-remote --listen-http=-3 --
output=/var/log/journal/remote/
```

```
$ sudo dnf install -y systemd-journal-remote
sudo systemctl daemon-reload
sudo systemctl enable --now systemd-journal-remote.socket
sudo firewall-cmd --zone=libvirt --add-port=19532/tcp
```

Étape 11 : La VM envoie son journal

Sur la **VM**, dans `/etc/systemd/journal-upload.conf` :

```
[Upload]
URL=http://192.168.122.1:19532
```

```
$ sudo dnf install -y systemd-journal-remote
sudo systemctl enable --now systemd-journal-upload.service
logger -t UPTEST 'message via journal-upload'
```

Sur l'**hôte** :

```
$ sudo journalctl --directory /var/log/journal/remote -t UPTEST
```

Questions :

1. Quel fichier l'hôte crée-t-il dans `/var/log/journal/remote/` ?
2. Quelle différence avec RELP : que transporte `journal-upload` ?

> **Réponses**
>
> Un fichier `remote-<adresse de l'émetteur>.journal` (voir slide « Centraliser avec journal-upload et journal-remote »). Il transporte le **journal** lui-même, avec tous ses champs, et non des lignes de texte syslog.

Étape 12 : Nettoyage et bilan

Sur la **VM** :

```
$ sudo rm -f /etc/rsyslog.d/50-tp.conf /etc/rsyslog.d/90-vers-hote.conf
sudo rm -f /etc/logrotate.d/tp /var/log/tp.log*
sudo systemctl disable --now systemd-journal-upload.service
sudo systemctl restart rsyslog
```

Sur l'**hôte** :

```
$ sudo rm -f /etc/rsyslog.d/00-recepteur.conf
sudo rm -rf /etc/systemd/system/systemd-journal-remote.service.d
sudo systemctl disable --now systemd-journal-remote.socket
sudo semanage port -d -t syslogd_port_t -p tcp 2514
sudo firewall-cmd --zone=libvirt --remove-port=514/tcp \
  --remove-port=2514/tcp --remove-port=19532/tcp
sudo systemctl restart rsyslog
```

`rsyslog` reste installé : ce réglage peut rester en place.

Étape 13 : Bilan

Questions :

1. Quelle différence entre `journald` et `rsyslog` ? Dans quel cas installer `rsyslog` ?
2. Pourquoi RELP et une file d'attente plutôt que du TCP seul ?
3. Pourquoi simuler avec `logrotate -d` avant de forcer ?

> **Réponses**
>
> `journald` collecte et stocke au format binaire, consulté par `journalctl`. `rsyslog` écrit des fichiers texte, filtre et envoie vers un serveur (voir slide « `rsyslog` : quand l'installer ? »). RELP accuse réception de chaque message et la file d'attente garde ce qui n'est pas acquitté. La simulation montre ce qui se passerait sans toucher aux fichiers (voir slide « `logrotate` : tester sa règle »).

Annexe 7

Intégrité du système

TP : rpm -V, AIDE et auditd

Objectif et machine

Contrôler l'intégrité d'une machine avec `rpm -V`, **provoquer** des modifications, les retrouver avec AIDE, puis identifier leur auteur avec `auditd`.

La machine

La **VM** AlmaLinux 10 des TP précédents, avec un compte qui dispose de `sudo`, en SSH. Aucune autre machine n'est nécessaire.

```
$ ssh admin@vm1-ks
sudo -v
```

> ⚠ La VM est modifiée, puis remise en état
>
> Le TP modifie `/etc/ssh/sshd_config` et `/etc/hosts`. L'étape 7 les remet en état. Une sauvegarde de `sshd_config` est faite dès l'étape 2 : sans elle, l'étape 7 ne peut pas restaurer le fichier.

Étape 1 : `rpm -V` sur un système sain

```
$ rpm -V openssh-server
sudo rpm -V openssh-server
```

Questions :

1. Que change l'ajout de `sudo` dans le résultat de `rpm -V openssh-server` ?

> ℹ Réponses
>
> Sans `sudo`, des `?` remplacent les sommes de contrôle et des lignes `missing ... (Permission denied)` apparaissent : le contrôle est inexploitable. Avec `sudo` et sans écart, rien ne s'affiche.

```
$ sudo rpm -Va > ~/rpm-reference.txt
wc -l ~/rpm-reference.txt
cat ~/rpm-reference.txt
```

Questions :

2. Que contient `rpm-reference.txt` sur une machine saine ?

> ℹ Réponses
>
> `rpm-reference.txt` contient quelques lignes sur une machine saine (fichiers de configuration modifiés à l'installation) : c'est la **référence** à laquelle comparer ensuite (voir slide « Exemple : une ligne ajoutée, des droits modifiés »).

Étape 2 : Modifier un fichier de configuration

```
$ sudo cp -a /etc/ssh/sshd_config /root/sshd_config.orig
```

Ajouter la ligne `# test integrite` à la fin de `/etc/ssh/sshd_config`, avec un éditeur lancé en root (*`sudo`*). Puis :

```
$ sudo rpm -V openssh-server
```

Questions :

1. Quelles lettres apparaissent après l'ajout de la ligne ?

> **Réponses**
>
> `S.5....T.  c /etc/ssh/sshd_config` : taille (`S`), somme de contrôle (`5`) et date (`T`), pour un fichier de configuration (`c`) (voir slide « Lire la sortie de `rpm -V` »).

```
$ sudo chmod 644 /etc/ssh/sshd_config
sudo rpm -V openssh-server
```

Questions :

2. Que change `chmod 644` dans la sortie ?

> **Réponses**
>
> La lettre `M` (droits) s'ajoute : `SM5....T.`.

Les modifications restent en place pour l'étape 4.

Étape 3 : Installer AIDE et créer la référence

```
$ sudo dnf install -y aide
sudo grep -v '^#' /etc/aide.conf | grep -v '^$' | head -40
```

Questions :

1. Quels chemins la configuration par défaut surveille-t-elle ?

> **Réponses**
>
> `/boot`, `/bin`, `/sbin`, `/lib`, `/lib64`, `/opt`, `/usr`, `/root` et une liste de fichiers de `/etc`.

```
$ sudo aide --init
sudo mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

Questions :

2. Sous quel nom la base de référence doit-elle se trouver pour le contrôle ?

> **Réponses**
>
> La base produite par `--init` s'appelle `aide.db.new.gz` : on la renomme en `aide.db.gz` (voir slide « Initialiser, contrôler, mettre à jour »).

> **À savoir**
>
> L'initialisation lit tout le système : une quinzaine de secondes sur une VM de test.

> **Attention : la base contient les modifications de l'étape 2**
>
> La ligne ajoutée à `sshd_config` fait déjà partie de la référence. Elle ne sera pas signalée à l'étape 4, qui crée de **nouvelles** modifications.

Étape 4 : Provoquer des écarts, puis contrôler

```
$ sudo aide --check; echo "code de retour : $?"
```

Questions :

1. Le premier contrôle (avant les modifications) est-il sans écart ?

> **Réponses**
>
> Le premier contrôle peut déjà signaler des fichiers qui changent seuls (`/etc/resolv.conf`, `/etc/hostname`) : c'est le « bruit normal » (voir slide « Protéger la base »).

Provoquer des écarts. Ajouter la ligne `# second test` à la fin de `/etc/ssh/sshd_config`, avec un éditeur lancé en root (*`sudo`*). Puis :

```
$ sudo touch /etc/tp-nouveau.conf
  sudo chmod 600 /etc/hosts
  sudo aide --check | head -40; echo "code de retour : ${PIPESTATUS[0]}"
```

Questions :

2. Quels fichiers le second contrôle signale-t-il, et avec quelles marques ?
3. Quel est le code de retour du second contrôle ?

> **Réponses**
>
> Le second contrôle ajoute `/etc/tp-nouveau.conf` (`f+++...`), `/etc/hosts` (`p`, droits, et `A`, ACL) et `/etc/ssh/sshd_config` (`H`, empreinte). Le code de retour est `5` : ajout (1) plus modification (4) (voir slide « Lire le rapport d'AIDE »).

> **À savoir**
>
> Le contrôle compare avec la base de l'étape 3 : une lecture du rapport complet (sans `head`) montre l'ancienne et la nouvelle valeur de chaque attribut.

Étape 5 : Retrouver l'auteur avec `auditd`

```
$ sudo systemctl is-active auditd
  sudo auditctl -w /etc/ssh/sshd_config -p wa -k sshd_conf
  sudo auditctl -l
```

Ajouter la ligne `# modification surveillee` à la fin de `/etc/ssh/sshd_config`, avec un éditeur lancé en root (*`sudo`*). Puis :

```
$ sudo ausearch -k sshd_conf -i | tail -12
  sudo aureport -k
```

Questions :

1. Quelles valeurs prennent `auid` et `uid` dans l'événement `SYSCALL` de la modification ?
2. Quelle commande a écrit dans le fichier ?

> **Réponses**
>
> `auid` est le compte de connexion (par exemple `admin`), `uid` vaut `root` à cause de `sudo`. La commande est l'éditeur utilisé (champ `exe`, ou la colonne `exe` de `aureport -k`) (voir slide « Qui est l'auteur : `auid` »).

> **À savoir**
>
> `auditctl` affiche `Old style watch rules are slower` : message d'information sur la forme `-w`, sans conséquence ici.

Étape 6 : Rendre la règle permanente (facultatif)

Créer `/etc/audit/rules.d/50-tp.rules` avec un éditeur lancé en root (*`sudo`*) :

```
-w /etc/ssh/sshd_config -p wa -k sshd_conf
```

Puis la charger :

```
$ sudo augenrules --load
sudo auditctl -l
```

Questions :

1. Que se passe-t-il pour la règle de l'étape 5 si le fichier `50-tp.rules` n'existe pas et que la machine redémarre ?

> **Réponse**
>
> La règle posée par `auditctl` disparaît au redémarrage : seule la règle écrite dans `/etc/audit/rules.d/` est rechargée (voir slide « Qui est l'auteur : `auid` »).

> **À savoir**
>
> `augenrules --load` affiche plusieurs fois l'état du sous-système d'audit (`enabled 1`, `backlog`…) : sans message d'erreur, la règle apparaît avec `auditctl -l`.

Étape 7 : Remettre la machine en état

```
$ sudo cp -a /root/sshd_config.orig /etc/ssh/sshd_config
sudo restorecon -v /etc/ssh/sshd_config
sudo chmod 644 /etc/hosts
sudo rm -f /etc/tp-nouveau.conf /etc/audit/rules.d/50-tp.rules
sudo auditctl -D
sudo augenrules --load
sudo dnf remove -y aide
sudo rm -rf /var/lib/aide /var/log/aide
sudo rpm -V openssh-server
sudo rpm -Va > ~/rpm-apres.txt
diff ~/rpm-reference.txt ~/rpm-apres.txt && \
  echo "identique à la référence"
```

Questions :

1. Que doit afficher `sudo rpm -V openssh-server` à la fin ?

Réponse

Rien : `cp -a` rétablit contenu, droits et date du fichier d'origine, et `restorecon` son contexte SELinux. La comparaison `diff` avec la référence de l'étape 1 est sans écart (voir slide « Un système sain n'est pas toujours muet »).

À savoir

Pour une machine réelle, on **garderait** AIDE, sa base et la règle `auditd` : la remise en état ne sert qu'à rendre la VM propre pour le TP suivant.