

Linux - Administration avancée



François MICAUX - ACTILIS

Tour de table...

Rôle, technos, où, depuis quand ?

Pour ce qui nous rassemble : débutant/initié ? attentes/objectifs ?

Contenu de la formation

- installation avec stockage spécifique
- installation automatisée avec Kickstart
- maintenir le système à jour
- maîtriser la compilation de logiciels
- produire des packages et mettre en oeuvre un dépôt
- dépanner une machine qui ne boote pas
- systemd : targets, services, timers, montages, sockets
- périphériques, drivers et modules
- processus de compilation et d'installation d'un nouveau kernel
- paramètres d'amorçage et paramètres de tuning du noyau
- acquisition, rotation, centralisation des logs
- analyse de la charge système
- optimisation des performances

Annexes : GRUB 2 et l'UEFI ; GRUB 2 : configuration avancée et sécurité ; Gestion avancée et centralisation des logs ; Miroir de dépôts local ; Montée de version majeure ; Accents, locales et jeux de caractères ; Intégrité du système

Table des matières

1. Installation avancée et déploiement	5	5. Blocage, crash et dépannage d'urgence	151
Installation d'AlmaLinux 10	6	Booter autrement	152
Installation automatisée avec Kickstart	25	Démarrage du système et GRUB 2	161
2. Maîtriser la configuration logicielle du système	37	systemd : démarrage et services	174
Mettre à jour le système	38	6. Maintenance et métrologie	190
Maîtriser la compilation des logiciels	57	Les logs	191
Maîtriser la production de packages RPM	72	Analyse de la charge et de l'activité système	205
3. Filesystems et unités de stockage	86	7. Optimisation des performances	221
Stockage, Swap, Systèmes de fichiers	87	Optimiser les performances	222
4. Noyau et périphériques	112	8. Supervision	238
Périphériques, modules et udev	113	Collecter et grapher des métriques système	239
Le noyau : compilation et installation	128	Annexes	
Le noyau : paramètres de boot et sysctl	139		

CHAPITRE 1

Installation avancée et déploiement

MODULE 1

Installation d'AlmaLinux 10

De quoi parle-t-on

Ce module suit une installation d'AlmaLinux 10 de bout en bout, écran par écran : de quoi retrouver les choix qui comptent (*partitionnement*, *compte administrateur*) en situation, avant de les refaire soi-même.

Au programme

- besoins matériels et démarrage de l'installateur,
- partitionnement : schéma LVM, tailles recommandées,
- sélection des paquets, création du compte administrateur, finalisation.

Installation minimale : démarrage

Besoins matériels

Composant	Minimum	Recommandé
RAM	2 Go	1 Go par processeur logique (RH)
CPU	1 processeur (VM incluse)	2 processeurs
Affichage (graphique)	-	1024×768
Disque	10 Go (base courante)	Selon usage
Architecture	x86-64-v3 (processeur de 2013 ou plus récent)	AlmaLinux seul : ISO <code>x86_64_v2</code> pour un matériel plus ancien

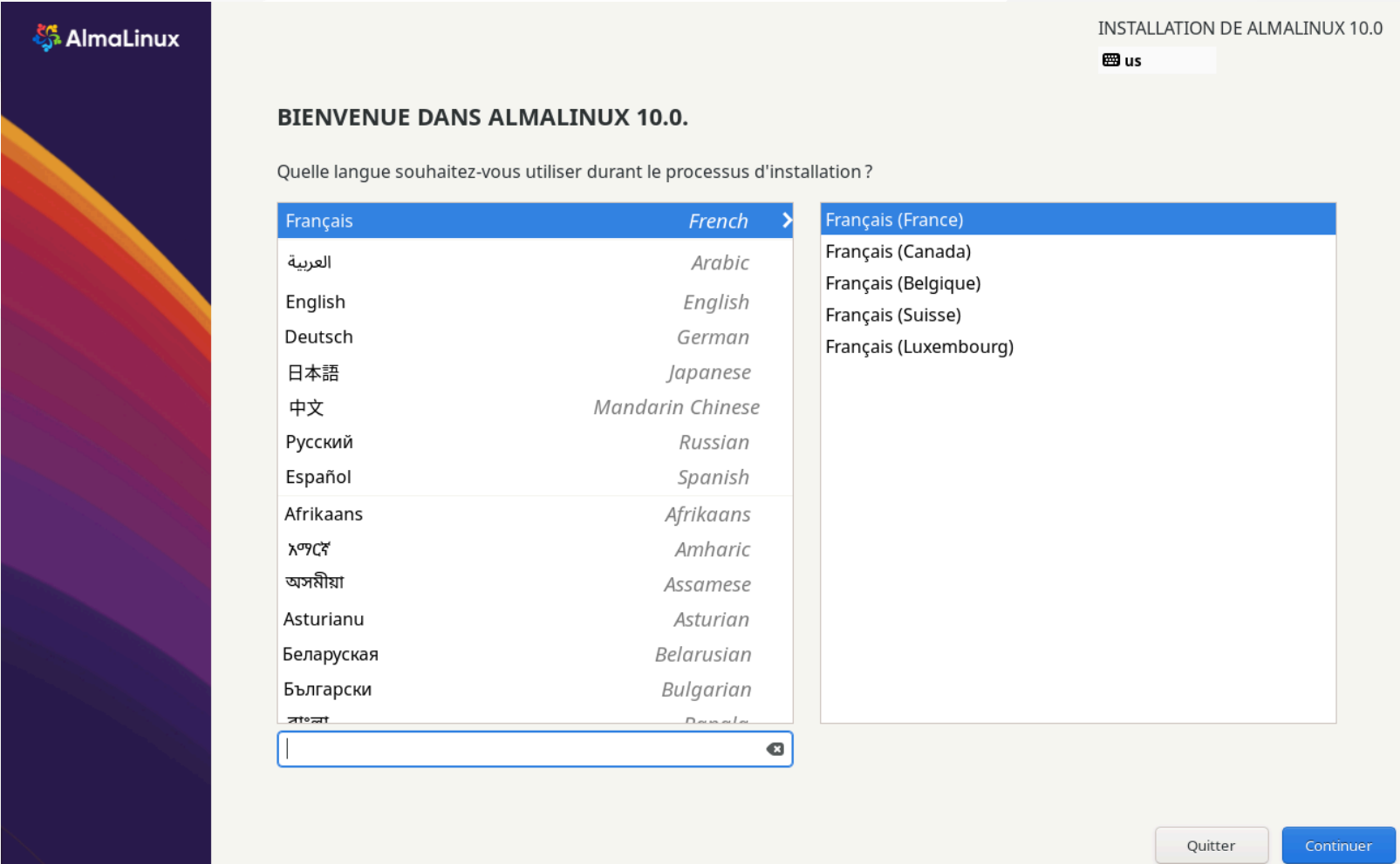
Booter



Écran d'accueil & Choix de la langue

 Langue

Première étape : sélectionner la **langue** utilisée pour le processus d'installation.



Panneau de contrôle de l'installation

Si des choix nécessaires ne sont pas faits, ils sont **mis en évidence**.

D'autres points peuvent nécessiter un peu d'attention...



RÉSUMÉ DE L'INSTALLATION

INSTALLATION DE ALMALINUX 10.0

fr (oss)

LOCALISATION

 **Clavier**
Français (variante)

 **Langues prises en charge**
Français (France)

 **Heure et date**
Fuseau horaire Europe/Paris

UTILISATEURS

 **Compte root**
Le compte root est désactivé

 **Création de l'utilisateur**
Aucun utilisateur ne sera créé

LOGICIELS

 **Source d'installation**
Auto-détection de la source

 **Sélection de logiciels**
Installation minimale

SYSTÈME

 **Destination de l'installation**
Partitionnement automatique sélectionné

 **KDUMP**
Kdump est activé

 **Réseau et nom d'hôte**
Connecté : enp1s0

Quitter

Commencer l'installation

Vos disques resteront intacts tant que vous n'aurez pas cliqué sur 'Commencer l'installation'.

 Complétez les éléments marqués de cette icône avant de passer à l'étape suivante.

26.10 - François MICAUX - ACTILIS 2007-2026 - lin/modules/installation

11/252

Nom d'hôte et réseau

Configurer le nom d'hôte et les interfaces réseau.

Quand c'est **"Fait"**, cliquer sur **"Terminé"**.

RÉSEAU ET NOM D'HÔTE

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

Ethernet (enp1s0)
Red Hat, Inc. Virtio 1.0 network device

Ethernet (enp1s0)
Connecté

Adresse du matériel 52:54:00:09:C6:5F

Vitesse

Adresse IP 192.168.201.221/24

Route par défaut 192.168.201.1

DNS 192.168.201.1

Configurer...

Nom d'hôte : alma10-demo.local

Appliquer

Nom d'hôte actuel: localhost

Choix du disque cible

Choisir le disque à partitionner, le mode **"Personnalisé"**, puis cliquer sur **"Terminé"**.

DESTINATION DE L'INSTALLATION

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

Sélection des périphériques

Sélectionnez les périphériques que vous souhaitez utiliser pour l'installation. Ils resteront intacts tant que vous n'aurez pas cliqué sur le bouton "Commencer l'installation" du menu principal.

Disques standards locaux

10 Gio

0x1af4

vda / 10 Gio d'espace libre

Les disques décochés ne seront pas modifiés.

Disques spéciaux et réseau

Ajouter un disque...

Les disques décochés ne seront pas modifiés.

Configuration du stockage

☐ Automatique

☒ Personnalisée

[Résumé complet des disques et bootloader...](#)

1 disque sélectionné ; 10 Gio de capacité ; 10 Gio d'espace libre [Actualiser...](#)

Partitionnement

PARTITIONNEMENT MANUEL

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

▼Nouvelle installation de AlmaLinux 10.0

Vous n'avez pas encore créé de point de montage pour votre installation de AlmaLinux 10.0. Vous pouvez :

- [Cliquez ici pour les créer automatiquement.](#)
- Créer de nouveaux points de montage en cliquant sur le bouton « + ».

Les nouveaux points de montage utiliseront le schéma de partitionnement suivant :

LVM

Chiffrer crée automatiquement les points de montage par défaut :

☐ Chiffrer mes données.

+ - ↺

ESPACE DISPONIBLE

10 Gio

ESPACE TOTAL

10 Gio

[1 périphérique de stockage sélectionné](#)

Ignorer toutes les modifications

Quand vous aurez créé des points de montage pour l'installation de AlmaLinux 10.0, vous pourrez en voir les détails [ici](#).

💡 LVM conseillé

LVM reste le choix conseillé pour la gestion des volumes de stockage.

26.10 - François MICAUX - ACTILIS 2007-2026 - lin/modules/installation

14/252

Partitionnement : Quelles partitions créer ?

Au moins **3 partitions** sont nécessaires :

- `/` (racine) : le *root filesystem*
- `swap` : la partition de swap (*swap primaire*)
- `/boot` : une **partition à part**

🔗 Il faut aussi une **partition de démarrage du firmware**

En **BIOS** : une petite partition `biosboot`.

En **UEFI** : l'**ESP** (*EFI System Partition*, FAT) montée sur `/boot/efi`.

`/boot` **ne peut pas** être un volume LVM : même avec `/` sur LVM, c'est une partition standard.

Le répertoire `/boot` contient :

- Les fichiers du chargeur de démarrage (**GRUB**)
- Le fichier noyau : `/boot/vmlinuz-x.y.z...` (*horodatage = celui de la distribution*)
- Le ramdisk initial : `/boot/initramfs-x.y.z.img` (*constitué à l'installation, il intègre les pilotes pour booter sur votre matériel*)

Partitionnement : Tailles recommandées

Partition	Taille recommandée	Remarques
/boot	1 Go	environ 200 Mo par noyau installé (la documentation Red Hat recommande 2 GiB)
/ et /usr	5 Go (serveur) / 25 Go+ (poste)	/ et /usr ne sont plus séparables. 5 Go minimum pour les TP
/var	≥ 1 Go	Minimum conseillé
/var/log	À définir	Éviter saturation de /var par les logs
/var/lib	À définir	BDD (Maria / Postgres)...
/var/lib/containers	Selon usage	Images et volumes de Podman
/var/spool	À définir	Éviter saturation par les spools (mail...)
/home et /srv	Selon usage	Dépendent de l'utilisation du serveur
/usr/src	≥ 10 Go	Sources et compilation du noyau (6.x)

Partitionnement : Interface

PARTITIONNEMENT MANUEL

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

▼ Nouvelle installation de AlmaLinux 10.0

SYSTÈME

/

rootvg-rootlv

3 Gio >

/boot

vda2

512 Mio

BIOS Boot

vda1

1024 Kio

swap

vda3

1024 Mio

+ - ↺

ESPACE DISPONIBLE

1,97 Mio

ESPACE TOTAL

10 Gio

[1 périphérique de stockage sélectionné](#)

rootvg-rootlv

Point de montage :

/

Capacité souhaitée :

3 Gio

Type de périphérique :

LVM

☐ Chiffrer

Système de fichiers :

xfs

☒ Reformater

Étiquette :

Périphérique :

0x1af4 (vda)

Modifier...

Groupe De Volumes :

rootvg (5,5 Gio d'espace libre)

Modifier...

Nom :

rootlv

Mise à jour des paramètres

Remarque : Les paramètres que vous définissez sur cet écran ne seront pas appliqués tant que vous n'aurez pas cliqué sur le bouton 'Commencer l'installation' du menu principal.

Ignorer toutes les modifications

Quand c'est terminé : "Terminé" !

Partitionnement : Configuration LVM

CONFIGURER GROUPE DE VOLUMES

Veuillez créer un nom pour ce groupe de volumes et choisir au moins un des disques ci-dessous.

Nom :

Description	Nom	Capacité	Libre
0x1af4	vda	10 Gio	1023 Kio

Niveau RAID :

Aucune ▼

☐ Chiffrer

Stratégie sur la taille

Aussi grand que possible ▼

Annuler

Enregistrer

Sélection des packages

Choisir l'environnement d'installation selon l'usage cible.

SÉLECTION DE LOGICIELS

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

Environnement de base

☒ Installation minimale

Fonctionnalité de base.

Logiciels supplémentaires pour l'environnement sélectionné

☒ Standard

Installation standard de AlmaLinux.

☐ Outils de développement

Un environnement de développement de base.

☐ Gestion à distance sans périphérique de contrôle

Outils pour gérer le système sans console graphique associée.

☐ Compatibilité héritée UNIX

Programmes de compatibilité pour des migrations en provenance de, ou pour travailler, avec des environnements hérités de UNIX.

☐ Serveurs de réseau

Ces packages comprennent des serveurs basés sur le réseau comme DHCP, Kerberos et NIS.

☐ Prise en charge Scientifc

Outils pour effectuer des calculs mathématiques et scientifiques, et des traitements parallèles.

☐ Outils de sécurité

Outils de sécurité pour vérification de confiance et intégrité.

☐ Prise en charge Smart Card

Prise en charge de l'authentification avec carte à puce.

☐ Outils système

Ce groupe est un ensemble d'outils système, ainsi qu'un client pour la connexion aux parts SMB et d'outils permettant de gérer le trafic dans le réseau.

L'utilisateur administrateur

Deux époques, deux approches :

PARAMÈTRES UTILISATEUR



Mot de passe administrateur

Le mot de passe administrateur n'est pas défini



Création Utilisateur

Aucun utilisateur ne sera créé

- ⇒ **Auparavant** : on devait choisir un mot de passe pour `root`.
- ⇒ **Aujourd'hui** : pas de mot de passe pour root = connexion root impossible. On passe par un **utilisateur administratif** déclaré dans `sudo` pour élever ses privilèges.

Création de l'utilisateur administratif

CRÉER UN UTILISATEUR

Terminé

INSTALLATION DE ALMALINUX 10.0

fr (oss)

Nom complet

almalinux

Nom d'utilisateur

almalinux

☒ Accorder les privilèges d'administration à ce compte utilisateur (ajout au groupe wheel)

☒ Exiger un mot de passe pour utiliser ce compte

Mot de passe

●●●●●●

👁

Faible

Confirmez le mot de passe

●●●●●●

👁

Avancé...

⚠

Le mot de passe ne passe pas la vérification dans le dictionnaire - basé sur un mot du dictionnaire. Vous devrez appuyer deux fois sur **Terminé** pour le confirmer.

Avancé (Paramètres du compte utilisateur)

CONFIGURATION AVANCÉE DE L'UTILISATEUR

Répertoire personnel:

ID d'utilisateur et ID de groupe

☐ Spécifier manuellement un ID d'utilisateur :

☐ Spécifier manuellement un ID de groupe :

Ajout à des groupes

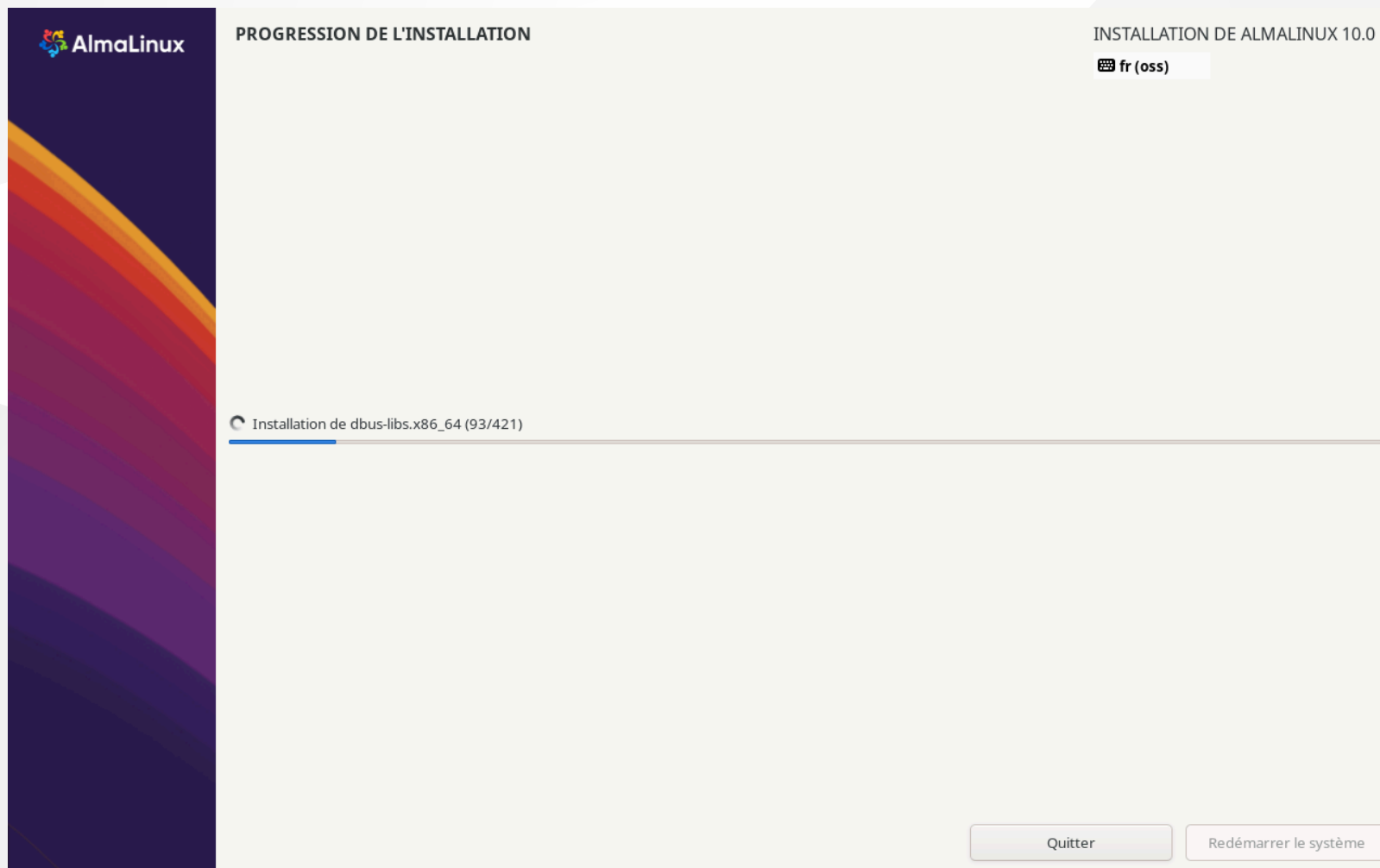
Ajouter l'utilisateur aux groupes suivants:

Exemple : wheel, mon-équipe (1245), projet-x (29935)

Astuce : Ici, vous pouvez saisir une liste de noms et d'ID de groupe séparés par des virgules. Les groupes qui n'existent pas déjà seront créés ; spécifiez leur GID entre parenthèses.

Lancement de l'installation et redémarrage

Il n'y a plus qu'à attendre un peu le temps de **finaliser l'installation** et redémarrer...



Bonnes pratiques

- Choisir le partitionnement avant de lancer l'installateur, avec une marge pour `/var` et `/var/lib/containers`.
- Mettre `/boot` en partition standard et le reste en LVM : les volumes restent extensibles.
- Séparer `/var/log` : des logs bavards ne saturent plus la racine.
- Vérifier le matériel : RHEL 10 exige x86-64-v3, AlmaLinux 10 propose aussi une variante `x86_64_v2`.
- Conserver l'`anaconda-ks.cfg` produit : il sert de base au prochain kickstart.
- Installation minimale, puis ajouter ce qui est utile.

MODULE 2

Installation automatisée avec Kickstart

De quoi parle-t-on

Kickstart automatise une installation Anaconda en lui fournissant, à l'avance, toutes les réponses attendues : un vrai gain de temps dès qu'on installe plus d'une machine à l'identique.

Au programme

- principe : Anaconda + un fichier de réponses,
- démarche : d'une installation manuelle à un fichier `.ks` réutilisable,
- syntaxe : partitionnement, utilisateurs, paquets, scripts `%pre/%post`.

Principe : rejouer automatiquement une installation

🔗 Objectif

Objectif : Fournir à l'installateur un **fichier de réponses** à ses questions.

À chaque installation, **Anaconda**, le système d'installation de RedHat, dépose `/root/anaconda-ks.cfg`, un fichier reprenant les réponses données, au bon format... à quelques éléments près.

Ce fichier est le **point de départ naturel** pour automatiser les installations suivantes.

Anaconda + Kickstart

Anaconda peut être automatisé grâce au système **Kickstart**.

Le fichier texte de réponses peut être :

- **sur le média** d'installation,
- ou accessible **par une URL** (HTTP, HTTPS, FTP, NFS).

Au démarrage de l'installation, on précise où le trouver via l'option de boot :

```
inst.ks=[protocole://][chemin/]fichier
```

Doc officielle : [Automatically installing RHEL \(10\)](#)

Démarche

1. Réaliser une **installation manuelle** proche du besoin final.
En mode graphique : le mode texte ne propose pas toujours toutes les possibilités.
2. Récupérer `/root/anaconda-ks.cfg` sur le système installé.
3. **Éditer une copie**, l'adapter (partitionnement, packages, utilisateurs, scripts).
4. Tester sur une VM, retester sur la même VM.

Référence des commandes : [Kickstart commands and options reference](#)

Utilisation : booter avec `inst.ks=`

Boot **réseau** (PXE, BOOTP + TFTP) ou **local** (ISO `boot` ou `minimal`, voir slide « Se procurer l'ISO d'installation »).

Au menu de boot : touche **Tab** sur le premier choix, puis ajouter `inst.ks=...`.

```
# Protocoles acceptés
inst.ks=http://serveur/chemin/fichier.cfg
inst.ks=https://serveur/chemin/fichier.cfg
inst.ks=ftp://serveur/chemin/fichier.cfg
inst.ks=nfs:serveur:/chemin/fichier.cfg
inst.ks=cdrom:/fichier.cfg
```

Déroulé

Cas nominal : **aucune question** posée pendant l'installation.

Erreurs possibles : fichier kickstart invalide, URL inaccessible.

Kickstart dans l'ISO

La commande `mkksiso` (paquet `lorax`) fabrique une ISO dont les lignes de démarrage portent déjà `inst.ks=` : plus rien à taper au démarrage. Elle est mise en oeuvre au TP. Pour le démarrage par le réseau, voir slide « PXE : démarrer par le réseau avec `dnsmasq` ».

Se procurer l'ISO d'installation

Les ISO d'AlmaLinux 10 se téléchargent sur le [miroir officiel](#) :

ISO	Contenu	Taille
boot	l'installateur seul, les paquets viennent du réseau	1 Go
minimal	l'installateur et un système de base	1,6 Go
dvd	tous les paquets	10 Go

Le téléchargement et le contrôle de l'empreinte se font au TP.

 Nom du fichier et processeurs anciens

`latest` désigne la dernière version 10.x. Le dossier `isos/x86_64_v2/` propose les mêmes ISO pour un processeur plus ancien.

Syntaxe du kickstart

Système, langue, réseau

```
≡ # Mode et redémarrage
text
reboot

# Sécurité de base
firewall --enabled --service=ssh
firstboot --disable

# Source d'installation : le réseau (ISO de démarrage)
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/"

# Localisation
keyboard --xlayouts='fr (oss)'
lang fr_FR.UTF-8
timezone Europe/Paris --utc

# Réseau
network --bootproto=dhcp --activate
network --hostname=srv-ks.exemple.lan
```

Partitionnement (avec LVM)

```
≡ ignoredisk --only-use=vda
clearpart --all --initlabel
zerombr

# Partitions physiques
part /boot --asprimary --fstype ext4 --size=1024 --ondisk=vda
part swap --asprimary --fstype="swap" --size=1024 --ondisk=vda
part pv.01 --asprimary --size=1 --grow --ondisk=vda

# Volume Group + Logical Volumes
volgroup vgroot pv.01
logvol / --fstype ext4 --name=lvroot --vgname=vgroot --size=3072
logvol /var --fstype ext4 --name=lvvar --vgname=vgroot --size=1024
```


Utilisateurs et add-ons

```
≡ # Pas de mot de passe root : compte verrouillé
rootpw --lock

# Utilisateur administratif (membre de wheel)
user --groups=wheel --name=admin \
    --password='$6$e/l0cmS3s1548SH5$SvrLHPWlhTr7lElVG0bL...' \
    --iscrypted --gecos="Administrateur"

# Add-on kdump
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
```

Mot de passe chiffré

Le mot de passe figure **déjà chiffré** dans le fichier (`--iscrypted`).

Section %packages

```
≡ %packages
  @^minimal-environment
  openssh-clients
  bash-completion
%end
```

Syntaxe	Signification
@nom	Groupe de packages
nom	Package individuel
- nom	Exclusion
@^nom	Environnement (ex. @^minimal-environment)

Scripts `%pre` et `%post`

Exécutés par `sh` (sauf interpréteur explicite via `--interpreter=`).

Post-install dans le chroot du futur système

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

Post-install hors chroot : "/" du futur système = `/mnt/sysimage`

```
%post --nochroot
echo "172.17.1.29 my-old-proxy.local" >> /mnt/sysimage/etc/hosts
%end
```

⚠ Écrire dans `/mnt/sysimage`

Avec `--nochroot`, écrire dans `/mnt/sysimage/...` (pas dans `/`). Selon la version d'Anaconda, le même arbre s'appelle aussi `/mnt/sysroot`.

Bonnes pratiques

- Partir d'un `anaconda-ks.cfg` réel, puis l'adapter.
- Contrôler le fichier avec `ksvalidator` avant de s'en servir.
- Un `%post` journalisé (`--log=`) : l'échec s'analyse après coup.
- Mot de passe `root` et utilisateurs : hash, pas de texte clair.
- Fichier versionné, servi en lecture seule.
- Effacement de disque (`clearpart`) : disques cibles nommés explicitement.
- Tester sur une VM avant le parc.

CHAPITRE 2

Maîtriser la configuration logicielle du système

MODULE 3

Mettre à jour le système

Monde RedHat / CentOS / RockyLinux / AlmaLinux

De quoi parle-t-on

Dans le monde RedHat, tout passe par le format **RPM** et une commande `rpm`, aujourd'hui pilotée par **DNF**. Ce module couvre le format, l'outil bas niveau `rpm`, et l'usage courant de la commande `dnf` : dépôts, recherche, mise à jour, cache.

Au programme

- le format RPM et la commande `rpm`,
- configurer et déclarer un dépôt,
- l'usage courant de `dnf` : chercher, installer, mettre à jour, gérer le cache.

Panorama des gestionnaires de paquets

Famille	Outil bas niveau	Méta-outil(s)	Distributions
RPM	rpm	yum (≤ RHEL 7) puis dnf, dnf5	RHEL, Alma, Rocky, CentOS, Fedora
RPM	rpm	zypper	(open)SUSE
RPM	rpm	urpmi	Mageia, ex Mandriva
DEB	dpkg	apt, apt-get, aptitude	Debian, Ubuntu, dérivés
APK	apk	apk	Alpine Linux
autre	pacman	pacman	Arch Linux, Manjaro
autre	divers	emerge (Portage)	Gentoo
autre	divers	slackpkg, slapt-get	Slackware

Aide-mémoire complet : [Package Management Cheatsheet \(distrowatch.com\)](https://distrowatch.com)

Le format RPM

Structure d'un fichier `.rpm`

Format mis au point par Red Hat. Un paquet RPM contient :

Section	Contenu
lead	En-tête de 96 octets : format, nom/version (66 o.), type binaire/source, OS cible, version de RPM
signatures	Taille du paquet et du payload, empreintes SHA-256 et SHA1 du header, MD5, signature GPG (visibles par <code>rpm --checksig --verbose</code>)
header	Tous les <code>RPMTAGS</code> : description, dépendances, provenance, scripts pre/post (visibles par <code>rpm -qi</code> , <code>--scripts</code> , <code>--requires</code> , <code>--provides</code>)
payload	Le contenu : archive cpio compressée en zstd sur RHEL 10 (listable avec <code>rpm -qpI</code> , <code>-qc</code> , <code>-qd</code>)

La commande rpm

rpm est à la fois la commande de gestion et une interface vers d'autres outils. Les binaires associés :

```
$ rpm -qlf $(which rpm) | grep -E '/s?bin/'
/usr/bin/rpm
/usr/bin/rpm2archive
/usr/bin/rpm2cpio
/usr/bin/rpmdb
/usr/bin/rpmkeys
/usr/bin/rpmquery      # synonyme de "rpm -q"
/usr/bin/rpmsort
/usr/bin/rpmverify
```

Binaire	Rôle
rpm	Installer, supprimer, interroger
rpm2cpio	Extraire le payload sans installer (utile en récupération)
rpmquery	Requêtes (équivalent rpm -q)
rpmverify	Vérifier l'intégrité des fichiers installés
rpmkeys	Gérer les clés GPG et vérifier les signatures
rpmsign	Signer un paquet (fourni par le paquet rpm-sign)

Installer, mettre à jour, désinstaller

```
# rpm -ivh paquet-version.arch.rpm    # Installation
rpm -Uvh paquet-version.arch.rpm      # Update (installe si absent, sinon met à jour)
rpm -Fvh paquet-version.arch.rpm      # Freshen : ne met à jour que si déjà installé
rpm -e paquet                          # Désinstallation
```

- **-v** : verbeux.
- **-h** : barre de progression ("hash marks").
- **rpm** **ne résout pas les dépendances** et refuse l'opération si une dépendance manque.
- En production, on utilise **dnf** qui résout les dépendances et gère les dépôts.

💡 rpm ou dnf

En pratique : **rpm** direct est utile pour un paquet isolé hors dépôt, pour de l'analyse, ou en mode rescue. Pour la maintenance courante, passer par **dnf**.

Les requêtes : `rpm -q` (alias `rpmquery`)

```
$ rpm -qa                # Lister tous les paquets installés
rpm -q bash             # Un paquet est-il installé ?
rpm -qi bash            # Infos (tags) d'un paquet installé
rpm -qpi paquet.rpm     # Infos d'un fichier .rpm (option -p = "package file")
rpm -ql bash            # Contenu (payload) d'un paquet installé
rpm -qpl paquet.rpm     # Contenu d'un fichier .rpm
rpm -qc bash            # Variante : seulement les fichiers de configuration
rpm -qd bash            # Variante : seulement la documentation
rpm -ql bash --dump      # Détails par fichier (taille, hash, perms, mtime...)
rpm -qf /etc/passwd     # Quel paquet a installé ce fichier ?

rpm -q --requires bash  # Dépendances nécessaires
rpm -q --provides bash  # Capacités fournies
rpm -q --scripts systemd # Scripts pre/post install/uninstall
```

Configuration de DNF

Fichiers de configuration

dnf hérite directement de la configuration de yum. Le fichier principal est commun :

```
$ ls -l /etc/yum.conf
lrwxrwxrwx. 1 root root 12 /etc/yum.conf -> dnf/dnf.conf
```

Élément	Emplacement
Configuration principale	/etc/dnf/dnf.conf (ou /etc/yum.conf)
Dépôts	/etc/yum.repos.d/*.repo
Plugins	/etc/dnf/plugins/ (ou /etc/yum/pluginconf.d/ pour les anciens)

Format INI : sections entre crochets, directives clé=valeur, commentaires en #.

```
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
skip_if_unavailable=False
```

Paramètres de la section [main]

Directive	Effet
cachedir	Emplacement du cache (défaut : /var/cache/dnf)
keepcache	1 = garder les RPM téléchargés après installation
gpgcheck	1 = vérifier la signature GPG des paquets (inhibé par --nogpgcheck)
plugins	true pour activer les plugins déclarés
installonly_limit	nombre de noyaux gardés installés (3)
metadata_expire	Durée de vie des méta-données en cache (défaut : 48 h pour DNF, 24 h dans les dépôts AlmaLinux)

Passage par un proxy authentifié (global ou par dépôt)

```
≡ proxy=http://localhost:3128/  
  proxy_username=usera  
  proxy_password=passa  
  http_caching=all           # all | packages | none
```

Désactiver le proxy global sur un dépôt : proxy=_none_ dans la section du dépôt.

Doc de référence : man 5 dnf.conf

Déclaration d'un dépôt

Chaque fichier `/etc/yum.repos.d/*.repo` peut contenir une ou plusieurs sections :

```
[baseos]
name=AlmaLinux $releasever - BaseOS
mirrorlist=https://mirrors.almaLinux.org/mirrorlist/$releasever/baseos
# baseurl=https://repo.almaLinux.org/almaLinux/$releasever/BaseOS/$basearch/os/
enabled=1
gpgcheck=1
countme=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-AlmaLinux-10
metadata_expire=86400
enabled_metadata=1
```

Directive	Rôle
<code>baseurl</code>	URL fixe pointant vers la racine du dépôt
<code>mirrorlist</code>	URL renvoyant une liste de miroirs (<i>le premier joignable est utilisé</i>)
<code>enabled</code>	<code>0</code> ou <code>1</code>
<code>gpgcheck</code> / <code>gpgkey</code>	Vérification de signature et clé associée

En ligne de commande : `--enablerepo=NOM` / `--disablerepo=NOM` (*joker `*` accepté*).

De `yum` à `dnf`

`yum` (Yellowdog Updater Modified, 2003) est devenu lent et difficile à maintenir (Python 2). À partir de RHEL 8 / CentOS 8, il a été remplacé par **DNF** (Dandified Yum), réécrit autour de `libdnf` et de `librepo`.

```
$ ls -l /bin/yum
lrwxrwxrwx. 1 root root  /bin/yum -> dnf-3
```

- L'interface en ligne de commande reste **largement compatible**.
- Les **mêmes dépôts** sont utilisés (mêmes méta-données : `repomd.xml`, `primary.xml.gz`, `filelists.xml.gz`...).
- `dnf` **résout les dépendances** et choisit les paquets selon les versions disponibles dans les dépôts actifs.

`dnf5`

`dnf5` (réécriture en C++) est le gestionnaire par défaut de Fedora depuis la version 41. RHEL 10 et AlmaLinux 10 livrent **DNF 4** (`dnf` 4.20) : `dnf5` n'est pas dans leurs dépôts.

Doc DNF : dnf.readthedocs.io

Lister et explorer les dépôts

```
$ dnf repolist          # Dépôts actifs (enabled=1)
dnf repolist all       # Tous les dépôts configurés, actifs ou pas
dnf repolist --enabled # Idem repolist
dnf repolist --disabled # Dépôts désactivés uniquement
dnf repoinfo NOM        # Détails d'un dépôt (URL, miroirs, état, nb de paquets)
```

Exemple sur AlmaLinux 10 :

```
$ dnf repolist
repo id                               repo name
appstream                            AlmaLinux 10 - AppStream
baseos                               AlmaLinux 10 - BaseOS
extras                               AlmaLinux 10 - Extras
```

i Les dépôts

Les dépôts **AppStream** contiennent les applications et bibliothèques utilisateur.
BaseOS contient le noyau du système.
CRB (bibliothèques et outils de développement) et les dépôts `*-debuginfo` et `*-source` existent mais sont désactivés par défaut (`dnf repolist all`).

Chercher des paquets

Par nom de package

```
$ dnf list                # Tous les paquets (installés + disponibles)
dnf list installed        # Uniquement les paquets installés
dnf list available        # Paquets disponibles non installés
dnf list updates          # Mises à jour disponibles
dnf list extras           # Installés mais absents des dépôts actifs
dnf list obsoletes        # Rendus obsolètes par d'autres paquets
dnf list recent           # Ajoutés récemment aux dépôts (cf. directive recent=)
dnf list --showduplicates curl # Toutes les versions disponibles
```

Par la description

```
$ dnf search MOT          # Recherche dans name + summary + description
dnf info PAQUET           # Détails (équivalent rpm -qi, fonctionne hors installation)
```

Par un nom de fichier ou une dépendance

```
$ dnf provides "*/lspci"  # Quel paquet fournit ce fichier ?
dnf repoquery --requires PAQUET # Dépendances (sans nécessiter l'installation)
```

Méta-données utilisées

`dnf search` exploite `primary.xml.gz`.

`dnf provides` nécessite `filelists.xml.gz`. Si une recherche par chemin de fichier renvoie peu, vérifier que les `filelists` sont bien téléchargées (cf. `dnf makecache`).

Installer, supprimer, annuler

```
# dnf install PAQUET1 [PAQUET2...] # Installation (ou mise à jour si déjà installé)
dnf reinstall PAQUET                # Réinstaller la version courante
dnf downgrade PAQUET                # Revenir à une version antérieure disponible
dnf remove PAQUET                   # Désinstaller (et ses dépendances orphelines)
dnf autoremove                      # Nettoyer les dépendances orphelines
```

L'historique

```
$ dnf history                        # Journal des transactions
dnf history info ID                 # Détails d'une transaction
sudo dnf history undo ID            # Annuler une transaction
```

- `dnf install ./paquet.rpm` installe un RPM **local** en résolvant ses dépendances depuis les dépôts.
- Les transactions sont **journalisées** dans `/var/log/dnf.rpm.log` et la base SQLite `history.sqlite`.

⚠ Limite de l'undo

L'`undo` n'est pas toujours possible : un paquet absent des dépôts actifs ne pourra pas être réinstallé.

Mettre à jour le système

Y a-t-il des mises à jour ? (sortie + code retour)

```
$ dnf check-update          # 0 = rien à faire
                             # 100 = mises à jour disponibles
                             # 1 = erreur
```

Mettre à jour

```
# dnf update [PAQUET...]    # Mise à jour prudente (ne supprime pas d'obsolètes)
# dnf upgrade [PAQUET...]   # Mise à jour complète, prend en charge les "obsoletes"
# dnf -y upgrade            # Mode non interactif (à manier avec précaution)

# dnf upgrade --security    # Uniquement les correctifs marqués "security"
```

```
$ dnf updateinfo list      # Avis publiés (security / bugfix / enhancement)
```

🔍 update et upgrade

Sur DNF 4 (RHEL 8/9/10), `dnf update` et `dnf upgrade` sont des **alias**. Sur DNF 5, seul `upgrade` est conservé.

Traiter les paquets par groupes

Les groupes regroupent des paquets cohérents (ex. "Serveur web", "Outils de développement").

Chaque groupe contient des paquets **obligatoires**, **par défaut** et **optionnels**.

```
$ dnf grouplist # Lister les groupes (installés + disponibles)
dnf groupinfo "Development Tools"
```

```
# dnf groupinstall "Development Tools"
dnf groupupdate "Development Tools"
dnf groupremove "Development Tools"
```

Avec `grouplist`, un groupe est marqué **Installé** si ses paquets **obligatoires** sont présents... ou s'il n'en a pas, au moins un paquet **par défaut** ou **optionnel** est présent.

`groupinstall` n'installe **pas** les paquets optionnels du groupe.

Pour les inclure :

```
# dnf install @"Nom du groupe" --setopt=group_package_types=mandatory,default,optional
```

Modules et streams (*héritage CentOS 8 / RHEL 8-9*)

Apparus avec DNF, les **modules** regroupent des paquets formant une application ou un langage cohérent (ex. `php`, `nodejs`). Un module peut exister en plusieurs **streams** (versions), avec différents **profils** (jeu de paquets selon l'usage).

```
$ dnf module list                # Modules disponibles, streams et profils
dnf module info php
```

```
# dnf module enable php:8.2
dnf module install php:8.2/devel
dnf module reset php           # Annuler le choix de stream
```

⚠ Version 10 de RedHat et de ses dérivées

La modularité est **dépréciée** : `dnf module list` ne renvoie rien : **aucun contenu modulaire n'est distribué**. Les versions multiples (`nodejs`, `postgresql`, `ruby`...) sont des paquets ordinaires, les commandes ci-dessus servent aux versions 8 et 9. La fonctionnalité disparaîtra dans une version majeure ultérieure.

Doc historique : docs.fedoraproject.org/modularity

Gérer le cache

DNF stocke localement les méta-données des dépôts pour ne pas les retélécharger à chaque commande. Durée de vie pilotée par `metadata_expire` (par défaut **48 h, 24 h** dans les dépôts AlmaLinux).

```
# dnf makecache           # Forcer le téléchargement complet des méta-données
dnf makecache --refresh   # Idem en ignorant la fraîcheur du cache courant

dnf clean expire-cache    # Marquer le cache comme expiré (re-téléchargement au prochain appel)
dnf clean metadata        # Supprimer les méta-données
dnf clean packages        # Supprimer les RPM téléchargés (cf. keepcache)
dnf clean dbcache         # Supprimer le cache SQLite des méta-données
dnf clean all             # Tout nettoyer (uniquement sur les dépôts activés)

dnf --enablerepo='*' clean all  # Vraiment tout, y compris les dépôts désactivés
```

Sans makecache

Sans `makecache`, les méta-données sont téléchargées **à la demande** lors des requêtes. C'est plus économique en interactif, moins prévisible pour des scripts.

Bonnes pratiques

- Lire les errata (`dnf updateinfo`) avant une mise à jour de sécurité.
- `dnf history` : repérer la transaction avant d'annuler (`undo`).
- Clés GPG importées depuis une source de confiance, `gpgcheck=1` partout.
- Peu de dépôts tiers : chacun élargit la surface d'attaque.
- Mise à jour du noyau suivie d'un redémarrage planifié.
- Tester sur une machine pilote avant le parc.
- Éviter `dnf clean all` en réflexe : il oblige à retélécharger les méta-données.

MODULE 4

Maîtriser la compilation des logiciels

De quoi parle-t-on

Installer un paquet binaire ne suffit pas toujours : ce module couvre la compilation depuis les sources, du tarball au binaire installé, puis le fonctionnement des Makefiles et de l'édition de liens.

Au programme

- pourquoi et comment compiler : du tarball au `make install`,
- comprendre et écrire un Makefile,
- liaison statique/dynamique, `ld.so`, `ldconfig`, `LD_PRELOAD`.

Compiler ou installer un paquet binaire ?

Deux approches complémentaires

- Installer des **paquets binaires**
- Compiler soi-même depuis les sources.

Pourquoi compiler ?

✎ Tenir à jour

Tenir à jour, parfois plus vite ou plus précisément que la distribution ne le permet.

- Profiter d'**optimisations** propres à l'architecture cible.
- Choisir l'**emplacement** des fichiers (par défaut `/usr/local`), facilite désinstallation et cohabitation de versions.
- Choisir les **modules / fonctionnalités** activés, les versions des bibliothèques.
- Réagir vite à un **correctif de sécurité** sans attendre le paquet de la distribution.

Veille : [CVE.org](https://www.cve.org), [Red Hat Security Advisories](https://www.redhat.com/en/security/updates/advisories), [errata AlmaLinux](https://errata.almalinux.org/), exploit-db.com.

Préparer la machine de compilation

Il faut un compilateur, `make` et les **fichiers d'en-têtes** (paquets `*-devel`) des bibliothèques dont le logiciel dépend.

```
# dnf group install "Development Tools"  
dnf install openssl-devel zlib-devel
```

- Le groupe `Development Tools` apporte `gcc`, `make`, `autoconf`, `patch`...
- Un message `configure: error: ... not found` désigne en général un paquet `-devel` manquant : `dnf provides '*/openssl/ssl.h'` retrouve lequel.
- Certains paquets `-devel` et `glibc-static` (pour `gcc -static`) sont dans le dépôt **CRB**.

💡 Compiler sans polluer le système

`./configure --prefix=/opt/foo` et un compte non privilégié pour `make` : seule l'étape `make install` demande `root`.

Démarche : du tarball au binaire

1. Téléchargement et vérification du contenu

```
$ curl -O https://exemple.org/foo-1.2.3.tar.xz  
sha256sum -c foo-1.2.3.tar.xz.sha256  
xz -dc foo-1.2.3.tar.xz | tar t | head
```

2. Extraction (gzip/bzip2/xz selon le suffixe)

```
$ tar xf foo-1.2.3.tar.xz  
cd foo-1.2.3
```

3. Lire la doc d'accueil

```
$ less README INSTALL
```

4. Configuration et génération des Makefiles

```
$ ./configure --help  
./configure --prefix=/opt/foo \  
             --sysconfdir=/etc/foo \  
             --enable-xxx --with-yyy
```

🔍 --prefix

--prefix cantonne l'installation à une arborescence dédiée (ex. /opt/foo).

Compilation, tests, installation

5. Compilation

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

6. Tests (quand le projet le propose)

```
$ make test # ou make check / make tests selon le projet # (PHP, zlib, OpenSSL... en proposent)
```

7. Installation (root)

```
# sudo make install
```

Finalisation : manuelle le plus souvent.

À faire	Où
Référencer les pages de man	<code>/etc/man_db.conf</code>
Ajouter les binaires au <code>PATH</code>	<code>/etc/profile.d/foo.sh</code> , <code>~/.bash_profile</code> , <code>/etc/environment</code>
Référencer les bibliothèques partagées	<code>/etc/ld.so.conf.d/*.conf</code> + <code>ldconfig</code>
Démarrage automatique	Unit <code>systemd</code> (<code>/etc/systemd/system/foo.service</code>)

À quoi sert un Makefile ?

Un **Makefile** est consommé par `make` et décrit **comment** construire un programme ou une bibliothèque, en gérant les **dépendances** entre fichiers sources.

Objectif	Bénéfice
Ne recompiler que le strict nécessaire	Construction et reconstruction rapides
Recompiler tout ce qui est nécessaire	Si un <code>.h</code> est modifié, tous les <code>.c</code> qui l'incluent sont retraités
Mémoriser des options et commandes	Variables <code>CFLAGS</code> , <code>LDFLAGS</code> , choix du compilateur (<code>CC=</code>)

C'est le rôle habituel de `./configure` de produire les Makefiles à partir de templates, adaptés à la machine cible

 **GNU Make**

C'est **GNU Make** qui est utilisé sur la plupart des distributions Linux.

La commande `make`

Sans l'option `-f fichier` ni `-C répertoire`, `make` cherche son fichier dans l'ordre : `GNUmakefile`, puis `makefile`, puis `Makefile` dans le répertoire courant.

```
$ make
make: *** Pas de cibles spécifiées et aucun makefile n'a été trouvé. Arrêt.

: > makefile
make
make: *** Pas de cibles. Arrêt.
```

Un makefile minimal : 2 cibles, ":" en fin de label

```
$ printf 'toto:\nall:\n' > makefile
make all
make: Rien à faire pour « all ».
```

Sans argument, "make" exécute la PREMIÈRE cible définie

```
$ make
make: Rien à faire pour « toto ».
```

🔗 Conventions

Conventions : la première cible est très souvent `all`.

Cibles usuelles : `clean`, `install`, `check`, `dist`.

Format d'un Makefile

Doc de référence : [GNU Make Manual](#)

```
≡ # Variable : VAR=valeur , utilisation : $(VAR)
CC      = gcc
CFLAGS  = -O2 -Wall
EXEC    = monprog
OBJS    = main.o utils.o

# cible: dépendances
#      <TAB>commandes      ... l'indentation DOIT être une TABULATION
all: $(EXEC)
$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c                # règle implicite pour passer d'un .c à un .o
        $(CC) $(CFLAGS) -c $< -o $@

clean:
        @rm -f $(OBJS) $(EXEC)    # le "@" cache l'affichage de la commande
```

Symbole	Sens
<code>\$@</code>	nom de la cible courante
<code>\$^</code>	toutes les dépendances
<code>\$<</code>	première dépendance

Liaison statique vs dynamique, ELF, ld.so

Tout exécutable Linux peut utiliser des bibliothèques ("*bibliothèques*") :

- **Statique** (option `-static` à `ld`) : le code des bibliothèques est **inclus** dans le binaire. Fichier plus gros, mais autonome. Elle demande les versions statiques des bibliothèques (`glibc-static`).
- **Dynamique** (par défaut) : le binaire est plus petit, mais il faut **retrouver les** `.so` à l'exécution. C'est le mécanisme des **bibliothèques partagées**.

Le chargeur (`ld.so`, devenu `ld-linux.so*`) trouve et charge les `.so` requises, puis lance le programme.

Chargeur	Format binaire	Usage
<code>/lib/ld.so</code>	<code>a.out</code> (historique)	Plus utilisé
<code>/lib/ld-linux.so.1</code>	ELF, ancienne glibc1 (= libc5)	Plus utilisé
<code>/lib64/ld-linux-x86-64.so.2</code>	ELF, glibc2	x86_64 actuel

ELF

ELF = *Executable and Linkable Format*, le format universel sous Linux.
Comme pour les exécutables, le système **ne cherche pas** les bibliothèques dans le répertoire courant.

Ordre de recherche des bibliothèques

Le chargeur `ld-linux-x86-64.so.2` suit l'ordre **a, b, d, e** (ou **b, c, d, e** si `DT_RUNPATH` est présent), et s'arrête à la première trouvée :

#	Source	Note
a	<code>DT_RPATH</code> (attribut ELF)	Ignoré si <code>DT_RUNPATH</code> est défini (<i>obsolète</i>)
b	Variable <code>LD_LIBRARY_PATH</code>	Ignorée si le binaire est SetUID ou SetGID
c	<code>DT_RUNPATH</code> (attribut ELF)	En remplacement de <code>DT_RPATH</code>
d	<code>/etc/ld.so.cache</code>	Construit par <code>ldconfig</code>
e	<code>/lib</code> , <code>/lib64</code> , puis <code>/usr/lib</code> , <code>/usr/lib64</code>	Sauf si le binaire a été lié avec <code>-z nodeflib</code>

Embarquer un RUNPATH dans un binaire à l'édition de liens

```
$ gcc -o monprog main.o -L./mylib -lfoo -Wl,--enable-new-dtags,-rpath,'$ORIGIN/../lib'
```

💡 `ORIGIN\ORIGIN`` = chemin du binaire => RUNPATH portable

🔧 `RPATH` par défaut sur RHEL et dérivées

Sans `--enable-new-dtags`, `-rpath` écrit un `DT_RPATH` avec l'éditeur de liens de RHEL et dérivées (`readelf -d` le montre).

La commande `ldd`

`ldd` liste les bibliothèques qu'un exécutable va utiliser, avec leur résolution effective :

```
$ ldd /bin/whoami
    linux-vdso.so.1 (0x00007f1c5a9a2000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f1c5a600000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f1c5a9a4000)
```

❗ `ldd` peut exécuter du code

`ldd` peut exécuter du code du binaire interrogé : à éviter sur un binaire non fiable.

`objdump -p binaire | grep NEEDED` ou `readelf -d binaire` n'exécutent rien.

ldconfig et /etc/ld.so.conf

Relancer `ldconfig` à chaque modification de `ld.so.conf` ou d'un fichier inclus.

`ldconfig` configure le **cache** que `ld-linux.so` utilise.

```
# ldconfig          # reconstruit /etc/ld.so.cache à partir de la configuration
ldconfig -p        # liste les bibliothèques connues du cache
ldconfig -v        # mode verbeux, montre les répertoires scannés
ldconfig -N -X     # ne reconstruit ni le cache ni les liens (diagnostic)
```

Configuration :

```
$ cat /etc/ld.so.conf
include /etc/ld.so.conf.d/*.conf
```

Le mot-clé `include` permet l'inclusion de fichiers.

Ajouter un répertoire de bibliothèques :

```
# echo /opt/foo/lib > /etc/ld.so.conf.d/foo.conf
ldconfig
ldconfig -p | grep libfoo
```

Le séparateur dans `ld.so.conf` est libre : espace, tabulation, virgule, **saut de ligne** (généralement utilisé), deux-points.

LD_PRELOAD, LD_TRACE_LOADED_OBJECTS, vDSO

Forcer le pré-chargement d'une bibliothèque

Objectif : debug, instrumentation, hooks

```
# LD_PRELOAD=/chemin/libhook.so ./monprog          # par variable
echo /chemin/libhook.so > /etc/ld.so.preload      # par fichier (système)
```

Lister les dépendances sans exécuter le programme

🔗 Alternative à ldd

C'est ce que ldd fait, mais en plus sûr

```
$ LD_TRACE_LOADED_OBJECTS=1 /bin/whoami
    linux-vdso.so.1 (0x00007fff8efff000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fa39f2f3000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fa39f686000)
```

linux-vdso.so.1 (anciennement **linux-gate.so.1**) n'est pas un fichier sur disque : c'est une **bibliothèque virtuelle (vDSO)** exposée par le noyau dans l'espace d'adressage de chaque processus pour accélérer certains appels système (**gettimeofday**, **clock_gettime**...).

Doc : **man ld.so**, **man vdso(7)**.

Bonnes pratiques

- Paquet du dépôt d'abord, compilation en dernier recours.
- Installer dans `/usr/local` ou `/opt` : hors de l'arborescence gérée par `rpm`.
- Compiler avec un utilisateur ordinaire, `make install` seul en root.
- Lire `README` et `INSTALL` avant `./configure`.
- Vérifier la signature ou la somme de l'archive.
- Contrôler les dépendances avec `ldd` sur un binaire de confiance (`ldd` exécute le programme inspecté sur certains systèmes).
- Un `.rpm` plutôt qu'un `make install` pour un déploiement multiple.

MODULE 5

Maîtriser la production de packages RPM

De quoi parle-t-on

Après la compilation, produire un vrai paquet RPM installable et distribuable : `rpmbuild`, le fichier SPEC, puis un dépôt local pour le servir.

Au programme

- `rpmbuild` et les stages de construction,
- le fichier SPEC,
- anatomie d'un dépôt RPM, `createrepo_c`.

La commande `rpmbuild`

`rpmbuild` construit un paquet RPM à partir d'une archive source `tar` et d'un fichier de description (le **SPEC**). Elle gère :

- pré-requis (`BuildRequires`),
- extraction des sources et patches,
- `configure` + `make` + `make install` dans un répertoire isolé,
- opérations de pré/post (dés)installation,
- méta-données (description, dépendances, licence...).

```
# dnf install rpm-build rpmdevtools           # rpmbuild, puis rpmdev-setuptree
```

```
$ rpmdev-setuptree                           # crée l'arborescence dans ~/rpmbuild
```

💡 Construire sans root

Pas besoin d'être `root` pour construire un paquet, **c'est même recommandé** : construire en utilisateur non privilégié évite que la phase `%install` ne déborde sur le système réel.

[RPM Packaging Guide](#) | Red Hat : [Packaging and distributing software \(RHEL 10\)](#)

Modes d'invocation et stages

Deux entrées possibles : `-t` ou `-b`

```
$ rpmbuild -tX archive.tar.gz # archive doit contenir un .spec
rpmbuild -bX fichier.spec # archive correspondante attendue dans SOURCES/
```

`X` est la lettre du stage cible (chaque stage inclut les précédents) :

Stage	-bX	Action
p	-bp	%prep : extraction, patches
c	-bc	+ %build : configure, make
i	-bi	+ %install : make install dans BUILDROOT/
l	-bl	+ contrôle de la section %files
b	-bb	+ produit le ou les RPM binaires
s	-bs	produit le RPM source (SRPM) uniquement
a	-ba	produit binaire ET source (cas standard)

🔗 Feuille de route

Roadmap : `rpmbuild -bp foo.spec`, vérifier l'arbre extrait, `rpmbuild -ba foo.spec`.

Le fichier SPEC : l'en-tête

Le fichier `foo.spec` décrit le paquet : ses métadonnées, puis les opérations à mener.

```
≡ Name:          foo
   Version:       1.2.3
   Release:       1%{?dist}
   Summary:       Outil d'analyse foo
   License:       GPL-3.0-or-later
   URL:           https://exemple.org/foo
   Source0:       %{url}/releases/%{name}-%{version}.tar.xz
   BuildRequires: gcc, make, openssl-devel

%description
foo analyse les paquets et produit un rapport.
```

- `Name`, `Version` et `Release` forment le nom du paquet : `foo-1.2.3-1.el10`.
- `Source0` désigne l'archive des sources, `BuildRequires` les paquets nécessaires à la compilation.

Le fichier SPEC : étapes et fichiers

```
%prep
%autosetup

%build
%configure
%make_build

%install
%make_install

%files
%license LICENSE
%doc README.md
%{_bindir}/foo
%{_mandir}/man1/foo.1*

%changelog
* Wed Mar 11 2026 François Micaux <fmicaux@actilis.net> - 1.2.3-1
- Première version packagée
```

- `%prep` extrait les sources, `%build` compile, `%install` copie le résultat dans un répertoire temporaire.
- `%files` liste ce que le paquet contient.

Les macros du SPEC

Macro	Équivaut à
%autosetup	extraire Source0 dans %{name}-%{version} et appliquer les patches
%configure	./configure avec les chemins standards (--prefix=/usr...)
%make_build	make avec les options de construction du système
%make_install	make install DESTDIR=%{buildroot}
%{_bindir}	/usr/bin (rpm --eval '%{_bindir}')

Ce qu'on n'écrit plus dans un SPEC

Un %post qui lance ldconfig est devenu inutile : glibc met à jour le cache des bibliothèques à chaque transaction. Requires: openssl est aussi superflu : RPM détecte lui-même les dépendances de bibliothèques.

Commentaires et macros

RPM développe les macros jusque dans les commentaires. Un commentaire s'écrit sur sa propre ligne, et %% affiche un %.

Arborescence `rpmbuild` et macros personnelles

`rpmbuild` travaille par défaut dans `~/rpmbuild/` (depuis RHEL 6) :

```
≡ ~/rpmbuild/
├── SOURCES/      # archives .tar, patches, icônes
├── SPECS/        # fichiers .spec
├── BUILD/        # lieu de %prep et %build
├── BUILDROOT/    # lieu de %install (nettoyé après build)
├── RPMS/         # paquets binaires produits (par architecture)
└── SRPMS/        # paquets source produits (.src.rpm)
```

Déplacer cette arborescence

🔧 **Priorité des macros**

`~/.rpmmacros` prédomine sur les fichiers système

```
$ echo '%_topdir /data/build/rpmbuild' > ~/.rpmmacros
mkdir -p /data/build/rpmbuild/{SOURCES,SPECS,BUILD,BUILDROOT,RPMS,SRPMS}
```

Lister les macros disponibles

```
$ rpm --showrc
rpm --eval '%{_bindir}'          # affiche /usr/bin
```

💡 **Build reproductible**

Pour un build reproductible, **mock** construit dans un chroot dédié (paquet d'EPEL).

Anatomie d'un dépôt RPM

Un dépôt est un répertoire **servi en HTTP, HTTPS, FTP ou NFS**, contenant :

- les paquets (par convention dans `Packages/`, mais le nom est libre),
- un répertoire `repodata/` avec les méta-données XML.

Name	Last modified	Size	Description
 Parent Directory		-	
 EFI/	2020-06-08 21:26	-	
 EULA	2020-06-08 21:17	298	
 GPL	2020-06-08 21:17	18K	
 Packages/	2020-09-30 01:37	-	
 extra_files.json	2020-06-08 21:17	487	
 images/	2020-06-09 21:04	-	
 isolinux/	2020-06-08 21:37	-	
 media.repo	2020-06-08 22:07	100	
 repodata/	2020-09-18 16:28	-	

 Ce que télécharge dnf

Le client `dnf` télécharge d'abord `repomd.xml` (*Repository Meta Data*), puis les fichiers qu'il pointe (à commencer par `primary.xml`, compressé), si le cache local est expiré.

Contenu de repodata/

Fichier	Rôle
repomd.xml	Index : pointe les autres fichiers (hash, taille)
primary.xml.gz	Description des paquets : nom, version, dépendances, chemin relatif
filelists.xml.gz	Liste des fichiers de chaque paquet (<i>dnf provides</i>)
other.xml.gz	Changelogs
comps.xml	Description multilingue des groupes (<i>catégories à l'installation</i>)
updateinfo.xml.gz	Avis de correctifs (<i>dnf updateinfo</i>)
repomd.xml.asc	Signature détachée GPG du repomd.xml

i Compression et bases SQLite

Les dépôts AlmaLinux publient des fichiers `.xml.gz` (et `.sqlite.bz2`). `createrepo_c` propose `--compress-type` (`gz`, `xz`, `zstd`, `bz2`, `zck`). Les bases SQLite sont **dépréciées** : `--database` les produit encore, `--no-database` les écarte. Sans option, `createrepo_c` 1.1.2 produit des `.xml.zst`.

Générer son propre dépôt avec `createrepo_c`

1. Installer l'outil

```
# dnf install createrepo_c
```

2. Préparer le répertoire

```
# mkdir -p /srv/repo/myrepo/Packages  
cp *.rpm /srv/repo/myrepo/Packages/
```

3. Générer les méta-données

```
# createrepo_c /srv/repo/myrepo/
```

4. Mise à jour incrémentale (n'analyse que les RPM nouveaux/modifiés)

```
# createrepo_c --update /srv/repo/myrepo/
```

Signer le dépôt

Signer le repomd.xml (optionnel mais recommandé)

```
# gpg --detach-sign --armor /srv/repo/myrepo/repdata/repomd.xml
```

⚠ Signer de nouveau après chaque génération

Dès qu'un paquet est ajouté ou modifié, `createrepo_c` réécrit `repomd.xml` : l'ancien `repomd.xml.asc` n'est plus valable et `dnf` refuse le dépôt (`Bad PGP signature`).

i createrepo_c

`createrepo_c` est la réimplémentation C de l'ancien `createrepo` (Python, abandonné).

Code : github.com/rpm-software-management/createrepo_c.

Outils associés : `modifyrepo_c` (modifier `repomd.xml`), `mergerepo_c` (fusionner plusieurs dépôts).

Config côté client

```
# Côté client : /etc/yum.repos.d/myrepo.repo
[myrepo]
name=Mon dépôt local
baseurl=https://repo.example.org/myrepo/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://repo.example.org/RPM-GPG-KEY-myrepo
```

`gpgcheck` contrôle la signature des **paquets**, `repo_gpgcheck` celle des **méta-données** (`repomd.xml.asc`).

✎ Signer les paquets

Les paquets se signent à part avec `rpmsign --addsign` (paquet `rpm-sign`) : sans signature, un client en `gpgcheck=1` les refuse.

Bonnes pratiques

- Un `.spec` versionné, construit par `rpmbuild` en utilisateur ordinaire.
- `rpmlint` avant diffusion.
- Signer les paquets (`rpmsign`) et les méta-données du dépôt.
- `gpgcheck=1` et `repo_gpgcheck=1` côté client.
- Relancer `createrepo_c` après chaque ajout ou retrait de paquet.
- Dépôt servi en HTTPS, en lecture seule.
- Numéros de version et `Release` incrémentés à chaque reconstruction.

CHAPITRE 3

Filesystems et unités de stockage

MODULE 6

Stockage, Swap, Systèmes de fichiers

De quoi parle-t-on

Du disque physique au volume monté : identifier les block-devices, les partitionner, choisir et régler un système de fichiers, puis passer à RAID et LVM pour plus de souplesse.

Au programme

- block-devices, partitionnement MBR/GPT,
- systèmes de fichiers : ext4 vs XFS, montage, `/etc/fstab`,
- RAID logiciel et LVM : concepts, snapshots, cache.

Block devices et interfaces disques

Disques, partitions, CD/DVD, clés USB et cartes mémoire sont des **block-devices**.

Interface	Pilote	Débit	Naming
SATA (Serial ATA)	natif	1.0 = 150 Mo/s, 2.0 = 300, 3.0 = 600	/dev/sd?
SCSI / SAS (Fiber Channel)	natif	SAS = 600 Mo/s	/dev/sd?
USB Mass Storage	usb-storage	dépend de l'USB	/dev/sd?
VirtIO (KVM / hôte virtualisé)	virtio-blk	performant	/dev/vd?
NVMe (SSD sur bus PCIe)	nvme	plusieurs Go/s	/dev/nvme0n1, partitions /dev/nvme0n1p1

 Limites en taille

Limites en taille (*bloc 4 ko, théoriques*) : par conception ext4 = 1 EiB, XFS = 8 EiB, Btrfs = 16 EiB.

Identifier les disques en place

Ce que le sous-système SCSI connaît

```
$ cat /proc/scsi/scsi
```

Disques / contrôleurs vus côté SCSI

```
$ lsscsi
```

Disques NVMe (*paquet nvme-cli*)

```
$ nvme list
```

Disques physiques (sd*), logiques (dm-*), partitions

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

Arbre block-devices + points de montage (le plus lisible)

```
$ lsblk
```

Détection à chaud (hot-plug)

Ajout d'un disque détecté par le BIOS / hyperviseur, sans redémarrage

```
# scsi-rescan -a # paquet sg3_utils  
echo "- - -" > /sys/class/scsi_host/host0/scan # même chose, sans paquet  
scsi-rescan -r # retrait à chaud d'un disque
```

MBR et table de partitions



- **Partitions primaires** : 4 maximum (`sda1` à `sda4`).
- **Partition étendue** (type **0x05**) : occupe l'une des 4 primaires, contient des partitions **logiques** chaînées par des **EBR** (*Extended Boot Record*).
- **Logiques** : `sda5` à `sda15` (15 partitions max par disque sous Linux pour les SATA/SCSI).

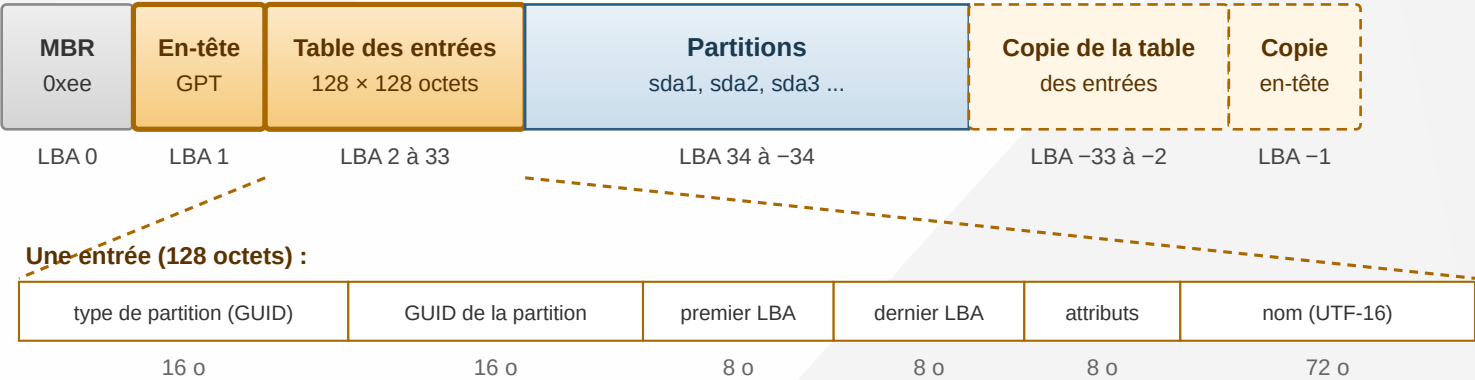
Limite MBR

Limite MBR : **2 To** par partition et par disque. Au-delà, passer à **GPT** (*GUID Partition Table*), conteneur visible comme une partition `0xee` sous MBR.

GPT : la table de partitions actuelle

GPT (GUID Partition Table) : standard de l'UEFI, requis au-delà de 2 Tio.

GPT : la table sur le disque



- Un **MBR de protection** (0xee) occupe le 1er secteur : les outils anciens ne voient pas un disque vide.
- L'**en-tête GPT** est au 2e secteur, avec une **copie à la fin du disque** : une table abîmée se répare.
- **128 partitions** par défaut, chacune avec un **GUID**, un **type** lisible, un **nom** et un **UUID** propre (PARTUUID).

```
$ lsblk -o NAME,SIZE,PARTYPENAME,PARTUUID
```

Où est la table ? `fdisk -l` indique `Disklabel type: gpt` ou `dos` (MBR), `parted /dev/sda print` indique `Partition Table: gpt` ou `msdos`.

Que faire avec une partition ?

Le **type** de la partition oriente son usage. Mais elle reste avant tout un **block-device**.

Type MBR	Type GPT	Usage	Outils de préparation	Activation
0x82	Linux swap	swap	mkswap	swapon
0x83	Linux filesystem	système de fichiers	mkfs, mkfs.ext4, mkfs.xfs...	mount
0x8e	Linux LVM	volume physique LVM	pvcreate	vgcreate, puis lvcreate
0xfd	Linux RAID	membre RAID logiciel	mdadm --create	mdadm --assemble
0xef	EFI System	partition ESP (/boot/efi)	mkfs.vfat	mount
	BIOS boot	amorce GRUB en BIOS sur GPT	aucune	aucune
0xee	(protection)	conteneur GPT	fdisk, parted	comme une partition classique

Un volume RAID peut devenir un PV LVM, un LV peut être formaté en swap, etc.

Outils de partitionnement

Outil	MBR	GPT	Mode	Notes
<code>fdisk</code>	✓	✓	interactif, scriptable via stdin	Le standard
<code>cfdisk</code>	✓	✓	interactif, ncurses	Pas scriptable
<code>sfdisk</code>	✓	✓	non interactif	Idéal pour scripts
<code>parted</code>	✓	✓	non interactif	Scripts, redimensionnement
<code>gdisk</code>		✓	équivalent fdisk spécialisé GPT	conversion MBR ⇔ GPT, réparation, MBR hybride (<i>paquet d'EPEL</i>)
<code>sgdisk</code>		✓	non interactif	idéal pour les scripts (<i>paquet <code>gdisk</code>, EPEL</i>)

Lister & rechargement de la table de partitions

```
# fdisk -l /dev/sda          # ou : parted /dev/sda print
partprobe /dev/sda          # demander la relecture par le noyau
hdparm -z /dev/sda          # équivalent ; sinon, reboot
```

 Relecture de la table

Relecture immédiate possible uniquement si **aucune partition du disque n'est en cours d'utilisation** (*non montée, swap inactif...*).

Panorama des systèmes de fichiers

VFS (*Virtual File System*, 1992) est la couche d'abstraction qui permet de monter des FS de types différents de manière transparente.

```
$ grep -v nodev /proc/filesystems # types reconnus par le noyau courant
```

Famille	Journal	Redim. à chaud	Forces
ext4 (2009)	oui	agrandir seulement	défaut historique Linux, jusqu'à 1 EiB
XFS (SGI)	oui	agrandir seulement	idéal pour gros fichiers / gros FS, défaut RedHat
Btrfs	CoW	oui	snapshots, compression, checksums (<i>AlmaLinux, pas RHEL</i>)
vfat / exfat / iso9660 / udf	n/a	n/a	interop. Windows / médias
NFS / SMB-CIFS	n/a	n/a	systèmes de fichiers réseau

i Ce que le noyau de RHEL 10 ne contient plus

`ext2` et `ext3` sont servis par le pilote **ext4** (`mkfs.ext2` et `mkfs.ext3` existent encore). `JFS`, `ReiserFS` et NTFS ne sont pas dans le noyau livré.

XFS face à ext4 : deux philosophies

🔗 Ni l'un ni l'autre

Les deux sont journalisés, matures et s'agrandissent à chaud.
⇒ aucun n'est un mauvais choix.

Critère	ext4	XFS
Défaut distribution	Debian, Ubuntu	RedHat et dérivées (7+)
Architecture	extents + journal	groupes d'allocation parallèles
Réduction (<i>shrink</i>)	oui , à froid (<code>resize2fs</code>)	impossible (<i>croissance seule</i>)
Agrandissement à chaud	oui (<code>fsadm</code> / <code>resize2fs</code>)	oui (<code>xfs_growfs</code>)
Parallélisme multi-cœurs	correct	excellent (<i>AG concurrents</i>)
Gros fichiers / gros FS	bon	très bon
Multitude de petits fichiers	bon , faible surcoût	surcoût métadonnées
reflink (<i>CoW</i>), quotas projet	non	oui
Réparation hors-ligne	<code>fsck.ext4</code>	<code>xfs_repair</code> , rapide

XFS ou ext4 : quand choisir l'un ou l'autre ?

⚠ XFS ne se réduit pas

La différence décisive : XFS ne se réduit **jamais**. Dès qu'une réduction de volume est envisageable, ext4 s'impose.

Privilégier ext4

- Volume qu'on pourra devoir **réduire** un jour (`resize2fs`, à froid).
- Hôtes **Debian / Ubuntu** : on suit le défaut de la distribution.
- `/boot`, petites partitions, embarqué : maturité, faible surcoût.
- Charge généraliste, **multitude de petits fichiers** (`/home`, *mail*).

Privilégier XFS

- **Gros volumes / gros fichiers** : vidéo, imagerie, sauvegardes, Po.
- Bases de données, serveurs **très parallèles** (*nombreux cœurs*).
- Hôtes **RedHat et dérivées** : défaut et support éditeur.
- Besoin de **reflinks** (*clones légers*) ou de **quotas projet**.

💡 Règle simple

Règle simple : suivre le défaut de la distribution, et ne dévier que pour une raison précise :

réduction à prévoir ⇒ **ext4**

très gros volumes très sollicités ⇒ **XFS**

Créer et vérifier un système de fichiers

Formater : appelle mkfs.ext4 (idem xfs, btrfs...)

```
# mkfs -t ext4 /dev/vg/lv
```

Options spécifiques : voir mkfs.(8)

```
# mkfs.xfs -L data /dev/vg/data
```

Repérer les blocs défectueux

```
# badblocks -o bad.lst /dev/sdb1
```

En tenir compte au formatage

```
# mkfs.ext4 -l bad.lst /dev/sdb1
```

Vérifier / réparer (selon le type)

```
# fsck -y /dev/vg/lv
```

 fsck et XFS

`fsck` ne fait rien sur XFS : on y utilise `xfs_repair`, démonté.

Redimensionner un système de fichiers

fsadm (générique, ext et xfs)

```
# fsadm resize /dev/vg/data 50G
```

✓ fsadm

Remplace `resize2fs` / `xfs_growfs` et les équivalents en `*ck` :

```
fsadm check /dev/vg/data
```

Agrandir, réduire : ce que permet chaque type

	Agrandir	Réduire
ext4	à chaud, <code>resize2fs</code>	à froid, démonté
XFS	à chaud, <code>xfs_growfs</code>	impossible

Réduire un FS demande de réduire **d'abord** le FS, puis le block-device (partition, LV).

Tuning d'un système de fichiers ext

Lister les options en place

```
# tune2fs -l /dev/vg/data
```

Poser un label

```
# tune2fs -L data /dev/vg/data
```

Régénérer un UUID

```
# tune2fs -U random /dev/vg/data
```

Désactiver fsck préventif (intervalle / nb. mounts)

```
# tune2fs -i 0 -c 0 /dev/vg/data
```

Réserver 1 % à root au lieu de 5 % par défaut

```
# tune2fs -m 1 /dev/vg/data
```

Monter et démonter un système de fichiers

Montage : le point de montage doit exister

```
# mount /dev/vg_data/lv_www /data/www  
mount -t xfs -o noatime,nodiratime /dev/sdb1 /data
```

⚠ Fichiers masqués

Les fichiers présents dans le point de montage **avant** le `mount` sont **masqués** jusqu'au démontage.

Changer une option sans démonter

```
# mount -o remount,ro /
```

Lister les montages en cours

```
$ mount  
cat /proc/mounts
```

Démontage

```
# umount /data/www      # ou : umount /dev/vg_data/lv_www  
umount -a               # tous les FS montés (sauf ceux occupés et les FS système)
```

🔧 Démontage refusé

Refus de démontage si un processus tient la ressource : voir `lsof` / `fuser`.

Le fichier /etc/fstab

Une ligne par filesystem, 6 champs (les 2 derniers optionnels, valeur 0 par défaut) :

# <device>	<mountpoint>	<type>	<options>	<dump>	<pass>
UUID=653a55dc-8342-440c-85ac-...	/boot	xfs	defaults	0	0
LABEL=data	/data	xfs	defaults,noatime	0	0
/dev/vg_data/lv_backup	/data/backup	ext4	defaults,noauto	0	0
/dev/mapper/vg00-lv_swap	none	swap	defaults	0	0

Champ	Rôle
device	/dev/..., UUID=..., LABEL=... (préférés pour stabilité)
options	defaults, noatime, usrquota, grpquota, noauto, ro, rw, nofail (ne bloque pas le boot si absent), _netdev (attendre le réseau), x-systemd.automount...
dump	1 si à inclure dans une sauvegarde dump incrémentale
pass	ordre fsck au boot : 1 pour /, 2 pour les autres, 0 = pas de contrôle (0 sur XFS)

🔧 fstab et systemd

Au boot, systemd-fstab-generator transforme chaque ligne en unité .mount et monte les FS sans noauto.

mount -a rejoue ce montage à la demande. Après une modification de /etc/fstab :
systemctl daemon-reload.

Identifier le processus qui bloque un démontage

```
# umount /data
umount: /data: target is busy.
```

Liste concise : PID, USER, ACCESS

```
# fuser -v /data
                USER    PID    ACCESS COMMAND
/data:         root 25401  ..c..  bash           # 'c' = répertoire courant
```

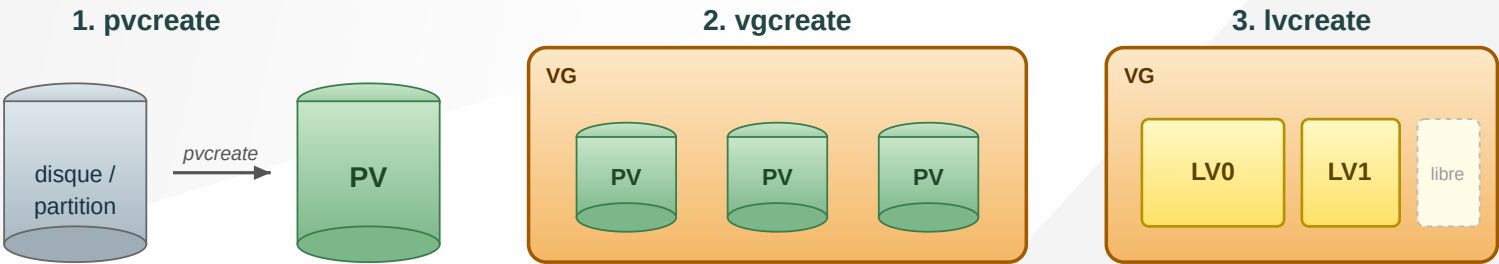
Tous les fichiers ouverts sous /data (`lsuf -p PID` : ceux d'un processus)

```
# lsuf /data
```

Code <code>fuser</code>	Signification
c	répertoire courant (<code>cwd</code>)
e	exécutable en cours
f	fichier ouvert
r	racine (<code>chroot</code>)
m	mmap / bibliothèque chargée

```
! fuser -km
fuser -km /data envoie un SIGKILL à tous les processus utilisant /data.
```

LVM : concepts



PV (*Physical Volume*) : tout block-device, de préférence partitionné en type **Linux LVM** (*0x8e en MBR*) : disque, partition, volume RAID, LUN iSCSI / FC.

VG (*Volume Group*) : agrège un ou plusieurs PV, sans limite de nombre par défaut. Un PV n'appartient qu'à un seul VG.

LV (*Logical Volume*) : tranche d'un VG. C'est un **block-device** classique : on le formate, on le monte, on le partage.

Unité d'allocation = **extent** (*PE côté PV, LE côté LV, taille identique*).

Nommage d'un volume logique

Vision Admin	Vision Noyau
/dev/groupe/logique	/dev/mapper/groupe-logique
/dev/vg00/lvroot	/dev/mapper/vg00-lvroot
/dev/vg-data/lv-www	/dev/mapper/vg--data-lv--www

LVM : du disque au système monté

0. Préparer la partition (type `lvm`)

```
# fdisk /dev/sdb...
```

2. Structurer le volume physique

```
# pvcreate /dev/sdb1
```

3. Créer le VG (peut accepter plusieurs PV)

```
# vgcreate vg_data /dev/sdb1 /dev/sdc1
```

4. Découper un LV

```
# lvcreate -L 20G -n lv_www vg_data
```

5. Formater puis monter

```
# mkfs.xfs /dev/vg_data/lv_www  
mkdir -p /data/www && mount /dev/vg_data/lv_www /data/www
```

Chemins d'un LV

Le LV est accessible via `/dev/vg_data/lv_www` ou `/dev/mapper/vg_data-lv_www`.

LVM : maintenance courante

Lister / inspecter

```
# pvs ; pvscan ; pvdisplay /dev/sdb1
# vgs ; vgscan ; vgdisplay vg_data
# lvs ; lvscan ; lvdisplay /dev/vg_data/lv_www
```

Étendre un LV (et le filesystem) à chaud

```
# lvextend -L +5G /dev/vg_data/lv_www # ou : -l +100%FREE
# lvextend -L +5G --resizefs /dev/vg_data/lv_www # étend FS dans la foulée
```

Étendre / réduire un VG, déplacer les données d'un PV

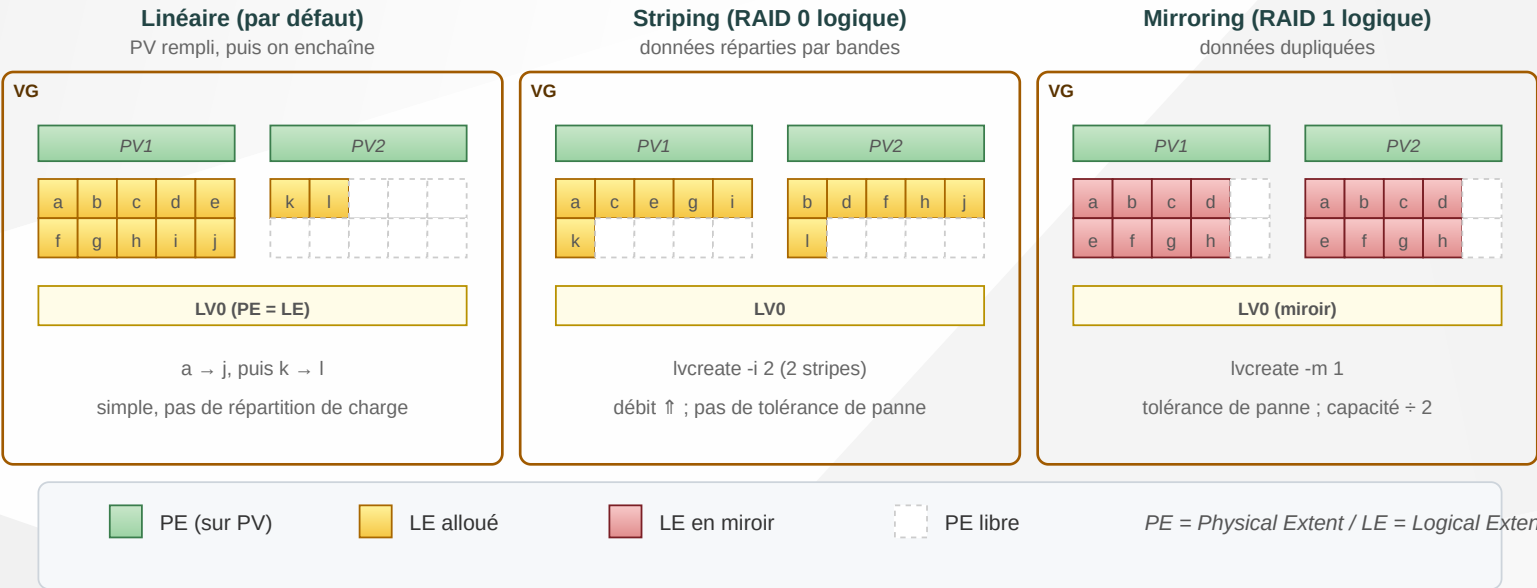
```
# vgextend vg_data /dev/sdd1 # ajoute un PV au VG
# pvmove /dev/sdc1 # libère le PV en redistribuant les LE
# vgreduce vg_data /dev/sdc1 # le retire du VG
# pvremove /dev/sdc1 # le démarque comme PV
```

! Réduire un LV

Réduire d'abord le filesystem (`fsadm resize`), puis `lvreduce`. Dans l'autre ordre, des données sont perdues. XFS ne se réduit pas.

Sur un XFS agrandi à chaud, `lvreduce --resizefs` peut réduire le volume sans réduire le système de fichiers : il ne se monte plus ensuite (constaté sur AlmaLinux 10.0, `lvm2` 2.03.28).

Modes d'agrégation à la création d'un LV



```
# lvcreate -L 10G -n lv_linear vg_data # mode par défaut
lvcreate -L 10G -i 2 -I 64 -n lv_stripe vg_data # stripping : 2 stripes, chunk 64k
lvcreate -L 10G -m 1 -n lv_mirror vg_data # miroir : 2 copies
```

⚠ Mirroring LVM

Le mirroring LVM est **un mécanisme de copie**, pas un substitut à `mdadm` pour la sécurité du disque système.

La sécurité d'un VG repose plutôt sur des **PV qui sont déjà des matrices RAID**.

Snapshots LVM

Capturer l'état d'un LV à un instant T, sans interrompre le service.

1. Créer le snapshot (réserver l'espace pour les écritures pendant sa durée de vie)

```
# lvcreate -s -L 2G -n snap_www /dev/vg_data/lv_www
```

2. Le snapshot est un LV accessible : sauvegarde fiable du LV à l'instant T

```
# mount -o ro,nouuid /dev/vg_data/snap_www /mnt/snap # nouuid : requis sur XFS
tar -C /mnt/snap -czf /backup/www.tgz .
umount /mnt/snap
```

3a. Le supprimer après usage

```
# lvremove /dev/vg_data/snap_www
```

OU

3b. le « fusionner » dans l'origine = retour arrière (rollback)

```
# lvconvert --merge /dev/vg_data/snap_www
```

Cas d'usage :

- **Sauvegarde cohérente** (*arrêt d'un service le temps du snapshot, puis backup long*).
- **Sécurité avant migration ou montée de version** : si KO, `lvconvert --merge`.

⚠ Snapshot saturé

Un snapshot saturé devient **invalide** (le LV d'origine, lui, reste intact).

Cache LVM : un SSD devant un disque mécanique

Cache LVM

Combine la capacité d'un HDD à la latence d'un SSD, sans toucher au LV applicatif.

Ici, rootvg contient 2 PV : /dev/sda (HDD) et /dev/sdb2 (SSD), et un LV (volapp)

1. Sur le SSD, créer les deux LV qui formeront le cache pool

```
# lvcreate -L 10G -n vol_CacheData rootvg /dev/sdb2      # données
# lvcreate -L 10M -n vol_CacheMeta rootvg /dev/sdb2     # métadonnées (~ 1/1000)
```

2. Combiner data + meta en cache pool

```
# lvconvert --type cache-pool --poolmetadata rootvg/vol_CacheMeta rootvg/vol_CacheData
```

3. Lier le cache pool au LV « lent » volapp

```
# lvconvert --type cache --cachepool rootvg/vol_CacheData rootvg/volapp
```

Démonter le cache (sans perte) et supprimer le cache pool

```
# lvconvert --uncache rootvg/vol
```

Doc : `man lvmcache(7)`, qui décrit aussi la forme simplifiée `--cachevol` (un seul LV rapide).

RAID : compromis sécurité / performances / coût

Agréger des disques en privilégiant deux éléments sur trois : *sécurité, perms, coût*.

Niveau	Tolérance	Lecture	Écriture	Coût (utile / brut)	Usage
0 (<i>stripe</i>)	aucune	+ + +	+ + +	100 %	perms, données jetables
1 (<i>miroir</i>)	1 disque	+ +	+	50 %	disques système, données critiques
3 / 4 (<i>parité dédiée</i>)	1 disque	+ +	+	(n-1)/n	gros volumes d'I/O séquentielles
5 (<i>parité distribuée</i>)	1 disque	+ +	+	(n-1)/n	lecture intensive
6 (<i>double parité</i>)	2 disques	+ +	--	(n-2)/n	tolérance accrue, lecture intensive
10 (<i>1+0</i>)	jusqu'à n/2	+ + +	+ +	50 %	bases de données, compromis idéal

- **Logiciel** (*noyau Linux*, `mdadm`) : flexible, sollicite le CPU.
- **Matériel** (*Smart Array, MegaRAID...*) : déchargement CPU, batterie de cache utile.

Bonnes pratiques

- Identifier le disque par `lsblk` avant toute écriture : une erreur de périphérique détruit des données.
- `UUID=` dans `/etc/fstab`, pas `/dev/sdX`.
- `mount -a` après modification de `fstab`, avant de redémarrer.
- `nofail` sur les montages non indispensables au démarrage.
- LVM : garder de l'espace libre dans le groupe de volumes.
- Un snapshot est temporaire : le supprimer après usage.
- RAID n'est pas une sauvegarde.
- Sauvegarder avant de réduire ou de redimensionner.

CHAPITRE 4

Noyau et périphériques

MODULE 7

Périphériques, modules et udev

De quoi parle-t-on

Du matériel détecté à son pilote chargé, jusqu'au fichier `/dev` qui le représente : ce module suit toute la chaîne, puis explique comment udev automatise sa gestion.

Au programme

- détecter le matériel : `lspci`, `lsusb`, résolution du bon module,
- fichiers spéciaux de `/dev`, majeur/mineur,
- udev : démon, règles, `udevadm`.

Détecter le matériel : lspci et lsusb

Lister les périphériques PCI

```
$ lspci
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller
lspci -n                # signatures brutes
00:03.0 0200: 8086:100e (rev 02)
#          CLASSE  VENDOR:DEVICE
```

 Lire la ligne

Emplacement : `00:03.0` : Bus PCI 0, Slot 3 (surement enp0s3)
Classe **0200** (Ethernet), constructeur **8086** (Intel), device **100e**.

Options utiles	Effet
<code>lspci -n</code>	identifiants numériques au lieu des noms
<code>lspci -v</code>	détails (IRQ, mémoire, ports I/O, capabilities)
<code>lspci -t</code>	vue arborescente du bus
<code>lsusb</code> / <code>lsusb -v -s bus:dev</code>	équivalent côté USB

Lister les périphériques USB

```
$ lsusb
Bus 002 Device 004: ID 12d1:1003 Huawei Technologies ... E220 HSDPA Modem
```

Les bases de noms : pci.ids et usb.ids

Les bases de données

`lspci` et `lsusb` affichent des **noms lisibles** grâce à des bases de correspondance.

Fichier	Emplacement	Source de mise à jour
pci.ids	/usr/share/hwdata/	pci-ids.ucw.cz
usb.ids	idem	linux-usb.org

```
# update-pciids # télécharge la version à jour de pci.ids
```

Structure de `pci.ids`

- **Début du fichier** : constructeurs et périphériques
 - *vendor*, puis *device* en retrait d'une tabulation.
- **Fin du fichier** :
 - classes de périphériques (`C 02` = Network controller, `00` = Ethernet...).

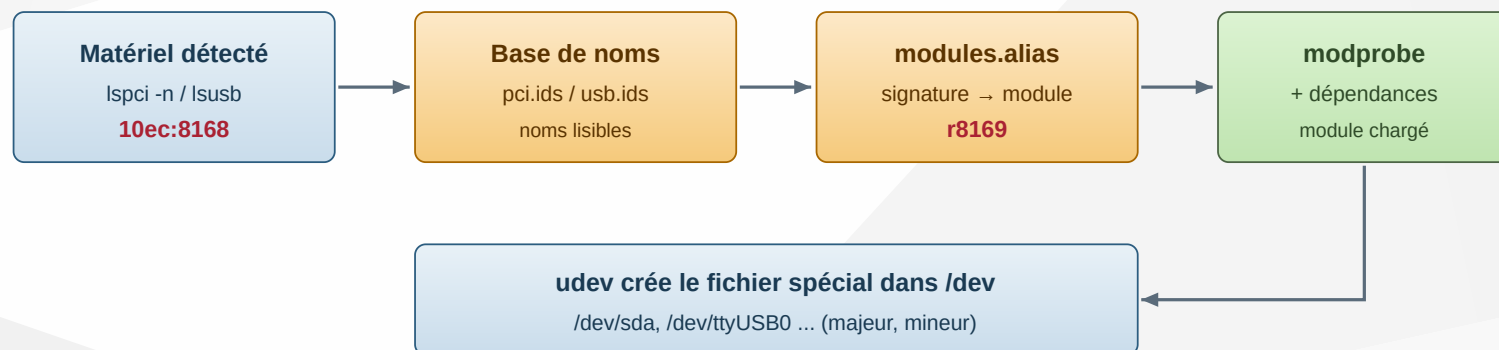
 Fichiers de noms

Ces fichiers ne servent qu'à **résoudre des noms** : ils ne pilotent rien.

Quel module pour quel périphérique ?

Les modules sont livrés avec des *maps* dans `/lib/modules/$(uname -r)/` (lien vers `/usr/lib/modules`).

Du matériel détecté au pilote chargé



```
$ grep -i 10ec.*8168 /lib/modules/$(uname -r)/modules.alias
alias pci:v000010ECd00008168sv*sd*bc*sc*i* r8169

grep -i 12d1.*1003 /lib/modules/$(uname -r)/modules.alias
alias usb:v12D1p1003d0000dc*dsc*dp*ic*isc*ip*in* usb_storage
```

modules.alias

On cherche dans `modules.alias` : la dernière colonne donne le module à charger.

Charger, lister, décharger

On ajoute ou retire un pilote **sans redémarrer** : c'est un module.

Commande	Rôle
<code>modinfo nom</code>	infos : fichier, description, auteur, alias, paramètres
<code>insmod fichier.ko</code>	charge un module (échoue si dépendances manquantes)
<code>modprobe nom</code>	charge le module et ses dépendances
<code>lsmod</code> / <code>cat /proc/modules</code>	liste les modules chargés et leur usage
<code>rmmod nom</code> / <code>modprobe -r nom</code>	décharge un module

```
$ modinfo option
description:  USB Driver for GSM modems
alias:       usb:v12D1p1003d*dc*dsc*dp*icFFiscFFipFF*
depends:      usbserial
```

 **modprobe**

`modprobe` résout les dépendances et fait *probe* le matériel dans la majorité des cas.

Configuration : `/etc/modprobe.d` et `/lib/modules`

Les modules vivent dans `/lib/modules/$(uname -r)/` :

- nommés `nom_module.ko.xz` (compressés en xz, signés), surtout sous `kernel/`.
- un module est lié à **une version** de noyau (recompilation nécessaire sinon).
- les *maps* (`modules.alias`, `modules.dep`...) y indiquent pilotes et dépendances.

Les fichiers `*.conf` de `/etc/modprobe.d/` peuvent porter options de chargement pour *kmod*.

Le fichier `/etc/modprobe.d/blacklist.conf`

```
# ignorer un module
blacklist nouveau
# passer des paramètres
options snd_hda_intel index=1
```

Le *blacklist* est utile si **plusieurs modules** peuvent gérer un même équipement.

⚠ Initramfs et chargement explicite

Un module chargé tôt au démarrage est dans l'**initramfs** : après un changement de `modprobe.d`, `dracut -f` le régénère. `blacklist` n'empêche ni un `modprobe nouveau` explicite ni un chargement par dépendance : `install nouveau /bin/false` bloque vraiment.

Fichiers spéciaux : majeur et mineur

Les périphériques sont accédés via des **fichiers spéciaux** de `/dev`.

```
$ ls -l /dev/sda1 /dev/null
brw-rw---- 1 root disk 8, 1 ... /dev/sda1
crw-rw-rw- 1 root root 1, 3 ... /dev/null
# ^type      ^majeur, ^mineur
```

Type	Sens
b	<i>block device</i> (disques...)
c	<i>character device</i> (consoles, ports série...)

À la place de la taille : **majeur** (le pilote) et **mineur** (l'instance).

- Ce sont les *adresses* normalisées entre noyau et administrateur.
 - voir `Documentation/admin-guide/devices.txt`
 - ou [Linux Assigned Names And Numbers Authority](#).
- `/proc/devices` liste les majeurs que le noyau actuel **peut** gérer.

Créer un fichier spécial : mknod

Syntaxe

```
# mknod nom Type Majeur Mineur
```

Il faut connaître le couple **majeur / mineur** attendu par le pilote (cf. `devices.txt`).

Exemple : convertisseurs USB-série (majeur 188)

```
# mknod -m 770 /dev/ttyUSB0 c 188 0  
mknod -m 770 /dev/ttyUSB1 c 188 1
```

🔗 mknod

En pratique, **udev** crée et supprime ces fichiers dynamiquement : `mknod` manuel reste l'exception.

udev et le démon udevd

udev limite `/dev` aux périphériques **réellement présents** et réagit à chaud aux évènements.

Le flux d'un évènement udev



[UDEVD] = action appliquée après traitement des règles

udevadm monitor montre les deux : [UEVENT] reçu, puis [UDEVD] émis.

- Le démon **udev** (`systemd-udev`) est averti par le noyau des ajouts/retraits.
- Il consulte les règles de `/usr/lib/udev/rules.d` et `/etc/udev/rules.d`.
- Les informations complémentaires viennent de l'arborescence `/sys`.

Les règles : clés de comparaison et d'action

Une règle = une ligne de couples **clé / valeur**. Si toutes les comparaisons passent, les actions s'exécutent.

Comparaison	Action / affectation
<code>==</code> (égalité)	<code>=</code> (affecter, réinitialise)
<code>!=</code> (différence)	<code>:=</code> (affecter et verrouiller)
	<code>+=</code> (ajouter à une liste)

Clés de comparaison	Clés de définition
<code>ACTION</code> , <code>SUBSYSTEM</code> , <code>KERNEL</code>	<code>NAME</code> (nom d'une interface réseau)
<code>DRIVER</code> , <code>DEVPATH</code>	<code>SYMLINK</code> (liens)
<code>ATTR{...}</code> , <code>ENV{...}</code>	<code>OWNER</code> , <code>GROUP</code> , <code>MODE</code>
	<code>PROGRAM</code> , <code>RUN</code>

Jokers et noms

Jokers shell autorisés dans les valeurs : `*`, `?`, `[]`. Ex. `sd[a-z]` = `sda...sdz`.
Un nœud de `/dev` ne se renomme pas : une règle lui ajoute des liens (`SYMLINK`),
`NAME` ne vaut que pour les interfaces réseau.

udevadm : observer et tester

Voir les évènements `[UEVENT]` et `[UDEV]`

```
$ udevadm monitor
```

Toutes les infos d'un périphérique

```
$ udevadm info -q all -n sda
```

Tous les attributs (attribute-walk)

```
$ udevadm info -a -p /sys/block/sda
```

Rejouer les évènements, simuler, recharger

```
# udevadm trigger          # rejouer les évènements
# udevadm test /sys/block/sda # simuler les règles sur un périphérique
# udevadm control --reload   # recharger les règles (udev les relit aussi seul)
```

```
$ udevadm info -q all -n sda1
S: disk/by-label/EOS_DIGITAL
E: ID_FS_LABEL=EOS_DIGITAL
E: ID_BUS=usb
```

 `udevadm info`

`udevadm info` fournit les attributs (`ATTR`, `ENV`) sur lesquels s'appuient nos règles.

Exemple : nommer un périphérique de façon stable

Associer un périphérique à un nom de device fixe

Problème : une clé USB s'appelle tantôt `sda1`, tantôt `sdb1`, mais son **label** ne change pas.

```
# /etc/udev/rules.d/99-montages.rules
ENV{ID_FS_LABEL}=="EOS_DIGITAL", SYMLINK+="camera"
```

Résultat : `/dev/camera` pointera vers la bonne partition, quel que soit son nom noyau.
(`/dev/disk/by-label/EOS_DIGITAL` existe déjà : *udev l'a créé sans règle personnelle.*)

Autres exemples de règles

Lancer un programme à l'ajout d'un disque

```
≡ ACTION=="add", SUBSYSTEM=="block", KERNEL=="sd[a-z]", RUN+="/usr/bin/logger -t udev ajout %k"
```

Nommer une interface réseau selon sa MAC : un fichier `.link` de systemd

```
≡ # /etc/systemd/network/10-lan.link  
[Match]  
MACAddress=08:00:27:c2:82:10  
  
[Link]  
Name=lan
```

🔗 Ordre de lecture

Les fichiers se nomment `nn-description.rules` et sont lus dans l'ordre des préfixes `nn`.

Bonnes pratiques

- `lspci -k` et `lsusb` avant de chercher un pilote.
- `modprobe` plutôt que `insmod` : les dépendances sont résolues.
- Options de module dans `/etc/modprobe.d/*.conf`, un fichier par sujet.
- Règle `udev` testée avec `udevadm test` avant d'être rechargée.
- Nommage par attribut stable (numéro de série, MAC) plutôt que par ordre de détection.
- Règles personnelles dans `/etc/udev/rules.d`.
- `blacklist` d'un module : régénérer l'initramfs si le module y figure.

MODULE 8

Le noyau : compilation et installation

De quoi parle-t-on

Le noyau standard suffit presque toujours, mais savoir en recompiler un soi-même reste utile pour comprendre le mécanisme : sources, configuration, compilation, installation et initramfs.

Au programme

- le noyau standard et son contenu dans `/boot`,
- récupérer les sources, configurer, compiler,
- installer noyau et modules, régénérer l'initramfs.

Le noyau standard

Installé par la distribution, amorcé par GRUB, stocké dans `/boot`. Le fichier noyau `vmlinuz-x.y.z` est **compressé** et accompagné de plusieurs fichiers versionnés.

Fichier	Rôle
<code>vmlinuz-x.y.z</code>	l'image bootable du noyau (compressée)
<code>initramfs-x.y.z.img</code>	ramdisk initial, construit sur place à l'installation
<code>config-x.y.z</code>	la configuration <code>.config</code> ayant servi à le compiler
<code>System.map-x.y.z</code>	correspondance adresses mémoire ↔ noms des symboles

✎ System.map

`System.map` sert à traduire les adresses en noms de fonctions (messages d'erreur du noyau, outils de débogage).

On limite les pilotes **built-in** (taille, mémoire) au profit de **modules** chargés à la demande.

✎ Pilotes pour monter /

Pour voir `/`, le noyau a besoin des pilotes du **contrôleur disque**, de la **table de partitions** et du **système de fichiers racine**.

Récupérer les sources du noyau

Source	Méthode
Paquet de la distribution	SRPM du noyau (RedHat et dérivées)
Site officiel	kernel.org : archive <code>linux-x.y.z.tar.xz</code>

```
# cd /usr/src
curl -O https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-x.y.z.tar.xz
tar xJf linux-x.y.z.tar.xz          # extrait dans linux-x.y.z/
```

🔗 Répertoire des sources

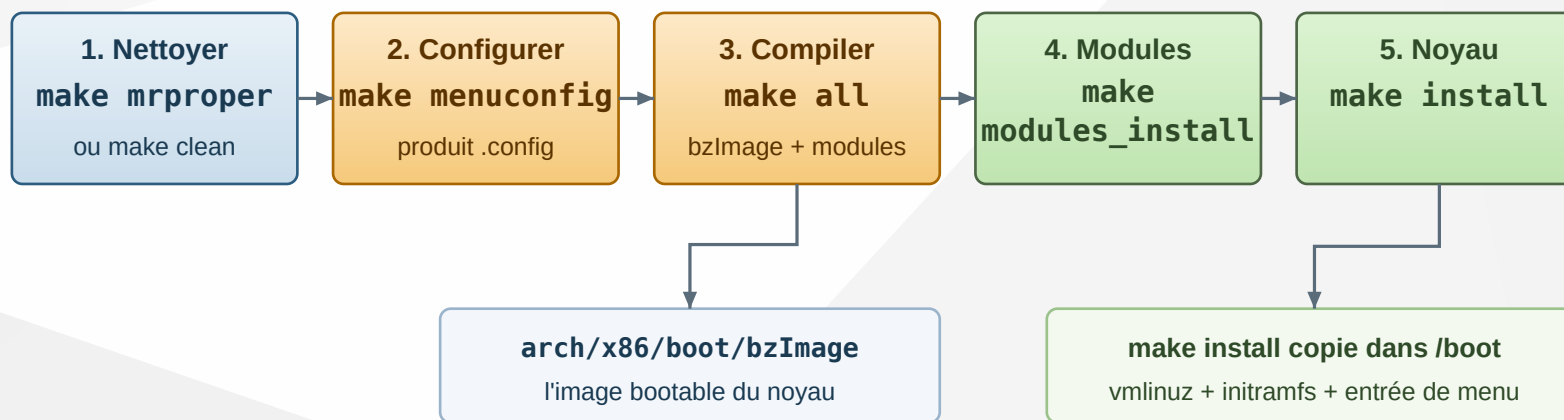
Par la suite, `/usr/src/linux-x.y.z` est le répertoire racine des sources.

Branches

kernel.org distingue les branches *mainline*, *stable* et *longterm* : pour un serveur, on privilégie une version **longterm** (support étendu), comme la 6.12, sur laquelle repose le noyau de RHEL 10.

Vue d'ensemble

Construire et installer un noyau dans `/usr/src/linux-x.y.z`



🔗 Cibles de make

`make help` liste toutes les cibles disponibles.

La cible `install` suppose les **modules déjà installés**.

Prérequis de compilation

Un compilateur et des bibliothèques de développement :

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

- `ncurses-devel` : pour `make menuconfig`,
- `openssl` (la commande) et `openssl-devel` : la compilation génère une clé de signature des modules, et s'arrête sans la commande `openssl`,
- `elfutils-libelf-devel` : outils du noyau,
- `dwarves` : informations de débogage BTF.

⚠ Une compilation demande de la place

Les sources, les fichiers objets et les modules occupent plusieurs Go : prévoir de l'espace dans `/usr/src` (`≥ 10 Go`, voir slide « Partitionnement : Tailles recommandées »).

Étapes 1 et 2 : nettoyer et configurer

```
# cd /usr/src/linux-x.y.z
make mrproper          # nettoie tout, y compris .config
```

L'objectif est de produire le fichier `.config` à la racine des sources.

Cible de configuration	Mode
<code>make config</code>	question par question (texte)
<code>make oldconfig</code>	reprend un <code>.config</code> , ne pose que les nouveautés
<code>make menuconfig</code>	menu texte ncurses (le plus courant)
<code>make localmodconfig</code>	ne garde que les modules actuellement chargés : un noyau bien plus petit et plus rapide à compiler
<code>make xconfig / gconfig</code>	interfaces graphiques

Ne pas partir de zéro : copier `/boot/config-$(uname -r)` en `.config`, puis `make oldconfig`.

Choix possibles : `[*]` built-in, `[M]` module, `[ ]` exclu. Ce choix demande de bien connaître le matériel.

Étape 3 : compiler

L'image noyau (`vmlinux`) devenant trop grosse, on produit une image **compressée** : la **bzImage** (gzip par défaut sur RHEL, ou bzip2, lzma, xz, lzo, lz4, zstd selon le choix de compression).

```
# make all # = make bzImage + make modules
```

Ou, pour forcer un rebuild propre sans perdre `.config`

```
# make clean && make all
```

Résultat :

- Le noyau bootable : `arch/x86/boot/bzImage`.
- Les modules sont **compilés**, mais **pas encore installés**.

🔗 `.version`

Supprimer `.version` remet à zéro le **compteur de compilations** (le `#1` de `uname -v`).

Étapes 4 et 5 : installer

1. Les modules d'abord (dans `/lib/modules/x.y.z`) :

```
# make modules_install INSTALL_MOD_STRIP=1      # strip = modules bien plus légers
```

2. Le noyau ensuite :

```
# make install
```

Rôle de `make install` : il enchaîne le travail fastidieux.

- copie `bzImage` dans `/boot` sous le nom `vmlinux-x.y.z` ;
- **génère l'initramfs** (sinon : `dracut`) ;
- ajoute une **entrée de menu BLS** (`/boot/loader/entries/`) et en fait **l'entrée par défaut** (sinon : `grubby`).

! Secure Boot

Un noyau compilé n'est **pas signé** : sous Secure Boot, le firmware ne le démarre pas, et le noyau, en mode `lockdown`, refuse aussi des modules non signés. Le TP se fait sur une machine sans Secure Boot, ou avec une clé que l'on enrôle (`mokutil`).

Le ramdisk initial et le cas monolithique

Toutes les distributions utilisent un **initramfs**.

✎ Rôle de l'initramfs

Il fournit au démarrage les pilotes nécessaires pour que le noyau puisse monter `/`.

```
# dracut --force /boot/initramfs-x.y.z.img x.y.z
```

✎ Se passer de l'initramfs

On peut s'en passer **si** le contrôleur disque, la table de partitions et le FS racine sont en **built-in**.

Noyau monolithique (sans aucun module) : désactiver `CONFIG_MODULES` (*Enable Loadable Module Support*).

Il ne reste alors que les choix `[*]` (oui) et `[ ]` (non), plus aucun `[M]`.

Bonnes pratiques

- Garder l'ancien noyau dans le menu GRUB : il sert de repli.
- Partir de la configuration du noyau en place (`/boot/config-*`).
- `make localmodconfig` pour une configuration réduite sur une VM.
- Compiler avec un utilisateur ordinaire, avec `-j` adapté au nombre de CPU.
- Prévoir la place disque : `/usr/src` et `/boot`.
- Noyau compilé soi-même : non signé, refusé sous Secure Boot.
- Suivi des correctifs de sécurité à assurer soi-même.

MODULE 9

Le noyau : paramètres de boot et sysctl

De quoi parle-t-on

Après avoir vu comment le noyau démarre et se compile, ce module couvre comment l'interroger et le paramétrer : ligne de commande de boot, `/proc`, et l'interface `sysctl`.

Au programme

- s'informer sur le noyau en place,
- la ligne de commande de boot, `/proc` et `/proc/sys`,
- `sysctl` et les paramètres utiles des classes kernel, vm, net.

Commandes et fichiers d'information

Commande / fichier	Information
<code>uname -a</code>	nom, version, date de compilation du noyau
<code>dmesg</code>	le <i>Kernel Ring Buffer</i>
<code>journalctl -k</code>	messages du noyau conservés par le journal, <code>-b -1</code> pour le démarrage précédent
<code>/proc/cmdline</code> , <code>/proc/version</code> , <code>/proc/cpuinfo</code>	état et configuration

Horodater dmesg à chaud (au boot : `printk.time=1`)

```
# echo Y > /sys/module/printk/parameters/time
```

Écrire dans le Kernel Ring Buffer

```
# echo "Salut" > /dev/kmsg
```

 Mine d'informations

Les pseudo-fichiers de `/proc` (`devices`, `partitions`, `interrupts`, `meminfo`, `ioports`...) sont une mine d'informations.

La ligne de commande du noyau

La **ligne de commande** du noyau peut être relue après coup dans `/proc/cmdline`.

```
$ cat /proc/cmdline
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-x.y.z root=/dev/mapper/vg-root ro rd.lvm.lv=vg/root rhgb quiet
```

Paramètre	Effet
root=	périphérique de la partition racine
ro / rw	montage initial de / en lecture seule / écriture
quiet, rhgb	démarrage silencieux / graphique
printk.time=1	horodater les messages noyau
init=/bin/bash, rd.break, systemd.unit=...	détourner le 1er processus / s'arrêter dans l'initramfs / choisir la cible
nomodeset, module.paramètre=...	options de pilotes

 **Pérenniser**

Pérenniser : `grubby --update-kernel=ALL --args="..."`, ou éditer `GRUB_CMDLINE_LINUX` dans `/etc/default/grub` puis `grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline` (voir slide « Modifier une option de démarrage du noyau »).

/proc et /proc/sys

`/proc` expose le noyau sous forme de **fichiers de taille nulle**, gérés en direct :

- un répertoire par processus : `/proc/<PID>/` (ex. `ls -l /proc/$$/cwd`).
- des informations système : `/proc/cpuinfo`, `/proc/meminfo`, `/proc/partitions`...
- des **paramètres réglables** sous `/proc/sys/...`.

```
# cat /proc/sys/net/ipv4/ip_forward      # lire un paramètre
echo 1 > /proc/sys/net/ipv4/ip_forward    # le modifier (volatile)
```

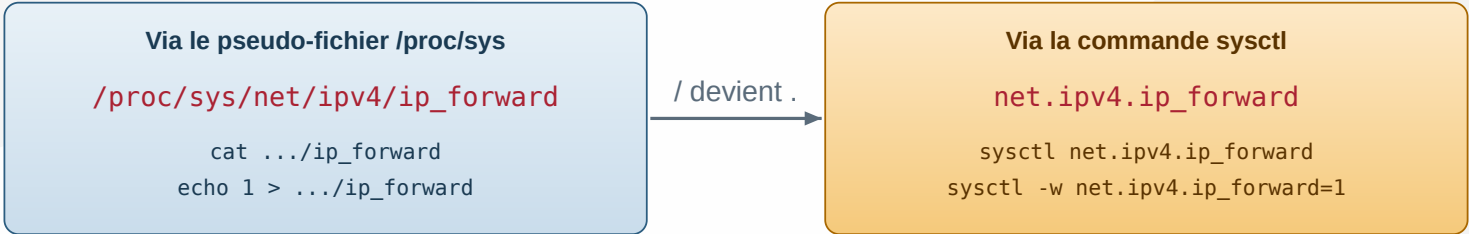
Documentation des paramètres : `kernel:Documentation/admin-guide/sysctl/`.

⚠ Modifier un paramètre avec `echo`

Cela fonctionne, mais n'est **pas persistant**. C'est plus propre d'utiliser `sysctl`.

sysctl : l'interface propre vers /proc/sys

Deux accès à un même paramètre



Usage	Effet
<code>sysctl param</code>	afficher la valeur (commande du paquet <code>procps-ng</code>)
<code>sysctl -a</code>	tout lister
<code>sysctl -w param=valeur</code>	modifier (volatile)
<code>sysctl -p [fichier]</code>	appliquer un fichier de conf (<code>/etc/sysctl.conf</code> par défaut)
<code>sysctl --system</code>	appliquer tous les fichiers de configuration

Persistence

Un fichier `/etc/sysctl.d/90-nom.conf` par thème, appliqué au démarrage (`sysctl --system` pour l'appliquer tout de suite). `/etc/sysctl.conf` est lu aussi, `/usr/lib/sysctl.d/` contient les valeurs fournies par les paquets.

Panorama des classes de paramètres

Le nombre de paramètres (`sysctl -a | wc -l`) dépend des fonctionnalités chargées dans le noyau.

Plusieurs classes

```
$ sysctl -a | cut -d. -f1 | sort -u
dev  fs  kernel  net  vm  ...
```

Classe	Domaine
kernel	internes du noyau (IPC, hostname, PID, panic...)
vm	mémoire virtuelle, swap, cache, OOM
net	réseau (la plus peuplée : core, ipv4, ipv6...)
fs	systèmes de fichiers (surtout des limites)
dev, sunrpc	périphériques, NFS/RPC

 Classes détaillées

On ne détaille ici que les classes les plus utiles à l'administration : **kernel**, **vm** et **net**.

Quelques paramètres kernel utiles

Nom d'hôte (modifiable)

```
$ sysctl kernel.hostname
```

1 + plus grand PID (2^{22} max en 64 bits)

```
# sysctl -w kernel.pid_max=131072
```

Reboot 10 s après un kernel-panic (0 = jamais)

```
# sysctl -w kernel.panic=10
```

Que faire en cas de ctrl-alt-del : 0 = transmis à init, >0 = reboot à chaud

```
$ sysctl kernel.ctrl-alt-del
```

IPC (certaines bases de données, applicatifs) : `kernel.sem`, `kernel.shmmax`, `kernel.shmall`, `kernel.msgmni`... Un besoin **applicatif**, pas un réglage de performances générique.

Non modifiables (informations) : `kernel.ostype`, `kernel.osrelease`, `kernel.version`.

```
# sysctl -w kernel.osrelease=8.0  
sysctl: setting key "kernel.osrelease": Operation not permitted
```

Swap, cache et écritures disque

Paramètre (défaut du noyau)	Effet
<code>vm.swappiness</code> (60)	tendance à swapper : 0 = éviter, 100 = swapper volontiers
<code>vm.vfs_cache_pressure</code> (100)	<100 garde <i>dentries/inodes</i> en cache, >100 les relâche
<code>vm.dirty_background_ratio</code> (10)	écriture en arrière-plan des <i>dirty buffers</i> au-delà de 10 %
<code>vm.dirty_ratio</code> (20)	écriture forcée au-delà de 20 %
<code>vm.drop_caches</code>	déclencheur : 1 pagecache, 2 <i>dentries+inodes</i> , 3 les deux

Sur un serveur : limiter le swap

```
# sysctl -w vm.swappiness=10
```

Vider les caches (tests, mesures)

```
# echo 3 > /proc/sys/vm/drop_caches
```

 `tuned` change ces valeurs

`virtual-guest` (VM) donne `vm.swappiness=30` et `vm.dirty_ratio=30`.
`drop_caches` n'est qu'un déclencheur ponctuel.

La surcharge mémoire : OOM killer et overcommit

Linux peut accepter d'allouer **plus** de mémoire que disponible (les programmes en demandent souvent trop). Si le pari échoue, le **OOM killer** tue un processus, parfois mal choisi (sshd, named...).

Paramètre	Valeurs
<code>vm.overcommit_memory</code> (0)	☐ heuristique, ☐ jamais de refus, ☐ refus strict
<code>vm.overcommit_ratio</code> (50)	% de RAM autorisé en plus du swap quand ☐ =2
<code>vm.oom_kill_allocating_task</code>	☐ tuer une victime heuristique, ☐ tuer le demandeur
<code>vm.panic_on_oom</code>	☐ tuer un processus, ☐ <i>kernel-panic</i>

Régler un refus d'allocation si trop de RAM demandé

```
# sysctl -w vm.overcommit_memory=2
```

Mode 2

Mode ☐2 : refus d'allocation au-delà de `swap + overcommit_ratio % de RAM`.

Routage et réglages par interface

C'est la classe la plus vaste (sous-catégories `core`, `ipv4`, `ipv6`...).

Le plus connu :

```
# sysctl -w net.ipv4.ip_forward=1      # activer le routage IPv4
```

Sous `net.ipv4.conf`, chaque interface a ses réglages, avec deux entrées spéciales :

Sous-clé	Portée
<code>net.ipv4.conf.all.X</code>	toutes les interfaces
<code>net.ipv4.conf.default.X</code>	valeur par défaut des futures interfaces
<code>net.ipv4.conf.<iface>.X</code>	une interface précise

```
≡ net.ipv4.conf.all.rp_filter = 1          # anti-spoofing
  net.ipv4.icmp_echo_ignore_broadcasts = 1 # ignorer les ping broadcast
  net.ipv4.ip_local_port_range = 32768 60999 # plage des ports sortants
```

Doc détaillée : pages de manuel `tcp(7)`, `ip(7)` et `kernel:Documentation/networking/ip-sysctl.rst`.

Bonnes pratiques

- Un paramètre à la fois, vérifié avant le suivant.
- Essai avec `sysctl -w`, puis réglage durable dans `/etc/sysctl.d/*.conf`.
- Un fichier par sujet dans `/etc/sysctl.d`, nommé avec un préfixe `nn-`.
- Commentaire dans le fichier : la raison du réglage.
- Paramètre de boot ajouté avec `grubby`, testé en console avant la production.
- Réglage appliqué seulement s'il corrige une mesure.

CHAPITRE 5

Blocage, crash et dépannage d'urgence

MODULE 10

Booter autrement

De quoi parle-t-on

Amorcer une machine sans passer par son disque local : pour réparer un système cassé, restaurer ou cloner une machine entière, ou installer en masse.

Au programme

- rescue : média d'installation, SystemRescue,
- **ReaR** : support de secours (ISO, clé USB), sauvegarde, restauration et clonage,
- **PXE avec dnsmasq** : installer des machines par le réseau avec kickstart.

Rescue : amorcer pour réparer

✎ Objectif

Objectif : amorcer une machine **sans passer par le disque local**.

Le média d'installation

L'ISO d'installation propose un **mode rescue** (entrée *Troubleshooting* du menu de démarrage) : il recherche les partitions existantes, monte les filesystems, et permet de `chroot` sur le système installé pour le réparer (GRUB, fstab, modules...).

SystemRescue (ex SystemRescueCd)

Live Linux spécialisé dans le dépannage : partitionnement, filesystems, LVM, RAID, sauvegarde/restauration d'images, accès réseau, éditeurs. Une ISO d'installation ou de SystemRescue est **hybride** : `dd` la copie telle quelle sur une clé USB.

```
# dd if=systemrescue-x.y-amd64.iso of=/dev/sdX bs=4M status=progress conv=fsync
```

! Désigner la clé entière

`/dev/sdX` doit pointer la clé entière, pas une partition. Vérifier avec `lsblk` avant `dd` : un mauvais périphérique est écrasé sans avertissement.

Site officiel : www.system-rescue.org

ReaR : sauvegarder une machine entière

ReaR (*Relax-and-Recover*) est la référence de la restauration **bare-metal** sous Linux, livrée dans le dépôt AppStream d'AlmaLinux 10 (version 2.9). Il fabrique un **support de secours amorçable** qui sait recréer partitionnement, LVM et systèmes de fichiers, puis restaurer les données.

Commande	Rôle
<code>rear mkrescue</code>	crée le support de secours seul
<code>rear mkbackup</code>	crée le support et la sauvegarde
<code>rear recover</code>	sur la machine cible démarrée sur le support : recrée le disque, restaure

Le support de secours (**OUTPUT**)

- **ISO** : image pour un CD/DVD ou une VM,
- **USB** : clé préparée par `rear format /dev/sdX`,
- **PXE** : démarrage par le réseau.

Doc : relax-and-recover.org | code : github.com/rear/rear

ReaR : configuration

Le fichier `/etc/rear/local.conf` dit quel **support** produire (`OUTPUT`), comment **sauvegarder** (`BACKUP`) et **où** ranger la sauvegarde (`BACKUP_URL`).

```
≡ OUTPUT=ISO
  BACKUP=NETFS
  BACKUP_URL=nfs://192.168.122.1/srv/rear
```

BACKUP_URL	Usage
nfs://serveur/export	archive <code>tar</code> sur un partage NFS, ISO comprise : le cas courant
usb:///dev/disk/by-label/REAR-000	clé USB préparée par <code>rear format</code>
iso:///backup	archive dans l'ISO : petites machines, moins de 2 Gio

 Outil de sauvegarde existant

ReaR sait aussi se limiter à recréer la machine et laisser un outil en place (Bareos, Veeam, NetBackup, IBM Storage Protect...) restaurer les données.

ReaR : restaurer ou cloner

1. Démarrer la machine **vierge** sur le support (ISO, clé, PXE), choisir l'entrée **Recover** suivie du nom d'hôte.
2. Ouvrir une session en `root` (sans mot de passe).
3. Lancer `rear recover`.
4. Retirer le support et redémarrer.

Cloner une machine revient à restaurer sa sauvegarde sur **une autre** machine.

⚠ Disque au moins aussi grand

ReaR recrée le partitionnement d'origine : le disque de la cible doit être **au moins aussi grand** que celui de la source.

💡 Un clone est une copie exacte

Nom d'hôte, `/etc/machine-id` et clés SSH sont ceux de la source : les renouveler sur le clone s'il vit à côté de l'original.

i Réétiquetage SELinux

ReaR laisse un fichier `/.autorelabel` : au premier démarrage, SELinux réétiquette tous les fichiers, ce qui rallonge ce démarrage.

PXE : démarrer par le réseau avec `dnsmasq`

1. Le client envoie une requête DHCP en indiquant son architecture (BIOS ou UEFI).
2. Le serveur répond avec une adresse, le serveur TFTP et le fichier de démarrage.
3. Le client télécharge ce fichier par TFTP : `pxelinux.0` en BIOS, `BOOTX64.EFI` en UEFI.
4. Ce chargeur lit son menu, puis télécharge le noyau et l'initrd.

`dnsmasq` rend à lui seul les deux services : **DHCP/BOOTP** et **TFTP**.

```
≡ interface=enp1s0
dhcp-range=192.168.150.201,192.168.150.250,12h
enable-tftp
tftp-root=/var/lib/tftpboot
# Architecture du client : 0 = BIOS, 7 et 9 = UEFI x86-64
dhcp-match=set:bios,option:client-arch,0
dhcp-match=set:efi64,option:client-arch,7
dhcp-match=set:efi64,option:client-arch,9
dhcp-boot=tag:bios,pxelinux/pxelinux.0
dhcp-boot=tag:efi64,redhat/EFI/BOOT/BOOTX64.EFI
```

À propos des serveurs DHCP

`dnsmasq` et **Kea** remplacent le serveur ISC DHCP (`dhcpd`), qui n'est plus livré depuis RHEL 10. Sur un hôte KVM, le réseau `default` est déjà servi par un `dnsmasq` que `libvirt` pilote.

PXE : fichiers de boot et menus

BIOS : `pxelinux.0` vient du paquet `syslinux-tftpboot` (dossier `/tftpboot/`). Le menu est `pxelinux/pxelinux.cfg/default`.

```
≡ label alma10
   kernel images/alma10/vmlinuz
   append initrd=images/alma10/initrd.img ip=dhcp inst.repo=http://serveur/depot inst.ks=http://serveur/ks.cfg
```

UEFI : les fichiers `EFI/` viennent de l'ISO de démarrage. Le menu est `redhat/EFI/BOOT/grub.cfg`.

```
≡ menuentry 'AlmaLinux 10' {
    linux images/alma10/vmlinuz ip=dhcp inst.repo=http://serveur/depot inst.ks=http://serveur/ks.cfg
    initrd images/alma10/initrd.img
}
```

Le noyau et l'initrd de l'installateur (`vmlinuz`, `initrd.img`) sont dans `images/pxeboot/` du dépôt d'installation (`inst.repo=`), `inst.ks=` donne le fichier kickstart.

💡 Un menu par machine

`pxelinux` cherche d'abord `01-<mac>` (MAC en minuscules, tirets), puis l'IP en hexadécimal (`gethostip`), puis `default`. GRUB cherche `grub.cfg-01-<mac>`, puis `grub.cfg`.

Bonnes pratiques

- Tester la restauration ReaR sur une machine vierge, avant le jour où elle sert.
- Ranger ISO et sauvegarde **hors** de la machine sauvegardée.
- Refaire `rear mkbbackup` après un changement de disques ou de LVM (`rear checklayout` le détecte).
- Cible de restauration : disque au moins aussi grand que la source.
- Un seul serveur DHCP par réseau.
- Un menu PXE par machine quand les installations diffèrent.
- Fichier kickstart servi : son mot de passe chiffré est lisible par tout le réseau.

MODULE 11

Démarrage du système et GRUB 2

De quoi parle-t-on

Entre l'allumage de la machine et le premier prompt de connexion, une chaîne précise d'étapes : firmware, chargeur, noyau, initramfs. GRUB 2 est le chargeur standard : ce module explique son rôle, sa configuration, et comment reprendre la main quand il ne se comporte pas comme prévu.

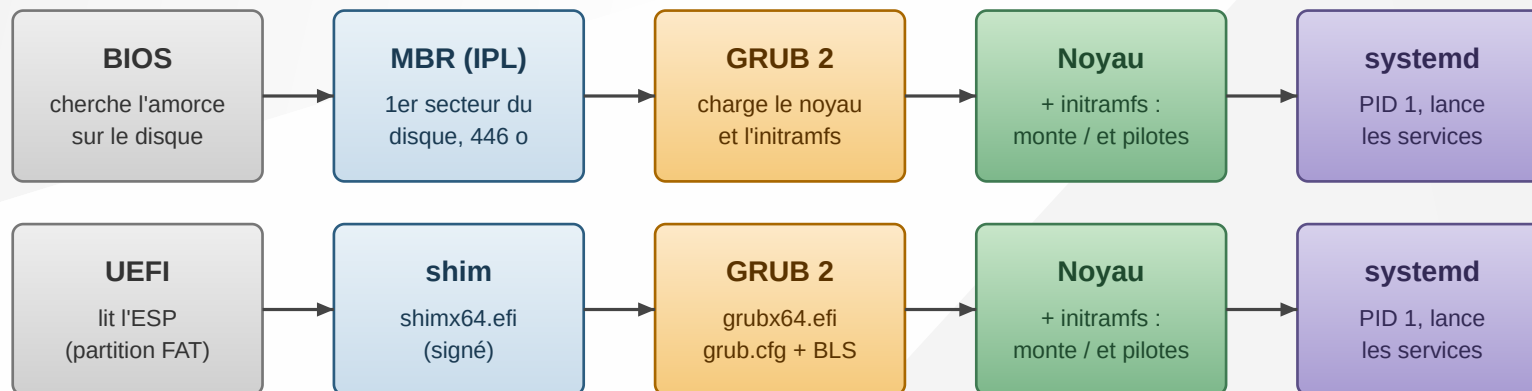
Au programme

- la chaîne de démarrage, en BIOS et en UEFI,
- utiliser le menu GRUB 2, en démarrage manuel aussi,
- reprendre la main sur un système dont le mot de passe est perdu,
- réinstaller GRUB 2, dépanner un système qui ne démarre plus.

La configuration est en annexe

`grub.cfg`, les entrées BLS, `grubby`, les options du noyau et le mot de passe de GRUB 2 sont dans l'annexe « GRUB 2 : configuration avancée et sécurité » (slide « Le fichier de configuration »).

La chaîne de démarrage



Sans programme exploitable à une étape : « OS not found » ou « kernel panic »

Le noyau doit **monter** son système de fichiers racine (`/`). La partition concernée est connue grâce au paramètre `root=` passé par le chargeur.

- Pour y accéder, le noyau a besoin de **pilotes** (contrôleur disque, lecture du format de filesystem...),
⇒ sinon : *"unable to mount root filesystem : kernel panic !"*
- On passe donc par un **initramfs** (*Initial RAM filesystem*, `/boot/initramfs-*.img`) : un mini-système chargé en mémoire par le chargeur, qui fournit ces pilotes.

BIOS : le MBR



Le **MBR** est le premier secteur d'un disque, historiquement appelé *boot-sector*. Sa structure de **512 octets** se décompose en trois zones :

- **446 octets** : l'**IPL**, qui peut contenir du code exécutable d'amorçage.
- **64 octets** : **table de partitions**, 4 partitions primaires au maximum (4 x 16 o).
- **2 octets** : le *magic number* (signature **0xAA55**).

Le **BIOS** exécute l'IPL du disque d'amorçage. S'il n'y en a pas, il exécute le premier secteur de la partition **active** (cas des OS Microsoft). Sinon : **"OS not found !"**.

💡 BIOS ou UEFI ?

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

 indique le mode dans lequel la machine a démarré.

Rôle du chargeur de démarrage

Pour démarrer un système, on utilise un **chargeur de démarrage** (*boot-loader*). Il permet de faire cohabiter plusieurs OS, ou de démarrer un OS de différentes manières.

 Rôle de GRUB

C'est le chargeur qui **charge le noyau Linux et lui passe des paramètres**.

Chargeur	Statut
LILO	Chargeur historique de Linux, n'évolue plus depuis 2015, remplacé par GRUB.
GRUB 2	Interpréteur de commandes paramétrable dynamiquement au boot, chargeur des distributions modernes , en BIOS comme en UEFI.
Autres	U-Boot (denx.de) et BareBox (barebox.org), surtout en embarqué.

 Avantage de GRUB 2

GRUB 2 évite la réinstallation lors d'un changement d'options ou de menu, au prix d'une configuration plus complexe.

Reprendre la main : mot de passe perdu

C'est au niveau du premier processus que tout se joue. Créé par le noyau, il exécute normalement `/sbin/init` (un lien vers `systemd`).

1. Dans GRUB, touche `e`, puis ajouter à la fin de la ligne `linux` :

```
≡ rd.break
```

2. **Ctrl-x** pour booter : le démarrage s'arrête dans l'`initramfs`, avant de passer la main au système. Le disque est monté sous `/sysroot` :

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

L'`initramfs` ne contient pas la commande `chroot` : on appelle celle du système monté, `/sysroot/sbin/chroot`. `touch /.autorelabel` demande à SELinux de réétiqueter les fichiers au démarrage suivant : sans lui, la connexion avec le nouveau mot de passe peut échouer.

Reprendre la main : `init=/bin/bash`

On peut aussi remplacer le premier processus par un shell, grâce au paramètre `init=` du noyau. Dans GRUB (`e`), ajouter à la ligne `linux` :

```
≡ init=/bin/bash
```

Une fois le shell obtenu, après **Ctrl-x** :

```
# mount -o remount,rw /      # passer la racine en lecture-écriture
passwd                      # changer le mot de passe
touch /.autorelabel         # relabel SELinux au reboot
exec /sbin/init              # reprendre un démarrage normal
```

⚠ Quand cela ne suffit pas

Un menu GRUB protégé par mot de passe (voir slide « Sécuriser le menu GRUB 2 ») interdit l'édition. Un disque chiffré (LUKS) demande sa phrase de passe.

i Compte root verrouillé

Une installation avec `rootpw --lock` n'a pas de mot de passe `root` à retrouver : c'est celui de l'utilisateur administrateur (`passwd admin`) qu'on réinitialise par les mêmes méthodes.

Le menu proposé

Une ligne par choix

On peut trouver plusieurs lignes pour un même système (versions de noyau ou options différentes), et souvent un mode de **récupération** (*rescue*).

Edition à la volée

Touche	Action
Entrée	valider et démarrer normalement
e	éditer l'entrée avant de démarrer
c	passer en ligne de commande

Démarrage automatique et timeout

GRUB démarre un choix par défaut au bout d'un certain délai. N'importe quelle touche **interrompt le compte à rebours** et fait apparaître le menu.

Paramétrage

Le choix par défaut et la durée du compte à rebours sont **paramétrables**.

Éditer une entrée avant de démarrer

Après un appui sur **e**, les lignes du choix sélectionné s'affichent dans un éditeur minimal façon Emacs et peuvent être modifiées.

Raccourci	Action
CTRL-x	booter
ESC	annuler
CTRL-c	passer en ligne de commande

- La touche **Tabulation** offre la complétion sur le premier mot (la commande) et le second (fichiers, périphériques...).
- Les modifications faites ici **ne sont pas enregistrées** dans le fichier de configuration.

Documentation des mots-clés : [Manuel GRUB](#)

Utiliser GRUB 2 en mode manuel

Il faut fournir à GRUB 2 toutes les informations permettant de démarrer le système :

```
≡ insmod part_gpt          # charger les modules GRUB 2 nécessaires
  insmod xfs
  set root='hd0,gpt2'      # ou search : où trouver les fichiers
  linux /vmlinuz-... root=/dev/mapper/vg00-lv_root ro
  initrd /initramfs-....img
  boot                    # démarre la configuration saisie
```

L'important est de lire la réponse à chaque étape.

GRUB vierge

Booter ainsi n'impose pas d'installer GRUB sur le disque au préalable : cela fonctionne avec un *GRUB vierge sur CD ou clé USB*.

Tout ceci est plus facile avec un **keymap français**, qui n'est pas installé par défaut.

Installer ou réinstaller GRUB 2

GRUB 2 est normalement installé pendant l'installation du système. En **BIOS**, `grub2-install` écrit le chargeur dans le MBR du **disque** (pas d'une partition) :

```
# grub2-install /dev/sda  
grub2-mkconfig -o /boot/grub2/grub.cfg
```

⚠ Seulement en BIOS

Sur une machine UEFI, `grub2-install` ne remet pas les fichiers signés de l'ESP : voir slide « Réinstaller GRUB 2 en UEFI ».

Quand le système ne boote plus

On réalise l'opération depuis l'**image ISO d'installation** :

- Booter l'ISO, puis **Troubleshooting** puis **Rescue a AlmaLinux system**.
- À la proposition de montage : *Continue*, puis *OK*. Le système du disque est monté dans `/mnt/sysroot`.

```
# loadkeys fr  
chroot /mnt/sysroot
```

Dans le `chroot`, en BIOS :

```
# grub2-install /dev/sda  
grub2-mkconfig -o /boot/grub2/grub.cfg
```

i Machine UEFI

La réparation en UEFI diffère : voir slide « Dépanner en UEFI ».

Bonnes pratiques

- Essayer `rd.break` sur une VM de test, avant d'en avoir besoin.
- Réinstaller GRUB 2 par le mode de démarrage de la machine (BIOS ou UEFI).
- Tester les changements dans la console d'une VM avant la production.
- `touch /.autorelabel` après un changement de mot de passe root en `rd.break`.
- Un accès à la console permet de reprendre la main : protéger GRUB par mot de passe (annexe « GRUB 2 : configuration avancée et sécurité »).

MODULE 12

systemd : démarrage et services

De quoi parle-t-on

Le premier processus gère le démarrage et pilote la machine : ce module suit son évolution (*SysVinit*, *Upstart*, *systemd*) puis détaille les outils et les unités de *systemd*, la référence actuelle.

Au programme

- runlevels et targets, prise en main de *systemd*,
- les unités *systemd* : structure, dépendances, surcharge,
- créer et activer son propre service.

Le premier processus

Le noyau crée un premier processus, souvent appelé *process d'init*, qui gère le démarrage (services notamment) et globalement pilote la machine.

Son rôle se découpe en 3 étapes :

1. Réaliser des actions de **bas niveau** : monter les filesystems, nommer la machine, activer le swap...
2. **Lancer les services** : établir une configuration **cible** (un état de la machine).
3. Mettre en place un **moyen de se connecter** (consoles, environnement graphique), avec relance des processus surveillés.

Le monde Linux a connu 3 époques, 3 projets de *process d'init* :

... ~ 2006	~ 2006-2010	~ 2014 ...
SysVinit	Upstart (Ubuntu 6.10 à RHEL 6)	systemd

Depuis systemd

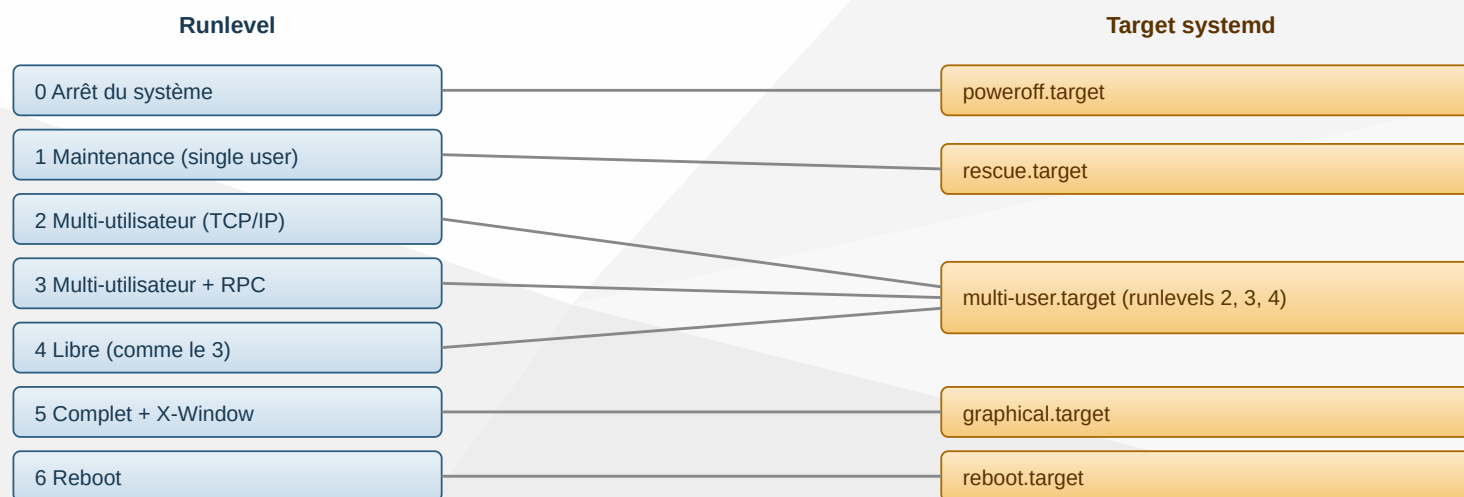
Depuis 2014 (RHEL/CentOS 7, Debian 8...), c'est **systemd** qui gère le démarrage et les services.

Runlevels et targets

Le premier processus gère différents **modes** dans lesquels certains services existent et d'autres non. Ces modes étaient des **runlevels** (SysVinit, Upstart) et deviennent des **targets** (systemd).

Ils sont paramétrés :

- par `/etc/inittab` dans le cas de **SysVinit** ;
- par les fichiers de `/etc/init` dans le cas de **Upstart** ;
- par des **unités** de type `target` (fichiers de `/usr/lib/systemd/system`) dans **systemd**.



🔧 Runlevel courant

Runlevel courant : `runlevel` ou `who -r`, ou plutôt `systemctl get-default`.

Prise en main de systemd

systemd a le même rôle que SysVinit et Upstart, mais offre bien plus de fonctionnalités.
AlmaLinux 10 livre **systemd 257** (`systemctl --version`).

C'est aussi un ensemble de **nouveaux outils** :

Commandes en <code>*ctl</code>	Commandes <code>systemd-*</code>	Les classiques
<code>systemctl</code>	<code>systemd-analyze</code>	<code>halt</code>
<code>journalctl</code>	<code>systemd-cgls</code> / <code>systemd-cgtop</code>	<code>init</code>
<code>hostnamectl</code>	<code>systemd-nspawn</code>	<code>poweroff</code>
<code>localectl</code>	<code>systemd-detect-virt</code>	<code>reboot</code>
<code>loginctl</code>	<code>systemd-tmpfiles</code>	<code>runlevel</code>
<code>timedatectl</code>	<code>systemd-run</code>	<code>shutdown</code>
<code>bootctl</code> / <code>busctl</code>	<code>systemd-firstboot</code>	<code>telinit</code>
<code>coredumpctl</code> / <code>machinectl</code>	<code>systemd-inhibit</code>	<code>udevadm</code>

Les targets, sortes de runlevels

À propos des targets

- Lister les targets : `systemctl list-units --type=target`.
- Changer de cible courante : `systemctl isolate nom.target`.
- Cible par défaut : `systemctl get-default` / `systemctl set-default`.

🔗 Cible par défaut

La target par défaut est celle définie dans `/usr/lib/systemd`. Si on utilise `systemctl set-default`, ce paramétrage est surchargé, dans `/etc/systemd`.

```
$ ls -l $(find {/etc,/usr/lib}/systemd/system -name default.target)
lrwxrwxrwx 1 root root 16 13 avril 21:38 /usr/lib/systemd/system/default.target -> graphical.target
```

Connaître l'état du système

```
$ systemctl is-system-running
degraded
```

Code retour

Le code retour est alors différent de 0 (utile pour scripter ou superviser).

État	Signification
initializing	première phase, avant <code>basic.target</code> ou la maintenance
starting	en cours de démarrage
running	système opérationnel
degraded	opérationnel, mais une ou plusieurs unités en échec
maintenance	mis en maintenance (ou <i>emergency</i>)
stopping	procédure d'arrêt en cours

```
$ systemctl status
State: degraded      Failed: 1 units
```

reset-failed

Une fois le service *Failed* identifié et réparé : `systemctl reset-failed le-service`.

Équivalences systemctl / service / chkconfig

`systemctl` remplace le couple `service` et `chkconfig`, qu'il faut désormais oublier.

<code>systemctl</code>	<code>service</code> / <code>chkconfig</code>
<code>start</code> / <code>stop</code> / <code>restart</code>	<code>service nom start</code> / <code>stop</code> / <code>restart</code>
<code>try-restart</code>	<code>service nom condrestart</code>
<code>reload</code>	<code>service nom reload</code>
<code>status</code> / <code>is-active</code>	<code>service nom status</code>
<code>enable</code> / <code>disable</code>	<code>chkconfig nom on</code> / <code>off</code>
<code>is-enabled</code>	<code>chkconfig --list service</code>

Lister :

```
$ systemctl list-unit-files --type service
systemctl list-units --type service --all
systemctl list-units --type mount
systemctl list-units --type timer
```

Les unités de systemd

Plus de *scripts* !

Toute chose est désormais une **unité**, définie par un fichier dans l'un de ces répertoires (par priorité croissante) :

Répertoire	Description
/usr/lib/systemd/system/	unités distribuées avec les paquets RPM
/run/systemd/system/	unités créées au <i>runtime</i> (l'emportent sur /usr/lib)
/etc/systemd/system/	unités de l'administrateur (l'emportent sur /run)

.service	.target	.mount	.automount
.timer	.socket	.swap	.path
.device	.scope	.slice	

 Types d'unités

Chaque type d'unité a son suffixe, et son bloc [Type] particulier.

Définition et structure d'une unité

Une unité est définie par un fichier au format proche des `.ini` :

```
[section]
# Affectation vide : réinitialise les valeurs précédentes
Directive=
Directive=valeur
Directive=valeur1,valeur2
```

Quel que soit le type, on trouve les sections `[Unit]` et `[Install]` :

- `[Unit]` : définitions indépendantes du type.
 - `Description=`,
 - `Documentation=`,
 - dépendances (`Requires=`, `Wants=`, `Conflicts=`),
 - ordonnancement et pré-requis (`Before=`, `After=`, `Condition*=`).
- `[Install]` : activation/désactivation (voir `systemctl enable` / `disable`).
- Une **section spécifique au type** : `[Service]`, `[Mount]`, `[Timer]`...

```
Doc: systemd.unit(5),
puis systemd.service(5), systemd.mount(5), systemd.timer(5)...
```

Dépendances entre unités

Dans la **section** `[Unit]` : de quelle unité nous dépendons.

Directive	Comportement
Requires=	dépendance absolue : si une dépendance citée échoue ou s'arrête, l'unité aussi
Bindsto=	lien fort dans les deux sens (arrêt mutuel)
Wants=	similaire à Requires mais plus souple : pas d'impact en cas d'échec de la dépendance

Dans la **section** `[Install]` : quelle unité réclame celle-ci.

`RequiredBy=` / `WantedBy=` : utilise des répertoires `.service.wants` ou `.service.requires` de l'unité mentionnée.

Lister les dépendances

```
$ systemctl list-dependencies
systemd-analyze dot --require --from-pattern='*.target' | dot -Tsvg > targets.svg
```


Surcharge d'une unité

Les unités livrées par les paquets sont dans `/usr/lib/systemd/system/unité.type`.

L'administrateur écrit ses propres unités dans `/etc/systemd/system/unité.type`.

On peut **surcharger** une unité fournie par un paquet.

- via un fichier dans `/etc/systemd/system/unité.type.d/fichier.conf`.
- La création de surcharges est facilitée par : `systemctl edit`

```
# systemctl edit unité.type      # crée le répertoire .d et le fichier de surcharge
```

Fichiers de surcharge

Les fichiers de surcharge contiennent eux aussi des sections et des directives.

Targets, mount, timer, socket

Fichier `/usr/lib/systemd/system/multi-user.target`

```
[Unit]
Description=Multi-User System
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes

[Install]
Alias=default.target
```

🔗 Pas de section `[Target]`

Il n'existe **pas** de section `[Target]`.

Une target ne fait que **grouper** des unités : elle les réclame par des dépendances (`Wants=` des unités qui la visent par `WantedBy=`) et fixe un point d'ordonnancement (`After=`).

Type	Exemple	Vocation
.mount	<code>tmp.mount</code>	pourrait remplacer <code>/etc/fstab</code>
.timer	<code>dnf-makecache.timer</code>	pourrait remplacer CRON
.socket	<code>cockpit.socket</code>	démarre un service à la première sollicitation

Les unités de type service

Section `[Service]` (`systemd.service(5)`) : décrit un processus à superviser .

Type=	Comportement
simple	processus père restant en avant-plan, les autres unités démarrent en //
oneshot	comme simple, mais le processus doit rendre la main avant de poursuivre
idle	comme simple, mais lancé une fois les autres unités dispatchées
exec	comme simple, mais <code>systemd</code> attend que le programme soit réellement lancé
forking	la commande forke au lancement (named...)
dbus / notify	on attend la présence sur DBUS, ou une notification

Directives courantes des services

- Pilotage du processus `ExecStart=` / `ExecReload=` / `ExecStop=`
 - Voir aussi / `ExecStartPre=` / `ExecStartPost=`.
- `Restart=` (relance en cas de mort, sauf `stop/restart`), `RestartSec=` (délai).
- `Environment=`, `EnvironmentFile=`
 - préfixe `-` : l'absence du fichier n'est pas une erreur
 - plusieurs clauses possibles,
- `PIDFile=`.

Installer un nouveau service

Créer `/etc/systemd/system/monservice.service`

```
[Unit]
Description=Mon service

[Service]
ExecStart=/path/to/mon-script

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

`mon-script` doit commencer par `#!/usr/bin/bash` et être exécutable (`chmod +x`).

⚠ SELinux et emplacement du script

`systemd` lance sans problème un script étiqueté `bin_t`, comme ceux de `/usr/local/bin`. Un script d'un répertoire personnel est refusé par SELinux.

Puis l'activer et le vérifier :

```
# systemctl enable monservice.service
systemctl cat monservice.service    # afficher / lister
systemctl start monservice.service
```

⚠ Application immédiate

`systemctl edit` fonctionne aussi, mais attention : **application immédiate !**

Bonnes pratiques

- Unités personnelles dans `/etc/systemd/system` (`/usr/lib/systemd/system` appartient aux paquets).
- `systemctl edit` (surcharge) plutôt que la copie d'une unité entière.
- `systemctl daemon-reload` après chaque modification d'unité.
- `systemctl cat` et `systemd-analyze verify` pour contrôler le résultat.
- Dépendances (`After=`, `Requires=`, `Wants=`) déclarées explicitement.
- Un `Type=` adapté au processus lancé.
- Diagnostic : `systemctl status`, puis `journalctl -u`.

CHAPITRE 6

Maintenance et métrologie

MODULE 13

Les logs

De quoi parle-t-on ?

Un système Linux *raconte sa vie* en permanence : démarrages, connexions, erreurs, refus d'accès. Ces messages suivent une chaîne : ils sont **émis**, **collectés**, **triés**, **stockés**, puis **purgés** avant de saturer le disque.

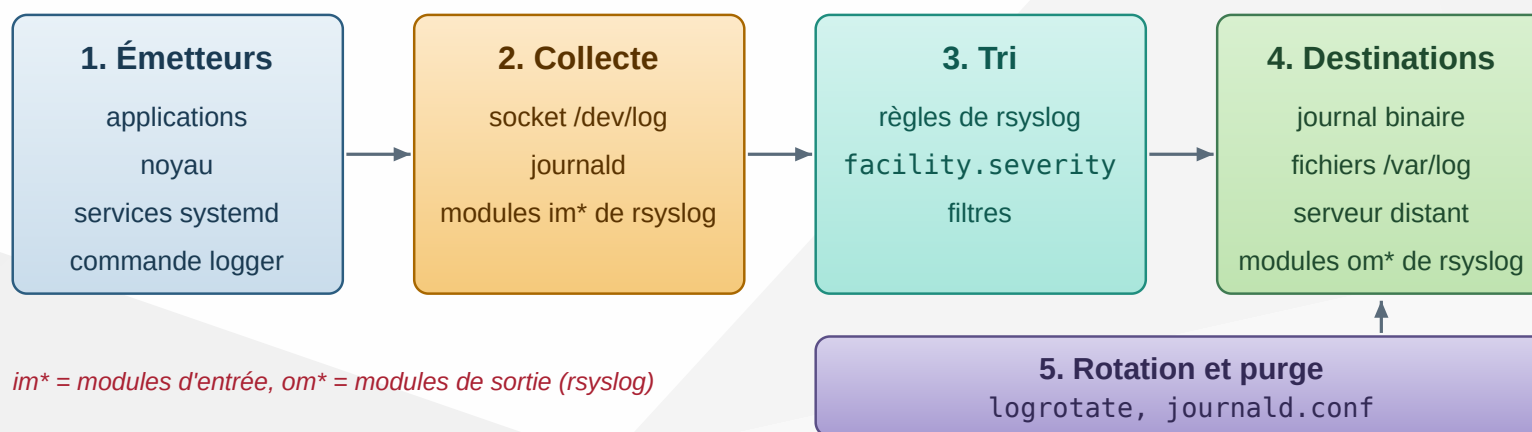
Sur AlmaLinux 10, `journald` est la base de cette chaîne. Selon les cas, on ajoute `rsyslog` pour écrire des fichiers texte dans `/var/log` ou centraliser les messages vers un autre serveur.

Au programme

- le principe de la chaîne, le protocole syslog (`facility.severity`, `logger`, `/dev/log`),
- `journald` et `journalctl`, persistance et taille du journal,
- l'annexe « Gestion avancée et centralisation des logs » : `rsyslog`, `logrotate` et la centralisation.

La chaîne de journalisation

De l'application au fichier de log



Un message part d'un **émetteur**, entre par un point de **collecte**, est **trié** par des règles, puis atteint une **destination**. La **rotation** empêche ensuite les fichiers de s'accumuler.

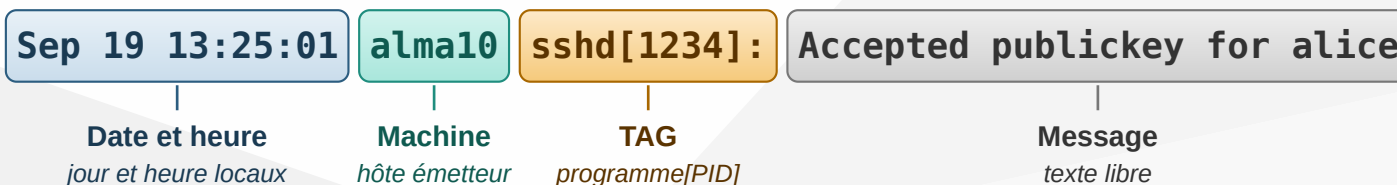
Syslog : la convention commune

Le protocole **syslog** normalise le **format** d'un message et sa **priorité**. C'est le standard historique de journalisation des systèmes Unix et Linux (RFC 3164 à l'origine, puis **RFC 5424**).

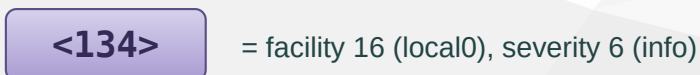
Les messages sont émis par les programmes (fonction C `syslog(3)`, commande `logger(1)`) et par le noyau.

Anatomie d'un message syslog

Tel qu'il apparaît dans un fichier de log :



Sur le réseau, ou avec `logger -p`, le message est précédé de sa priorité :



🔗 Calcul de la priorité : $PRI = facility \times 8 + severity$.

Ici `<134>` vaut $16 \times 8 + 6$: facility `local0`, severity `info`.

Priorité : facility.severity

Facility : qui émet ?			Severity : quelle gravité ?		
0	kern	Messages du noyau	0	emerg	Système inutilisable
1	user	Programmes utilisateur	1	alert	Action immédiate requise
2	mail	Messagerie	2	crit	Condition critique
3	daemon	Services sans catégorie	3	err	Erreur
4	auth	Authentification	4	warning	Avertissement
5	syslog	Messages internes de syslog	5	notice	Normal, mais significatif
6-8	lpr news uucp	Historiques, peu utilisés	6	info	Information
9	cron	Planificateur (cron, at)	7	debug	Débogage
10	authpriv	Authentification (privé)			
11	ftp	Serveur FTP			
16-23	local0-7	Libres pour vos applications			

■ système

■ sécurité

■ services

■ usage libre

Plus le numéro est petit, plus le message est grave.

Un sélecteur comme `mail.err` retient les messages de la facility `mail` de gravité `err` et plus graves.

La commande `logger`

`logger` poste un message dans le journal système depuis le shell ou un script : pratique pour tester une chaîne de journalisation. Il écrit sur `/dev/log`, ou envoie vers un serveur distant avec `--server`.

```
$ logger -p local0.info -t ESSAI 'Message à caractère informatif'
ls -l | logger -p local0.debug -t LOGSTDIN      # trace l'entrée standard
```

Option	Rôle
<code>-p</code>	priorité : couple <code>facility.severity</code>
<code>-t</code>	tag : un nom libre, souvent <code>programme[PID]</code>

Pour retrouver le message dans le journal :

```
$ journalctl -t ESSAI -n 1
```

```
≡ Sep 19 13:30:58 alma10 ESSAI[3832]: Message à caractère informatif
```

`/dev/log` : la porte d'entrée

Les programmes qui utilisent `syslog()` écrivent dans une **socket Unix**, `/dev/log` : c'est le point de collecte historique.

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

```
lrwxrwxrwx. 1 root root 28 Dec  1 11:55 /dev/log -> /run/systemd/journal/dev-log
```

🔗 Cette socket aujourd'hui est tenue par l'unité `systemd-journald-dev-log.socket`.

Un message émis avec `syslog()` ou `logger` arrive donc d'abord dans le journal.

ℹ Dans des distributions sans systemd

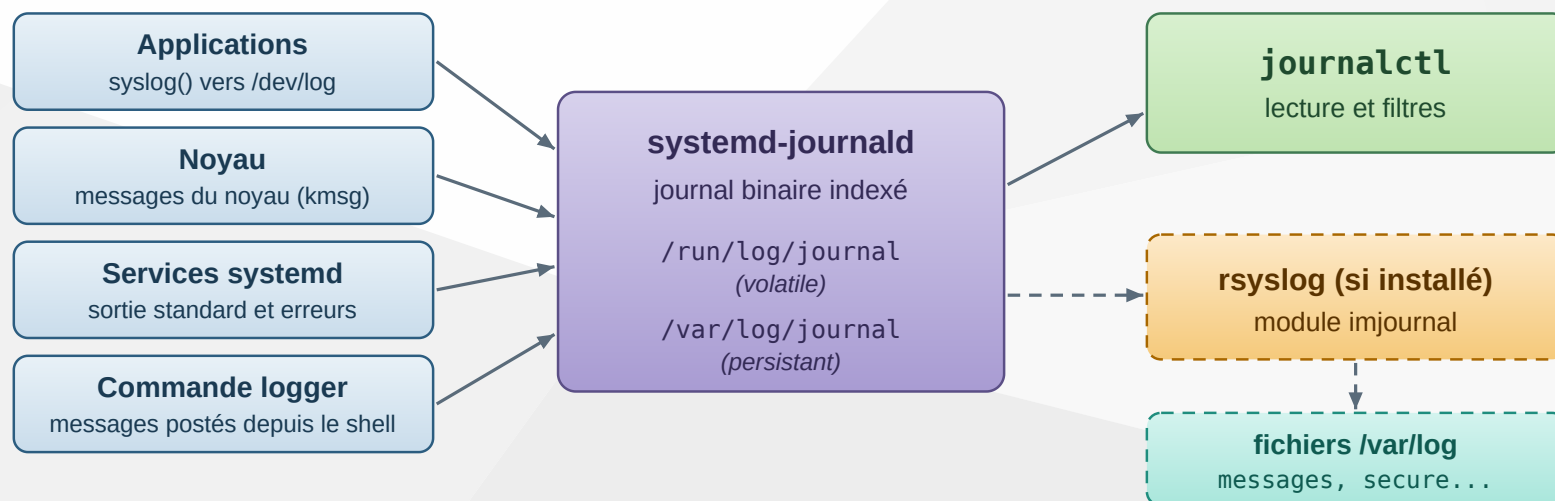
Alors pas de `journald`, et `rsyslog` écoute `/dev/log` lui-même avec le module `imuxsock`.

journal : le journal de systemd

`systemd-journald` **collecte**, **stocke** et **indexe** les messages de toute la machine : programmes (`/dev/log`), noyau, sortie standard des services systemd.

- un format **binaire indexé**, pour des recherches rapides,
- des **métadonnées** ajoutées à chaque message : unité, PID, UID, démarrage...
- une lecture avec `journalctl` : le fichier n'est pas lisible tel quel.

Qui écrit dans le journal, qui le lit



journalctl : lire le journal

```
$ journalctl -n 20           # les 20 derniers messages
journalctl -f               # suivre en direct
journalctl -u sshd          # une unité systemd
journalctl -b               # depuis le dernier démarrage
journalctl -b -1            # le démarrage précédent
journalctl -p err           # priorité err et plus grave
journalctl --since "1 hour ago" # une plage horaire
journalctl -k               # les messages du noyau
journalctl -t ESSAI         # un tag précis
journalctl -g "Failed"      # une recherche (expression rationnelle)
```

Les filtres se combinent : `journalctl -u sshd -p warning --since today`.

💡 Voir tout le journal

Sans privilèges, un utilisateur ne voit que ses propres messages.

`sudo journalctl`, ou l'appartenance au groupe `systemd-journal`, `adm` ou `wheel`, donne accès à tout le journal.

Les champs d'un message

`journalctl -o verbose` montre tout ce que `journald` retient d'un message :

```
$ journalctl -t ESSAI -n 1 -o verbose
```

```
≡ Sat 2026-09-19 13:30:58.118019 CEST [s=...;i=2123b;b=...;m=...;t=...;x=...]  
  PRIORITY=6  
  _UID=1000  
  _GID=1000  
  _MACHINE_ID=3f9a1c0e5b7d4e28a6c1d9b04f7e2a13  
  _HOSTNAME=alma10  
  _RUNTIME_SCOPE=system  
  _TRANSPORT=syslog  
  _BOOT_ID=8c2d6e41f0a94b7fb3a5d1e9c07f6a52  
  SYSLOG_FACILITY=16  
  SYSLOG_IDENTIFIER=ESSAI  
  SYSLOG_TIMESTAMP=Sep 19 13:30:58  
  MESSAGE=Message à caractère informatif  
  _PID=3832  
  _SOURCE_REALTIME_TIMESTAMP=1789817458118019
```

Chaque champ peut servir de filtre : `journalctl _UID=1000`.

Persistence du journal

Le paramètre `Storage` de `journald.conf` choisit où le journal est conservé :

```
# systemd-analyze cat-config systemd/journald.conf | grep Storage
#Storage=auto
```

Valeur	Stockage
<code>auto</code> (défaut)	persistant si <code>/var/log/journal</code> existe, <i>volatile</i> sinon
<code>volatile</code>	<code>/run/log/journal</code> , en mémoire : effacé au redémarrage
<code>persistent</code>	<code>/var/log/journal</code> , sur disque
<code>none</code>	rien n'est conservé

⚠ Un journal *volatile* disparaît au redémarrage

Sur AlmaLinux, `/var/log/journal` n'existe pas à l'installation : le journal est donc *volatile*. Les messages qui précèdent un redémarrage, planté ou non, sont perdus.

💡 Rendre persistant le journal

Il suffit de créer le répertoire `/var/log/journal` et de rebooter... ou bien...

Éditer la configuration de journald

On peut surcharger la configuration par défaut, plutôt grâce à un *drop-in directory*.

```
# mkdir -p /etc/systemd/journal.conf.d  
vi /etc/systemd/journal.conf.d/10-local.conf
```

Exemple pour rendre le journal persistant :

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

Créer le répertoire de stockage et redémarrer ensuite `journald`

```
# mkdir /var/log/journal  
systemctl restart systemd-journal
```

Vérifier :

```
$ journalctl --list-boots
```

⚠ Dans tous les cas il semble nécessaire de rebooter.

Sans cela, `/var/log/journal` reste vide.

Limiter l'espace occupé par le journal

`journald` fait lui-même la rotation de son journal : il supprime les fichiers les plus anciens quand une limite est atteinte. Les limites se règlent dans la config.

Paramètre	Rôle
<code>SystemMaxUse</code>	espace maximal occupé par le journal (<i>défaut : 10 % du système de fichiers</i>)
<code>SystemKeepFree</code>	espace à laisser libre (<i>défaut : 15 %</i>)
<code>SystemMaxFileSize</code>	taille maximale d'un fichier de journal
<code>MaxRetentionSec</code>	durée maximale de conservation

```
$ journalctl --disk-usage
sudo journalctl --vacuum-size=200M      # ramener le journal à 200 Mo
sudo journalctl --vacuum-time=2weeks    # ne garder que 2 semaines
```

```
≡ Archived and active journals take up 96.0M in the file system.
```

i Deux mécanismes de rotation

`journald` gère seul la rotation de son journal. Les fichiers texte de `/var/log` sont traités par `logrotate` : voir l'annexe « Gestion avancée et centralisation des logs ».

Bonnes pratiques

- Logs sur une partition dédiée ou limités en taille.
- Journal persistant (`Storage=persistent`) sur les serveurs.
- Machines à l'heure (NTP) : les horodatages restent comparables.
- Permissions des logs restreintes.
- Logs centralisés : la trace survit à la machine compromise. Les moyens (`rsyslog`, RELP, `systemd-journal-remote`) sont dans l'annexe « Gestion avancée et centralisation des logs ».

MODULE 14

Analyse de la charge et de l'activité système

avec les outils standards

De quoi parle-t-on

Mesurer et interpréter la charge et l'activité d'un serveur, avant de songer à optimiser quoi que ce soit : outils instantanés (`top`, `vmstat`, `iostat`...) et collecte historique avec `sysstat`.

Au programme

- charge système : `uptime`, `load average`, PSI,
- outils instantanés : `top/htop`, `vmstat`, `pidstat`, `iostat`, `mpstat`, `free`,
- collecte historique avec `sysstat/sar`, panorama des outils modernes.

Quoi mesurer, et avec quoi

On distingue deux familles d'outils, complémentaires :

- les outils **temps réel** (instantané) : ils échantillonnent l'état courant (`top`, `vmstat`, `iostat`, `pidstat`...) ;
- les outils **historiques** : ils enregistrent en continu pour rejouer un incident passé (`sysstat`/`sar`).

Quel outil pour quelle question ?

	Temps reel (instantane)	Historique (sysstat)
Charge globale	uptime top htop btop /proc/loadavg /proc/pressure	sar -q (LOAD, PSI)
CPU	vmstat mpstat pidstat	sar -u sar -P ALL
Memoire / swap	free vmstat -s pidstat -r	sar -r sar -S sar -W
Disque / I/O	iostat -x pidstat -d iotop	sar -b sar -d -p
Reseau	ss ip -s nicstat iftop	sar -n DEV sar -n TCP

Methode USE : pour chaque ressource, mesurer Utilisation, Saturation et Erreurs.

i Méthode USE

La **méthode USE** (Brendan Gregg) : pour chaque ressource, regarder son **Utilisation**, sa **Saturation** et ses **Erreurs**. C'est un fil conducteur fiable.

uptime et /proc/loadavg

La commande `uptime` donne la **charge système** : le nombre moyen de processus **runnable** (état `R`) ou en **attente d'I/O ininterruptible** (état `D`).

```
$ uptime
18:00:34 up 81 days, 10:20, 3 users, load average: 0.52, 1.98, 3.10
```

Trois moyennes glissantes : **1 min, 5 min, 15 min**. Elles sont **indépendantes du nombre de CPU**.

```
$ cat /proc/loadavg
0.52 1.98 3.10 2/415 33871
```

Champ	Signification
0.52 1.98 3.10	les trois moyennes
2/415	tâches actuellement runnable / tâches existantes
33871	PID du dernier processus créé

 Lire la charge

Un chiffre proche de 0 ne signifie pas "aucun processus" : un processus qui dort en attente utilisateur n'augmente pas la charge.

Interpréter la charge

Le **load average** ne dit pas POURQUOI : CPU saturé, I/O ou les deux ?

Charge élevée + CPU idle élevé ⇒ **suspecter les I/O (état D), pas le CPU.**

Pressure Stall Information

PSI (*Pressure Stall Information*) mesure le **temps perdu** faute de ressource.

Indicateurs : `/proc/pressure/{cpu,memory,io}`. Voir la [doc](#) du noyau.

```
$ cat /proc/pressure/cpu
some avg10=4.31 avg60=2.90 avg300=1.12 total=...
cat /proc/pressure/io
some avg10=27.0 ... full avg10=18.3 ...
```

🔗 Lire les indicateurs

`some` = au moins une tâche bloquée

`full` = toutes les tâches sont bloquées (la machine n'avance plus).

⚠️ PSI désactivé par défaut

Sur AlmaLinux 10, PSI est compilé mais **désactivé** : ajouter `psi=1` à la ligne de commande du noyau (`grubby --update-kernel=ALL --args=psi=1`, puis redémarrer).

top, htop, btop

`top` reste l'outil de premier réflexe.

Touches : `P` (tri CPU), `M` (tri mémoire), `1` (par CPU), `H` (threads), `k` (kill), `r` (renice).

Indicateur (en-tête <code>top</code>)	À surveiller
load average	charge sur 1/5/15 min
%Cpu(s): us sy id wa st	<code>wa</code> (I/O), <code>st</code> (CPU vole en VM)
MiB Mem / MiB Swap	mémoire dispo, swap utilisé
colonne <code>S</code>	état : <code>R</code> run, <code>D</code> I/O, <code>S</code> sleep, <code>Z</code> zombie

Successeurs plus lisibles, à installer au besoin :

- `htop` : couleurs, arborescence, actions à la souris
- `btop` : graphes CPU/mémoire/réseau/disque dans le terminal.

i Origine des paquets

Sur RedHat et ses dérivées (version 10), ces deux outils sont dans **EPEL** : `dnf install epel-release`, puis `dnf install htop btop`.

vmstat : vue d'ensemble

En temps réel

```
$ vmstat -w 1 3
--procs--  -----mémoire-----  -échange-  -----io-----  -système-  -----cpu-----
 r    b      swpd      libre      tampon      cache    si    so    bi    bo    in    cs    us    sy    id    wa    st    gu
  2    0          0    159508    52420    1643368    0    0    280   1366   190    1    1    1    99    0    0    0
  0    0          0    159508    52420    1643368    0    0      0      0   277   152    0    0   100    0    0    0
  0    0          0    159508    52420    1643368    0    0      0      0   280   149    0    0   100    0    0    0
```

 Première ligne

La première ligne est une moyenne depuis le démarrage : on l'ignore.

Groupe	Colonnes
procs	r tâches runnable, b bloquées en I/O
mem	swpd quantité en swap , free , buffers , cache , ou (-a : inactive , active)
swap	si/so pages lues/écrites en swap (alerte si > 0 durablement)
io	bi/bo blocs lus/écrits
system	in interruptions, cs changements de contexte
cpu	us user, sy noyau, id libre, wa attente I/O, KVM : st volé à une VM, gu : temps CPU consommé par les VM

Totalisation (compteurs depuis le boot) : vmstat -s, vmstat -d/-D (par disque).

Charge élevée mais CPU "idle"

Symptôme : load average très haut, alors que la consommation CPU est faible.

```
$ sar -u          # ou : vmstat
              CPU  %user  %nice  %system  %iowait  %steal  %idle
all          0.06   0.00   0.10    99.83   0.00   0.00
```

Le CPU passe son temps en **%iowait** : il attend des entrées/sorties. Les tâches en état **D** gonflent la charge sans consommer de CPU.

Qui engendre les I/O ?

```
$ pidstat -d 1
```

À retenir

Leçon à retenir : un **iowait** élevé n'est pas toujours dû au disque local. Une socket réseau lente (NFS, **rsync** qui sature un lien) produit le même symptôme. Il faut identifier le **coupable**, pas seulement constater l'attente.

pidstat : l'activité par processus

Fourni par `sysstat`, `pidstat` ventile l'activité **par tâche** (là où `vmstat`/`iostat` donnent un agrégat).

```
$ pidstat [-u] [-r] [-d] [-w] [-t] [-p PID] [delai [nombre]]
```

Option	Mesure
<code>-u</code>	CPU par processus (défaut)
<code>-r</code>	mémoire (fautes de page, RSS)
<code>-d</code>	I/O disque par processus (kB_rd/s, kB_wr/s)
<code>-w</code>	changements de contexte (voluntary / involuntary)
<code>-t</code>	détail par thread

```
$ pidstat -u -d 2 3 # CPU + I/O, 3 mesures espacées de 2 s
```

💡 Du symptôme au processus

C'est l'outil pour **passer du symptôme global au processus fautif** sans ouvrir `top` en aveugle.

iostat : CPU et disques

Statistiques étendues, en kB/s, par device

```
$ iostat -x -d -k 2 3
```

Device	r/s	w/s	rkB/s	wkB/s	r_await	w_await	aqu-sz	%util
nvme0n1	12.0	48.0	512.0	6144.0	0.65	1.85	0.42	23.4

Colonne	Lecture
r/s w/s	opérations lecture / écriture par seconde
rkB/s kB/s	débit lecture / écriture
r_await w_await	latence moyenne d'une I/O en lecture / écriture (ms) : indicateur clef
aqu-sz	profondeur moyenne de la file d'attente
%util	d'occupation du device (proche de 100 % = saturé, nuance sur SSD/NVMe)

✎ Unités et SSD

Unités : `-k` (kB/s), `-m` (MB/s). `iostat -c` limite au CPU, `iostat -d` aux disques.
Sur SSD/NVMe, `%util` sature moins vite : privilégier `r_await`, `w_await` et le débit.

mpstat (par CPU) et free (mémoire)

mpstat détaille l'usage **par coeur** : utile pour repérer un seul CPU saturé (mono-thread) ou des interruptions mal réparties.

```
$ mpstat -P ALL 1
  CPU  %usr  %nice %sys  %iowait %irq  %soft  %steal %idle
  all   0.25   0.00 0.25    0.00 0.00   0.00   0.00 99.50
    0    1.00   0.00 0.00    0.00 0.00   0.00   0.00 99.00
```

free : ce qui compte aujourd'hui est la colonne **available** (mémoire réellement allouable), pas **free**.

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:           15Gi       1.1Gi        12Gi       567Mi        2.1Gi        13Gi
Swap:           0B           0B           0B
```

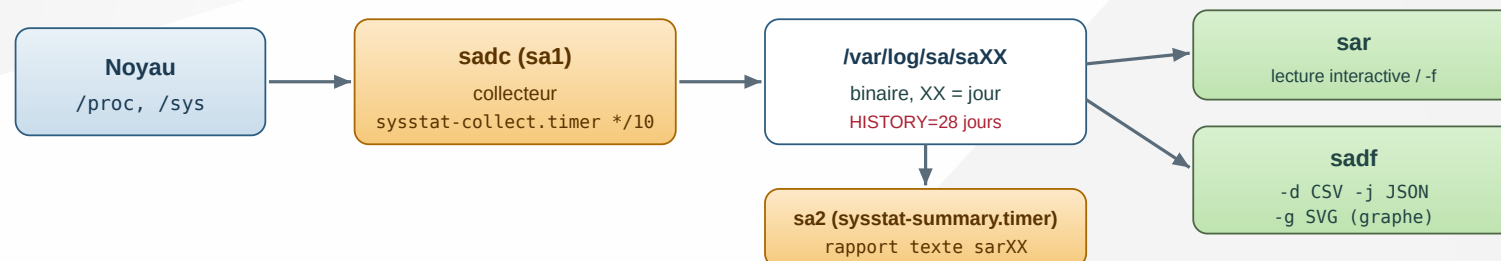
🔗 Le cache n'est pas perdu

buff/cache n'est **pas** de la mémoire perdue : le noyau la rend dès qu'un programme en a besoin. Surveiller **available** et l'usage du **Swap**.

sysstat : enregistrer pour rejouer

Le paquet `sysstat` collecte des mesures en continu et permet de les **rejouer après coup** (analyse d'un incident nocturne, capacity planning).

Chaine sysstat sur RedHat et dérivées 10 (timers systemd)



Activer : `dnf install sysstat ; systemctl enable --now sysstat`

Changer l'intervalle : `systemctl edit sysstat-collect.timer` (override OnCalendar)

Plus de cron : ce sont des timers systemd qui déclenchent la collecte.

```
# dnf install sysstat
# systemctl enable --now sysstat
```

i Collecte par timers

Différence majeure avec l'ancien monde : la collecte n'est plus pilotée par **cron** mais par des **timers systemd** (`sysstat-collect.timer`, `sysstat-summary.timer`). Les fichiers binaires `saXX` vivent dans `/var/log/sa/`, conservés **28 jours** par défaut (`HISTORY` dans `/etc/sysconfig/sysstat`).

sar : exploiter l'historique

sar relit les fichiers et cible une ressource et une plage horaire.

Usage	Affiche
sar -u	CPU (valeur par défaut)
sar -r / -S / -W	mémoire / swap / activité de swap
sar -b / -d -p	I/O globales / par device (noms lisibles)
sar -q / sar -q PSI	run-queue et load average / PSI (si psi=1)
sar -n DEV / -n TCP	réseau par interface / TCP

```

$ sar -u -f /var/log/sa/sa15 -s 01:30:00 -e 01:44:00 # rejouer le 15, plage horaire
  sar -P ALL 2 3                                     # en direct : par CPU, 3 mesures
    
```

sadf exporte vers des formats exploitables (post-traitement, base, graphe) :

```

$ sadf -d sa15 -- -u > cpu.csv      # CSV
  sadf -j sa15 -- -r > mem.json     # JSON
  sadf -g sa15 -- -u > cpu.svg      # graphe SVG
    
```

sadf

sadf est le pont entre sar et un outil de graphe.

Au-delà des classiques

Outil	Apport
atop	comme <code>top</code> , mais enregistre l'historique (rejeu avec <code>atop -r</code>), EPEL
iotop-c	les I/O disque par processus , en direct, AppStream
dstat	flux de colonnes combinables, export CSV : fourni par <code>pcp-system-tools</code>
ss	sockets et connexions (remplace <code>netstat</code>), paquet <code>iproute</code>
iftop	débit réseau par connexion, EPEL
bpftrace, bcc-tools	traçage fin base eBPF (latence I/O, lenteurs noyau), AppStream
pcp-system-tools	versions PCP de <code>atop</code> , <code>htop</code> , <code>iostat</code> , <code>mpstat</code> , <code>pidstat</code> ... (<code>pcp atop</code>), AppStream

💡 Aller plus loin : eBPF

Pour creuser une latence inexpliquée, l'**eBPF** (`bpftrace`) donne accès à des événements noyau qu'aucun compteur agrégé ne montre.

i Hors dépôts

`glances` et `nicstat` ne sont dans aucun des dépôts AlmaLinux et EPEL 10.

Quel outil pour quel symptôme

Symptôme observé	Premier réflexe
Charge élevée, origine inconnue	<code>top</code> / <code>uptime</code> , puis <code>/proc/pressure</code>
CPU saturé	<code>mpstat -P ALL</code> , <code>pidstat -u</code>
<code>iowait</code> élevé	<code>iostat -x</code> , <code>pidstat -d</code> , vérifier le réseau
Mémoire / swap	<code>free -h</code> , <code>vmstat -s</code> , <code>pidstat -r</code>
Réseau	<code>ss</code> , <code>sar -n DEV</code> , <code>iftop</code>
"Que s'est-il passé cette nuit ?"	<code>sar -f /var/log/sa/saXX</code> , <code>atop -r</code>

Démarche en trois temps :

- 1. **Constater** la charge globale (USE : utilisation, saturation, erreurs) ;
- 2. **Localiser** la ressource en tension (CPU, mémoire, I/O, réseau) ;
- 3. **Identifier** le processus ou le flux responsable (`pidstat`, `iotop-c`, `ss`).

 À retenir

Les outils instantanés diagnostiquent l'incident en cours.
`sysstat`/`sar` permet de l'expliquer après coup.
Les deux sont indispensables.

Bonnes pratiques

- Constater, localiser, identifier : dans cet ordre.
- Comparer la charge au nombre de CPU, pas à une valeur fixe.
- `sysstat` activé en avance : l'historique n'existe qu'après.
- Une mesure isolée ne prouve rien : relever plusieurs échantillons.
- Mesurer en dehors de la charge normale, pour avoir un point de comparaison.
- Mesurer coûte : un outil lourd aggrave un système déjà saturé.

CHAPITRE 7

Optimisation des performances

MODULE 15

Optimiser les performances

De quoi parle-t-on

Une fois le goulet d'étranglement identifié, ce module couvre comment agir dessus : disque et scheduler I/O, mémoire, filesystem, processus, démarrage, et réseau.

Au programme

- mesurer avant d'optimiser : la méthode,
- disque (*scheduler I/O*, *filesystem*), mémoire, processus,
- démarrage rapide et performances réseau.

Mesurer avant d'optimiser

⚠ Pas d'optimisation à l'aveugle

Une optimisation commence par une mesure.

Il faut d'abord **identifier le goulet d'étranglement** parmi les quatre grandes ressources.

CPU

top htop mpstat pidstat

Mémoire

free vmstat ps_mem slabtop

Disque (I/O)

fio ioping iostat iotop-c

Réseau

iperf3 ping mtr ss

Une optimisation non mesurée est une superstition.

💡 Une seule modification à la fois

Puis on **re-mesure**.

Tester les performances du disque

D'abord identifier le support : disque rotatif (HDD) ou SSD/NVMe ?

```
$ lsblk -d -o NAME,ROTA,DISC-GRAN,MODEL # ROTA=1 rotatif, 0 = SSD/NVMe
```

fio : l'outil de référence (lecture/écriture aléatoire, 4 Kio, direct)

```
$ fio --name=test --filename=/data/t --size=1G --direct=1 \
    --rw=randrw --bs=4k --numjobs=4 --runtime=30 --time_based
```

ioping : la latence d'accès (*paquet d'EPEL*)

```
$ ioping -c 10 /data
```

Outil	Mesure
fio	débit (MB/s) et IOPS, séquentiel ou aléatoire, lecture/écriture
ioping	latence par requête (proche du temps réel)
dd	mesure séquentielle grossière (dépannage rapide)

 **hdparm**

Sur des machines physiques, `hdparm -tT` donne des informations. Sous virtualisation, il ne mesure presque rien : à écarter.

Choisir le scheduler d'I/O (blk-mq)

Avant le noyau 5.X on choisissait l'*io-scheduler* par défaut (`cfq`, `noop`, `deadline`, `anticipatory`) **au boot** grâce au *kernel-parameter* `elevator=`.

Celui-ci a disparu en version 5.X au profit du **multi-queue block layer** (`blk-mq`).

Lire le scheduler actif d'un disque

```
$ cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber bfq none
```

Scheduler	Usage conseillé
<code>mq-deadline</code>	défaut HDD/SATA, latence bornée, bon compromis serveur
<code>bfq</code>	équité entre processus, bonne réactivité (poste de travail)
<code>kyber</code>	faible surcoût, orienté faible latence
<code>none</code>	NVMe et contrôleurs / hyperviseurs qui ordonnancent déjà

Changer à chaud (volatile)

On choisit disque par disque dans `/sys/block/${DISK}/queue/scheduler`.

```
# echo none > /sys/block/${DISK}/queue/scheduler
```

 **Persistence**

Persistence : on privilégie une **règle udev** (par WWN/série) ou un profil **tuned**.

Choisir le bon filesystem

Défaut de la version 10 : **XFS** sur LVM. **Btrfs**, que RHEL ne prend pas en charge, est proposé par AlmaLinux à l'installation depuis la 10.1.

FS	Points forts	À réserver à
XFS	très bon en parallèle, gros fichiers, gros volumes (défaut RHEL)	serveurs, BDD, stockage volumineux
ext4	mature, faible latence sur petits fichiers, redimensionnable	usage général, <code>/boot</code> , petits volumes
Btrfs	CoW : snapshots, compression, checksums, sous-volumes (<i>AlmaLinux, pas RHEL</i>)	besoin d'intégrité, rollback, instantanés

 Benchmarks

Tendance des benchmarks : **XFS** domine en débit parallèle et gros fichiers, **ext4** garde l'avantage en latence sur de très nombreux petits fichiers, **Btrfs** paie un surcoût d'écriture qu'il compense par ses fonctionnalités.

 Le seul juge est la charge

Rejouer des accès réels avec `fio` avant de trancher.

Régler le filesystem

Options de montage importantes (`/etc/fstab`) :

Option	Effet
<code>noatime</code> / <code>relatime</code>	ne pas (ou peu) écrire la date de dernier accès : gros gain
<code>nodiratime</code>	idem, limite aux répertoires
<code>defaults,noatime</code>	combinaison courante sur les volumes de données

SSD : libérer les blocs (TRIM) de façon périodique (préférable au `discard` continu)

```
# systemctl enable --now fstrim.timer
```

Appliquer un profil global avec tuned

```
$ tuned-adm list # profils disponibles
```

```
# tuned-adm profile throughput-performance
```

 `tuned`

`tuned` règle d'un coup scheduler I/O, paramètres `vm`, gouverneur CPU et réseau selon un profil (`throughput-performance`, `latency-performance`, `virtual-guest`...).

Analyse détaillée de la mémoire

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:          15Gi       3.2Gi       6.1Gi       210Mi         6.4Gi         11Gi
```

La colonne qui compte est **available** : mémoire réellement mobilisable sans swap.
Le cache (**buff/cache**) **n'est pas perdu**, le noyau le récupère au besoin.

Outil	Apport
<code>free -h</code> , <code>/proc/meminfo</code>	vue globale, <code>MemAvailable</code> , swap
<code>vmstat 1</code>	colonnes <code>si/so</code> (swap), <code>bi/bo</code> (I/O), <code>wa</code> (attente I/O)
<code>ps_mem</code> (<i>AppStream</i>)	conso réelle par programme, mémoire privée + part de la mémoire partagée (<code>smem</code> , absent d'EPEL 10)
<code>slabtop</code>	structures noyau (caches d'entries/inodes)

 **RSS et PSS**

RSS (*Resident Set Size*) est la mémoire résidente en RAM d'un processus : trompeur, car il compte plusieurs fois la mémoire partagée.

PSS (*Proportional Set Size*) est la mémoire privée + une part équitable de la mémoire partagée : c'est ce que `ps_mem` additionne par programme.

Identifier les processus inutiles et consommateurs

Outil	Ce qu'on observe
<code>top</code> / <code>htop</code> (EPEL)	CPU, RAM, état, en temps réel et interactif
<code>pidstat 1</code>	CPU / mémoire / I/O par processus dans le temps
<code>iostat -c</code>	quels processus font des I/O disque
<code>vmstat 1</code>	santé globale : run-queue, swap, <code>wa</code>
<code>time cmd</code>	temps real / user / sys d'une commande

Mesurer une commande

```
$ /usr/bin/time -v ./traitement # paquet `time`, détaille le pic mémoire (RSS max)
```

Abaissier la priorité d'un gros calcul (`nice` : -20 prioritaire ... +19 gentil)

```
$ nice -n 19 ./batch_nocturne # au lancement
renice -n 10 -p 4321 # sur un PID existant
```

 `ionice`

`ionice` fait de même pour la **priorité d'I/O** (utile avec le scheduler `bfq`).

Comprendre le vocabulaire

Terme	Définition
Thread	fil d'exécution partageant l'espace mémoire du processus
Daemon	processus de service en arrière-plan (souvent enfant de PID 1)
Zombie (<code>Z</code> , <i>defunct</i>)	processus terminé dont le parent n'a pas lu le code de retour (<code>wait()</code>)
Orphelin	processus dont le parent est mort, adopté par systemd (PID 1)

Zombies

Un zombie ne consomme ni CPU ni RAM, juste une entrée dans la table : trop de zombies trahit un parent qui ne fait pas son `wait()`.

Respawn / xinetd / activation par socket

Autrefois :

- SysV `respawn` dans `inittab`, relançait un service qui s'arrêtait
- `xinetd` (re)lançait un service **à la demande** sur connexion.

Relancer automatiquement un service (l'équivalent de respawn)

```
# /etc/systemd/system/mon-service.service
[Service]
Restart=always
RestartSec=2
```

`xinetd` n'existe plus sur RedHat et ses dérivées : tout passe par **systemd**.

Lancer à la demande : l'activation par socket

```
# mon-service.socket : systemd écoute, ne démarre le service qu'à la 1ère connexion
[Socket]
ListenStream=12345
Accept=no
```

```
# systemctl enable --now mon-service.socket
```

Zéro processus

Zéro processus si aucun client connecté, démarrage instantané à la sollicitation.

Booter rapidement son système

Outils : `systemd-analyze` décortique le temps de démarrage.

```
$ systemd-analyze          # total : firmware + loader + kernel + userspace
systemd-analyze blame      # services triés par durée
systemd-analyze critical-chain # chaîne critique réellement bloquante
systemd-analyze plot > boot.svg # chronogramme visuel
```

⚠ blame

Attention à `blame` : un service long n'est gênant que s'il est sur la **chaîne critique**. C'est `critical-chain` qui désigne le vrai coupable.

Désactiver ce qui ne sert pas (la meilleure optimisation reste de ne pas lancer)

```
# systemctl disable --now service-inutile
systemctl mask service-a-bannir      # interdiction stricte
```

Tester les performances du réseau

Besoin	Outil
Débit entre deux hôtes	<code>iperf3 -s</code> (serveur) / <code>iperf3 -c <ip></code> (client)
Latence	<code>ping</code> (RTT), <code>mtr <hote></code> (latence + perte saut par saut)
Connexions et files d'attente	<code>ss -s</code> , <code>ss -tan</code>
Statistiques d'interface	<code>ip -s link</code>

Cache DNS : éviter de (re-)résoudre en permanence

```
$ resolvectl statistics # taux de hit du cache, si systemd-resolved est actif
```

Cache local	Quand
<code>systemd-resolved</code>	paquet à part, non activé par défaut : NetworkManager écrit <code>/etc/resolv.conf</code>
<code>dnsmasq</code> , <code>unbound</code>	cache résolveur dédié, pour des réseaux chargés

 Cache DNS

Un cache DNS local supprime des allers-retours réseau : effet spectaculaire sur les applications qui résolvent beaucoup (mail, web, proxies).

MTU, fenêtres et files d'émission

MTU : taille max d'une trame. Les *jumbo frames* (9000) aident sur un LAN 10G+ homogène.

```
# ip link set dev enp1s0 mtu 9000
```

File d'émission de l'interface (txqueuelen)

```
# ip link set dev enp1s0 txqueuelen 10000
```

Fenêtres TCP : l'autotuning fait le gros du travail, on relève surtout les plafonds sur les liens rapides à forte latence :

```
≡ net.ipv4.tcp_window_scaling = 1          # Window Scale (RFC 7323) indispensable sur lien à gros BDP
  net.core.rmem_max = 134217728          # plafond buffer reception
  net.core.wmem_max = 134217728          # plafond buffer emission
  net.ipv4.tcp_rmem = 4096 131072 67108864 # min / défaut / max (auto)
  net.ipv4.tcp_congestion_control = bbr    # algo moderne (vs cubic, le défaut)
  net.core.default_qdisc = fq              # file d'attente conseillée avec bbr
```

💡 Persister

À persister dans `/etc/sysctl.d/*.conf`. `bbr` est un module du noyau (`tcp_bbr`) : `sysctl net.ipv4.tcp_available_congestion_control` liste les algorithmes disponibles.

Sur un LAN sain, l'autotuning suffit en général : mieux vaut le laisser en place.

Les outils standards d'analyse

Famille	Outils	Pour
sysstat	<code>sar</code> , <code>iostat</code> , <code>mpstat</code> , <code>pidstat</code>	historique et instantané CPU / I/O / réseau
mémoire / vm	<code>free</code> , <code>vmstat</code> , <code>ps_mem</code> , <code>slabtop</code>	RAM, swap, caches
temps réel	<code>top</code> , <code>htop</code> (EPEL), <code>iotop-c</code>	vue interactive globale
profilage	<code>perf</code> , <code>bpftrace</code> (eBPF)	descendre au niveau fonction / appel système
supervision	PCP (Performance Co-Pilot)	métrologie continue, archivage

Activer la collecte historique (sysstat)

```
# systemctl enable --now sysstat
```

Consulter les traces

```
$ sar -u 1 10      # CPU
  sar -r 1 10      # mémoire
  sar -d 1 10      # disque
```

D'abord une vue large (`sar`, `top`), puis on zoome (`pidstat`, `iotop-c`), enfin on profile (`perf`, `bpftrace`) si nécessaire.

Bonnes pratiques

- Mesurer avant, mesurer après.
- Un réglage à la fois, noté avec sa valeur d'origine.
- Test de disque sur un fichier, pas sur un périphérique en production.
- Scheduler d'I/O adapté au support (SSD, disque mécanique).
- Services inutiles désactivés.
- Un réglage qui ne corrige aucune mesure ne se garde pas.

CHAPITRE 8

Supervision

MODULE 16

Collecter et grapher des métriques système

De quoi parle-t-on

Passer de la supervision ponctuelle à une vraie métrologie : la solution native de RedHat et de ses dérivées (*Cockpit* + *PCP*), puis la pile de référence Prometheus/Grafana, et un panorama des alternatives.

Au programme

- Cockpit + PCP : la solution native, sans serveur dédié,
- la pile Prometheus : node_exporter, PromQL, Grafana, alerting,
- panorama des alternatives (*Netdata*, *VictoriaMetrics*, *Zabbix*, *OpenTelemetry*).

Cockpit + PCP : le tableau de bord intégré

Cockpit est la console web d'administration de RedHat et de ses dérivées. Couplée à **Performance Co-Pilot (PCP)**, elle affiche les **métriques historiques** sans serveur de métrologie dédié.



```
# dnf install cockpit pcp-zeroconf
systemctl enable --now pmcd pmlogger
systemctl enable --now cockpit.socket           # https://serveur:9090
```

Pour **une** machine ou un petit parc, Cockpit + PCP suffit et ne demande presque aucune configuration.

- page **Metrics** : CPU, mémoire, disque, réseau, en direct et sur l'historique PCP ;
- corrélation immédiate avec les pages **Services**, **Logs**, **Conteneurs**.

Pour une **flotte**, on bascule vers Prometheus ou Grafana + PCP centralisé.

i `cockpit-pcp` selon la version

Sur AlmaLinux 9.4 et antérieures, `cockpit-pcp` est nécessaire à la page **Metrics**. Depuis Cockpit 326 (AlmaLinux 10), `pcp-zeroconf` suffit.

Changement de paradigme

Cacti (SNMP + RRDTool) a longtemps fait référence. Le modèle a montré ses limites : résolution figée à l'avance, peu de dimensions, alerting pauvre, polling SNMP lourd.

Avant : Cacti / RRDTool (poll SNMP, base cyclique)



Aujourd'hui : métriques en séries temporelles (pull, TSDB, labels)



Chaque métrique porte des labels (host, device, mode...) : requêtes multidimensionnelles, alerting riche.

Prometheus / Grafana / Alertmanager change la logique du monitoring :

- **Base de séries temporelles (TSDB)** : ingestion de métriques **étiquetées**
 - hôte, service, environnement
- **Outils d'exploitation** : visualisation, tableaux de bord et alerting
 - **interrogent la base à la demande**
- Hier on préconfigurait chaque graphe...
 - Aujourd'hui les vues se construisent **a posteriori**, par requêtes

Métrique ?

C'est une **série temporelle** : un nom, des **labels**, et des couples (instant, valeur).

```
≡ node_cpu_seconds_total{instance="srv-1",cpu="0",mode="idle"} 1.83e6 @ 1717400000
```

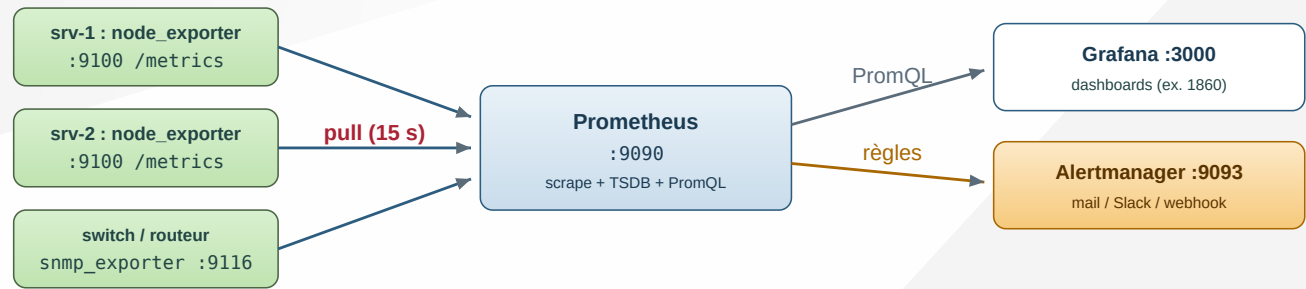
Notion	RRDTool / Cacti	Modèle actuel
Stockage	base cyclique RRD	TSDB (rétention configurable)
Identité	un fichier par courbe	nom + labels
Acquisition	poll SNMP planifié	pull sur un endpoint <code>/metrics</code> (ou push)
Restitution	graphes pré-définis	requêtes (PromQL) + dashboards
Alerting	plugin limité	règles natives + notifications

 Labels

Les labels rendent les requêtes **multidimensionnelles** : "le `%idle` CPU de tous les serveurs du datacenter B", sans créer un graphe par machine.

Architecture Prometheus

La pile de référence, libre et omniprésente (projet CNCF) : Prometheus **3.x**, node_exporter **1.12**, Grafana **10.2** (AppStream) ou **12.x** (en amont).



Prometheus va chercher les métriques (pull), les cibles n'ont qu'à exposer /metrics.
Sécurité : n'exposer le port :9100 qu'au serveur Prometheus (firewall).

Composant	Rôle	Port
node_exporter	expose les métriques de l'hôte	9100
Prometheus	scrute (pull), stocke (TSDB), évalue PromQL	9090
Grafana	dashboards et exploration	3000
Alertmanager	route et notifie les alertes	9093

i Où trouver les paquets

grafana est dans AppStream. prometheus, alertmanager et node-exporter sont dans EPEL 10.

Déployer `node_exporter` sur la cible

Sur **chaque** machine à superviser, on installe l'agent qui expose `/metrics`.

RedHat et dérivées, version 10 (paquet d'EPEL) :

```
# sudo dnf install epel-release
sudo dnf install node-exporter
sudo systemctl enable --now node_exporter
```

Vérifier ce qui est exposé

```
$ curl -s http://localhost:9100/metrics | head
node_cpu_seconds_total{cpu="0",mode="idle"} 1.83e+06
node_memory_MemAvailable_bytes 1.36e+10
node_filesystem_avail_bytes{mountpoint="/"} 4.2e+10
```

⚠ Port 9100

Ouvrir le port `9100` **uniquement** vers le serveur Prometheus :

```
# firewall-cmd --permanent \
--add-rich-rule='rule family=ipv4 source address=IP-PROM/32 port port=9100 protocol=tcp accept'
firewall-cmd --reload
```

Configurer la collecte

Prometheus va **chercher** les métriques auprès des cibles, à intervalle régulier.

```
# /etc/prometheus/prometheus.yml
global:
  scrape_interval: 15s

  scrape_configs:
    - job_name: "noeuds"
      static_configs:
        - targets:
            - "srv-1.actilis.net:9100"
            - "srv-2.actilis.net:9100"
```

```
# systemctl enable --now prometheus      # TSDB locale dans /var/lib/prometheus
```

La **rétenion** (15 jours par défaut) et les autres options se règlent dans `/etc/default/prometheus` :

```
# ARGS='--storage.tsdb.retention.time=30d'
```

⚠ Le port 9090 est aussi celui de Cockpit

Sur la même machine, déplacer Prometheus dans `ARGS` : `--web.listen-address=0.0.0.0:9091`.

La page `/targets` confirme l'état de chaque *scrape*.

PromQL : du compteur à l'indicateur

La plupart des métriques sont des **compteurs cumulatifs** : on en dérive un **taux** avec `rate()`.

```
≡ # % CPU non-idle, par machine
100 * (1 - avg by (instance)(rate(node_cpu_seconds_total{mode="idle"}[5m])))

# % de mémoire disponible
100 * node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes

# espace disque libre sous 10 % sur la racine
node_filesystem_avail_bytes{mountpoint="/"}
/ node_filesystem_size_bytes{mountpoint="/"} < 0.10
```

🔗 `rate()` et agrégation

`rate()` calcule la pente sur une fenêtre.

`avg by (label)` agrège en gardant une dimension. C'est l'équivalent dynamique des "graphes" figés de Cacti.

Grafana et le dashboard 1860

Grafana se branche sur Prometheus comme **source de données**, puis affiche des tableaux de bord.

1. **Connections** puis **Add data source** : type Prometheus, URL `http://localhost:9090` (`9091` si Cockpit est sur la même machine).
2. **Dashboards** puis **Import**, saisir l'ID **1860** ("Node Exporter Full").
3. Sélectionner la source : CPU, mémoire, disque, réseau sont prêts, sans construire un seul panneau.

🔗 Dashboard 1860

Le dashboard **1860** est le plus téléchargé de Grafana (10M+). On part de là, puis on adapte. Au-delà : panneaux sur mesure en PromQL, et alertes définies directement dans Grafana ou dans Prometheus.

Des règles d'alerte

L'alerting remplace la surveillance manuelle des graphes.

```
# règle Prometheus
groups:
- name: noeud
  rules:
  - alert: DisqueRacinePresquePlein
    expr: node_filesystem_avail_bytes{mountpoint="/" }
        / node_filesystem_size_bytes{mountpoint="/" } < 0.10
    for: 10m
    labels:
      severity: warning
    annotations:
      summary: "Moins de 10 % libres sur / ({{ $labels.instance }})"
```

🔗 Durée de l'alerte

`for: 10m` évite les fausses alertes sur un pic transitoire. **Alertmanager** se charge ensuite du routage (mail, Slack, webhook), du regroupement et des périodes de silence.

Les autres solutions

Solution	Pour qui / quand
Netdata	un serveur, visibilité immédiate, zéro configuration
VictoriaMetrics	TSDB compatible Prometheus, gros volumes, longue rétention
Zabbix	supervision classique agent + SNMP, découverte, alerting intégré
collectd	agent à plugins déjà présent sur un parc : expose ses mesures à Prometheus (plugin <code>write_prometheus</code>), paquets dans EPEL 10
OpenTelemetry	standard émergent unifiant métriques, logs et traces
snmp_exporter	faire entrer les équipements SNMP (switchs, onduleurs) dans Prometheus

i SNMP

SNMP n'est pas mort : pour le matériel réseau qui ne peut pas héberger d'*exporter*, **snmp_exporter** traduit le SNMP en métriques Prometheus. On garde l'héritage de Cacti tout en unifiant la collecte.

Quelle stack pour quel besoin ?

Contexte	Choix raisonnable
1 serveur, rapidement up, out of the box	Cockpit + PCP, ou Netdata
Parc de serveurs	Prometheus + node_exporter + Grafana
Équipements réseau	+ snmp_exporter
Très gros volumes / longue rétention	VictoriaMetrics
Métriques + logs + traces	OpenTelemetry en collecteur

📋 Principe

Collecter largement, requêter à la demande, alerter sur l'essentiel. C'est l'inverse de l'ancien modèle où chaque graphe devait être prévu d'avance.

Bonnes pratiques

- Collecter large, alerter sur l'essentiel.
- Exporters accessibles au seul collecteur (pare-feu).
- Cardinalité des labels maîtrisée.
- Rétention définie dès le départ.
- Configuration et dashboards versionnés.
- Alertes avec seuil et durée : pas de bruit pour un pic isolé.

Annexes

GRUB 2 et l'UEFI

De quoi parle-t-on

Sur une machine UEFI, GRUB 2 ne se loge plus dans le MBR : il vit dans une partition FAT, derrière un programme signé. Cette annexe suit la chaîne de démarrage UEFI, puis montre comment réinstaller et dépanner GRUB 2 dans ce mode.

Au programme

- l'ESP, `shim`, GRUB 2 et Secure Boot,
- la configuration de GRUB 2 en UEFI,
- réinstaller et dépanner en UEFI.

UEFI : l'ESP, shim et GRUB

Le firmware **UEFI** ne lit pas de secteur d'amorçage : il lance un programme `.efi` rangé dans l'**ESP** (*EFI System Partition*), une partition **FAT** montée sur `/boot/efi`.

Fichier de l'ESP	Rôle
<code>/boot/efi/EFI/almalinux/shimx64.efi</code>	shim : premier programme lancé, signé, il vérifie la signature de GRUB
<code>/boot/efi/EFI/almalinux/grubx64.efi</code>	GRUB 2 , signé par la distribution
<code>/boot/efi/EFI/BOOT/BOOTX64.EFI</code>	chemin de repli que le firmware essaie sans entrée dans sa mémoire

Le firmware retient dans sa mémoire (NVRAM) une liste d'entrées de démarrage :

```
# efibootmgr -v
```

i Secure Boot

Avec **Secure Boot**, le firmware ne démarre que des programmes signés : `shim`, puis GRUB, puis le noyau. Un noyau compilé soi-même est refusé tant qu'il n'est pas signé. `mokutil --sb-state` indique l'état.

La configuration en UEFI

- `/boot/grub2/grub.cfg` reste le fichier de configuration, **le même qu'en BIOS** : `grub2-mkconfig -o /boot/grub2/grub.cfg`.
- `/boot/efi/EFI/almalinux/grub.cfg`, dans l'ESP, n'est qu'un petit fichier de redirection (produit par `gen_grub_cfgstub`).
- Le script `30_uefi-firmware` ajoute au menu l'entrée qui redémarre dans la configuration du firmware. La commande équivalente est `systemctl reboot --firmware-setup`.

💡 Reconnaître le mode

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

 indique le mode dans lequel la machine a démarré.

Réinstaller GRUB 2 en UEFI

Les fichiers de l'ESP viennent de **paquets** : on les réinstalle plutôt que de les reconstruire.

```
# dnf reinstall shim-x64 grub2-efi-x64  
grub2-mkconfig -o /boot/grub2/grub.cfg
```

⚠ Pas de `grub2-install` en UEFI

Sur une machine UEFI, `grub2-install` ne remet pas les fichiers signés de l'ESP.

💡 Entrée du firmware perdue

```
efibootmgr -c -d /dev/sda -p 1 -L AlmaLinux -l  
'\EFI\almalinux\shimx64.efi'
```

recrée l'entrée de démarrage (disque et numéro de partition de l'ESP à adapter).

Dépanner en UEFI

Depuis l'image ISO d'installation :

- Booter l'ISO, puis **Troubleshooting** puis **Rescue a AlmaLinux system**.
- À la proposition de montage : *Continue*, puis *OK*. Le système est monté dans `/mnt/sysroot`.

```
# loadkeys fr
chroot /mnt/sysroot
findmnt /boot/efi
dnf reinstall shim-x64 grub2-efi-x64
grub2-mkconfig -o /boot/grub2/grub.cfg
```

`findmnt /boot/efi` vérifie que l'ESP est montée avant la réinstallation.

i Réseau en rescue

`dnf reinstall` demande un accès aux dépôts. En rescue, la configuration réseau reste à activer à la main si l'installateur ne l'a pas fait.

Bonnes pratiques

- UEFI : réinstaller `shim-x64` et `grub2-efi-x64`, pas `grub2-install`.
- ESP montée avant toute réparation (`findmnt /boot/efi`).
- Secure Boot actif : noyau et modules signés (`mokutil --sb-state`).
- Entrée du firmware contrôlée après une réparation (`efibootmgr -v`).

GRUB 2 : configuration avancée et sécurité

De quoi parle-t-on

Au-delà du menu, GRUB 2 se configure par des fichiers générés et des entrées par noyau. Cette annexe montre comment le configurer, ajouter ou modifier une entrée, puis comment protéger le menu par un mot de passe.

Au programme

- `grub.cfg`, `/etc/default/grub` et les entrées BLS,
- `grubby` : entrée par défaut, nouveau noyau, options du noyau,
- une entrée personnalisée, puis le mot de passe de GRUB 2.

GRUB 2 sur RHEL 10 : fichiers et entrées

- `/boot/grub2/grub.cfg`, en BIOS comme en UEFI, produit par `grub2-mkconfig` à partir de `/etc/default/grub` et de `/etc/grub.d`.
- **Une entrée de menu par noyau**, dans un fichier **BLS** de `/boot/loader/entries/` : GRUB le relit à chaque démarrage, changer une entrée ne demande pas de régénérer `grub.cfg`.
- `grubby` modifie les entrées BLS directement.

✎ `/boot` hors LVM

`/boot` ne peut pas être un volume LVM : c'est une partition à part. 1 Go suffit pour quelques noyaux (environ 200 Mo par noyau installé), la documentation Red Hat recommande 2 GiB.

Le fichier de configuration

Le fichier lu par GRUB 2 est **dynamiquement créé**. Son édition directe n'est pas recommandée.

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

Cette commande s'appuie sur `/etc/default/grub` et sur les scripts de `/etc/grub.d` :

Script de <code>/etc/grub.d</code>	Produit
<code>00_header</code>	réglages généraux du menu
<code>01_users</code>	utilisateur et mot de passe de GRUB (voir slide « Sécuriser le menu GRUB 2 »)
<code>10_linux</code>	une entrée par noyau installé
<code>30_os-prober</code> , <code>30_uefi-firmware</code>	autres systèmes, entrée de configuration du firmware
<code>40_custom</code> , <code>41_custom</code>	entrées personnelles

/etc/default/grub

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg00/lv_root rd.lvm.lv=vg00/lv_swap rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

Directive	Rôle
GRUB_TIMEOUT	durée du compte à rebours du menu
GRUB_DEFAULT	entrée par défaut (numéro, identifiant, ou saved)
GRUB_CMDLINE_LINUX	options passées à la ligne de commande du noyau
GRUB_ENABLE_BLSCFG	true : les entrées sont des fichiers BLS
GRUB_TERMINAL_OUTPUT	sortie du menu (console, gfxterm...)

D'autres paramètres figurent selon l'installation (crashkernel=, resume=...).

Les entrées BLS

Un fichier par noyau installé, nommé `<machine-id>-<version du noyau>.conf` :

```
# ls /boot/loader/entries/  
  
≡ title AlmaLinux (6.12.0-211.7.4.el10_2.x86_64) 10.2  
version 6.12.0-211.7.4.el10_2.x86_64  
linux /vmlinuz-6.12.0-211.7.4.el10_2.x86_64  
initrd /initramfs-6.12.0-211.7.4.el10_2.x86_64.img  
options root=/dev/mapper/vg00-lv_root ro rd.lvm.lv=vg00/lv_root rhgb quiet  
grub_users $grub_users  
grub_arg --unrestricted  
grub_class kernel
```

Champ	Rôle
title	texte du menu
linux, initrd	noyau et initramfs, relatifs à /boot
options	ligne de commande du noyau

grubby lit et modifie ces fichiers :

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

Le choix par défaut dans le menu

`GRUB_DEFAULT=saved` : le choix par défaut est mémorisé dans l'*Environment Block* `/boot/grub2/grubenv`, sans régénérer `grub.cfg`.

```
# grub2-editenv list
grubby --default-kernel
grubby --default-index
```

```
# grubby --set-default=/boot/vmlinuz-6.12.0-211.7.4.el10_2.x86_64
grubby --set-default-index=1
```

`GRUB_DEFAULT` accepte aussi le **numéro** d'une entrée (à partir de 0) ou son **identifiant**.
`grub2-set-default` fait le même travail que `grubby --set-default`.

💡 Repérer les entrées

`grubby --info=ALL` liste les entrées avec leur `index`, leur `title` et leur `id`.

Ajouter son propre noyau

Un noyau installé par un paquet reçoit son entrée BLS automatiquement. Pour un noyau ajouté à la main, `grubby` crée l'entrée :

```
# cd /boot
cp vmlinuz-6.12.0-211.7.4.el10_2.x86_64      vmlinuz-test
cp initramfs-6.12.0-211.7.4.el10_2.x86_64.img  initramfs-test.img
grubby --add-kernel=/boot/vmlinuz-test --initrd=/boot/initramfs-test.img \
      --title="Noyau de test" --copy-default
```

Vérification :

```
# grubby --info=ALL | grep title
```

L'entrée `Noyau de test` apparaît au prochain démarrage, dans le menu GRUB.

🔧 Noyau compilé

`make install` sait aussi créer cette entrée (voir slide « Étapes 4 et 5 : installer »).

Modifier une option de démarrage du noyau

Retirer `rhgb` et `quiet` pour voir les messages de démarrage, sur **toutes** les entrées :

```
# grubby --update-kernel=ALL --remove-args="rhgb quiet"
```

Après le reboot :

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

L'inverse, ajouter une option : `--args="option"`.

i `grubby` ou `/etc/default/grub` ?

`grubby` modifie les fichiers BLS et pas `/etc/default/grub`. Dans l'autre sens, modifier `GRUB_CMDLINE_LINUX` demande de propager le changement aux entrées existantes : `grub2-mkconfig -o /boot/grub2/grub.cfg --update-bls-cmdline`.

⚠ Toutes les entrées

`--update-kernel=ALL` change tous les noyaux : une option erronée touche aussi les noyaux de secours.

Modifier juste une entrée

Pour une seule entrée, `grubby` désigne le noyau :

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="systemd.unit=multi-user.target"
```

Pour ajouter une entrée qui n'est pas un noyau Linux (autre système, outil), on écrit un `menuentry` dans `/etc/grub.d/40_custom`, puis on régénère `grub.cfg` :

```
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.
menuentry 'Entrée personnalisée' {
    ...
}
```

✎ Pourquoi à partir de la 3e ligne

C'est la **seconde ligne** (`exec tail -n +3 $0`) qui fait intégrer la 3e et les suivantes à `grub.cfg`, au prochain `grub2-mkconfig`.

Sécuriser le menu GRUB 2

```
# grub2-setpassword
```

La commande demande un mot de passe, puis range son hachage PBKDF2 dans `/boot/grub2/user.cfg` :

```
GRUB2_PASSWORD=grub.pbkdf2.sha512.10000.6A839D73...DEA59
```

Le script livré `/etc/grub.d/01_users` lit ce fichier et déclare l'utilisateur `root` de GRUB 2 comme *superuser*. Après `grub2-mkconfig`, modifier une entrée (`e`) ou ouvrir la ligne de commande (`c`) demande cet utilisateur et ce mot de passe.

Démarrer une entrée du menu reste libre : les entrées portent `--unrestricted` (voir `grub_arg` des entrées BLS).

D'autres utilisateurs

`grub2-mkpasswd-pbkdf2` produit un hachage pour `password_pbkdf2 utilisateur` hachage, à écrire dans un script de `/etc/grub.d`. Une clause `menuentry` accepte alors `--users utilisateur`.

Limites de cette protection

Elle n'empêche pas de lire le disque depuis un autre support. Un accès physique appelle aussi le chiffrement du disque (LUKS) et un mot de passe du firmware.

Bonnes pratiques

- `grubby` pour modifier une entrée, plutôt qu'un `grub.cfg` édité à la main.
- Après `/etc/default/grub` : `grub2-mkconfig ... --update-bls-cmdline`.
- Conserver plusieurs noyaux installés, pour avoir un repli.
- `/boot` hors LVM, de 1 Go au moins pour garder plusieurs noyaux.
- Menu protégé par `grub2-setpassword`.
- Disque chiffré (LUKS) et mot de passe du firmware si l'accès physique est un risque.
- Essayer `rd.break` sur une VM de test, avant et après la protection du menu.

Gestion avancée et centralisation des logs

De quoi parle-t-on

Le module « Les logs » s'arrête au journal de `systemd`. Cette annexe ajoute `rsyslog`, pour écrire des fichiers texte ou envoyer les messages vers un autre serveur, puis `logrotate`, qui fait tourner ces fichiers. Elle va ensuite plus loin dans la centralisation : formats et transports de syslog, **RELP**, envoi vers **Graylog**, `systemd-journal-upload` et `systemd-journal-remote`, backends de logs.

Au programme

- `rsyslog` : modules d'entrée (`im*`), règles, modules de sortie (`om*`), centralisation simple,
- `logrotate` : la rotation des fichiers de log,
- formats et transports de syslog, RELP, SELinux et pare-feu,
- Graylog, `journal-upload` et `journal-remote`, backends.

rsyslog : quand l'installer ?

`rsyslog` est le « serveur syslog » historique : il reçoit des messages, les trie avec des règles, puis les écrit dans des fichiers ou les envoie ailleurs.

Il n'est plus indispensable : `journald` suffit à collecter et à consulter les logs.

`rsyslog` devient utile pour :

- obtenir des **fichiers texte** dans `/var/log` (`messages`, `secure`...),
- **envoyer** les messages vers un serveur central,
- appliquer des **règles de tri** précises.

```
$ rpm -q rsyslog
```

```
# dnf install rsyslog  
systemctl enable --now rsyslog
```

i Rien de grave si `/var/log` est presque vide

- Sans `rsyslog`, `/var/log/messages` et `/var/log/secure` n'existent pas.
- `journalctl` est alors le seul moyen de lire les logs.
- Avec `rsyslog`, les messages sont conservés deux fois, dans le journal et dans les fichiers texte.

Comment rsyslog reçoit les messages

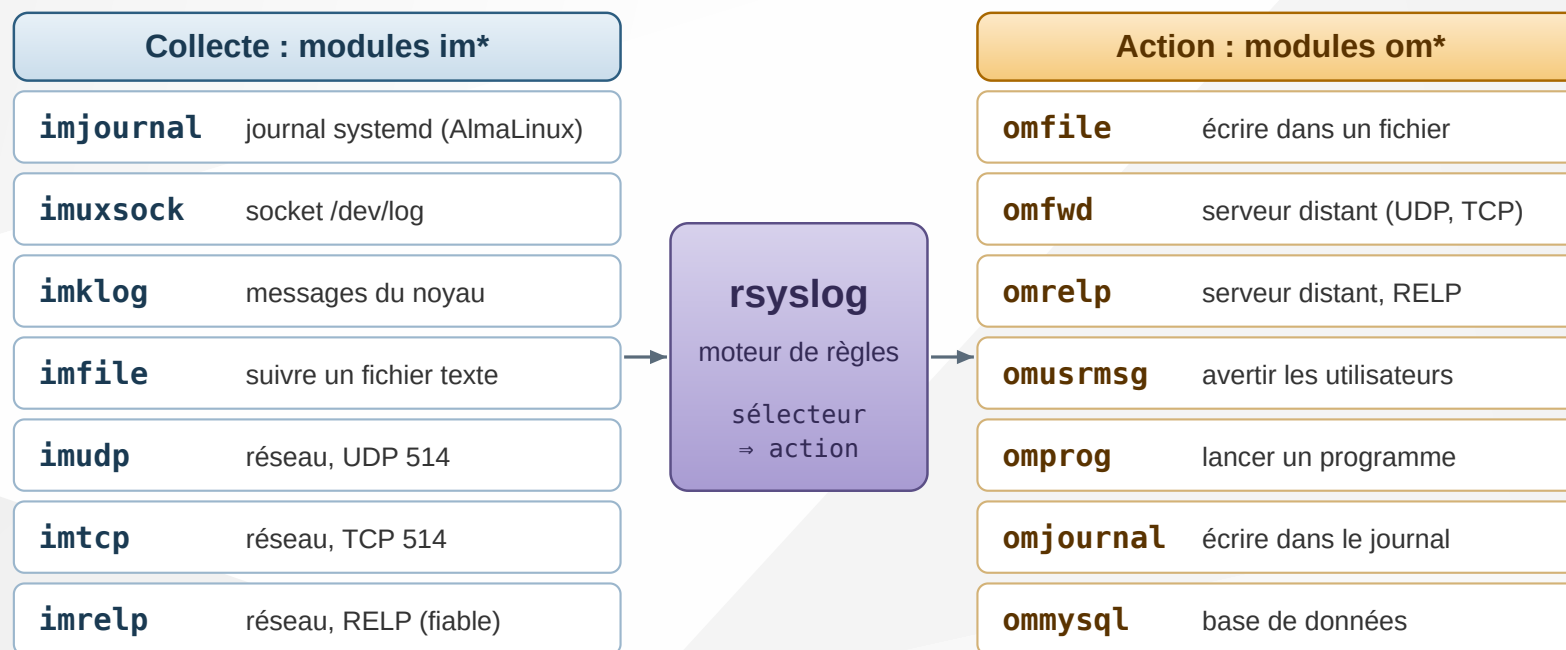
On peut alimenter `rsyslog` depuis *le journal* de deux façons

1. **Push** : `journal`d collecte `/dev/log` et fait suivre (`ForwardToSyslog=yes`) à `rsyslog` qui reçoit grâce à son module `imuxsock`.
2. **Pull** : `rsyslog` consomme directement le journal via [l'API sd-journal](#) grâce à son module `imjournal`, en gardant sa position dans un fichier d'état (`/var/lib/rsyslog/imjournal.state`).

i Sur la version 10

C'est la seconde solution qui est câblée dans la configuration par défaut.

Les modules de rsyslog



im = input, om* = output : chargés par module(load=...) dans /etc/rsyslog.conf*

Modules et paquets

Les modules courants sont dans le paquet `rsyslog`.

D'autres viennent de paquets séparés, comme `rsyslog-relp` (`imrelp`, `omrelp`) ou `rsyslog-mysql` (`ommysql`).

Configuration : `/etc/rsyslog.conf`

Le fichier principal inclut ceux de `/etc/rsyslog.d/`.

Il contient trois familles de directives :

- **modules** et paramètres globaux : `module( load="..." )`,
- **templates** : format des lignes, noms des fichiers,
- **règles** : sélecteur $\Rightarrow$ action.

```
# Extrait de /etc/rsyslog.conf (AlmaLinux)
module(load="imuxsock" SysSock.Use="off")
module(load="imjournal" UsePid="system" FileCreateMode="0644"
        StateFile="imjournal.state")
include(file="/etc/rsyslog.d/*.conf" mode="optional")
```

`imuxsock` est chargé mais n'écoute pas `/dev/log` (`SysSock.Use="off"`).

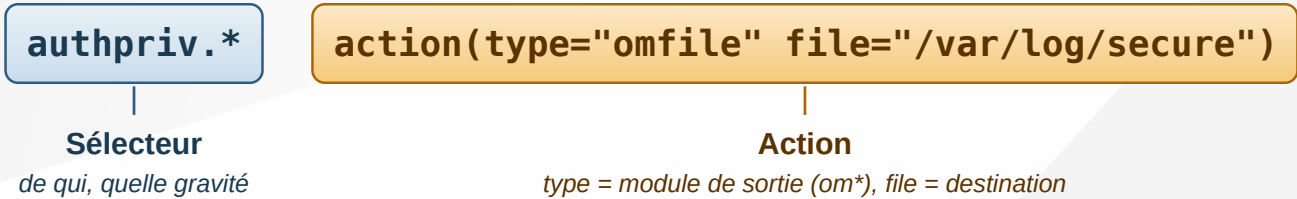
`imjournal` lit le journal systemd.

💡 Vérifier avant de redémarrer

`rsyslogd -N1` contrôle la syntaxe de la configuration sans démarrer le service.

Règles : sélecteur ⇒ action

Anatomie d'une règle rsyslog



Sélecteur	Signification
mail.info	facility mail, niveau info et plus grave
mail.=info	uniquement le niveau info
mail,cron.err	facility mail ou cron, niveau err et plus grave
.info	toutes les facilities (est un joker)
*.info;mail.none	tout, sauf mail (; combine, none exclut)

i Deux écritures pour une action

Les anciennes configurations : *.info /var/log/messages.

La syntaxe action(type="omfile" ...) fait la même chose, mais la syntaxe est plus souple pour les modules nécessitant des paramètres.

Les règles fournies : quels fichiers ?

Les règles de `/etc/rsyslog.conf` expliquent le contenu de `/var/log` :

Sélecteur	Fichier	Contenu
<code>*.info;mail.none;authpriv.none;cron.none</code>	<code>messages</code>	messages généraux
<code>authpriv.*</code>	<code>secure</code>	authentification (<i>accès restreint</i>)
<code>mail.*</code>	<code>maillog</code>	messaging
<code>cron.*</code>	<code>cron</code>	tâches planifiées
<code>local7.*</code>	<code>boot.log</code>	messages de démarrage

D'autres fichiers sont écrits par les programmes eux-mêmes :

- `audit/audit.log` (`auditd`),
- `dnf.log` (`dnf`),
- `httpd/` (Apache),
- et `btmpt/wtmp`, binaires lus avec `lastb` et `last`.

Écrire sa propre règle

Une application maison peut émettre sur une facility libre, de `local0` à `local7`. Il suffit d'une règle qui l'envoie vers un fichier :

```
# /etc/rsyslog.d/50-monappli.conf
local0.*  action(type="omfile" file="/var/log/monappli.log")
```

```
$ sudo rsyslogd -N1
sudo systemctl restart rsyslog
logger -p local0.info -t MONAPPLI 'test de la règle'
tail -n 1 /var/log/monappli.log
```

```
≡ Sep 19 13:40:12 alma10 MONAPPLI[3421]: test de la règle
```

💡 Des facilities pour les applications

`local0` à `local7` sont réservées aux usages qui n'avaient pas été prévus.

Centraliser : envoyer vers un serveur

Deux rôles, deux modules : le **serveur** reçoit avec `imtcp`, le **client** envoie avec `omfwd`.

```
# Sur le serveur : /etc/rsyslog.d/00-serveur.conf
module(load="imtcp")
input(type="imtcp" port="514")
```

```
# Sur le client : /etc/rsyslog.d/90-vers-serveur.conf
*.info action(type="omfwd" target="logs.example.com" port="514" protocol="tcp")
```

⚠ Ouvrir le port côté serveur

Le port `514/tcp` doit être autorisé dans le pare-feu du serveur (`firewall-cmd --add-port=514/tcp`). Sans cela, aucun message n'arrive.

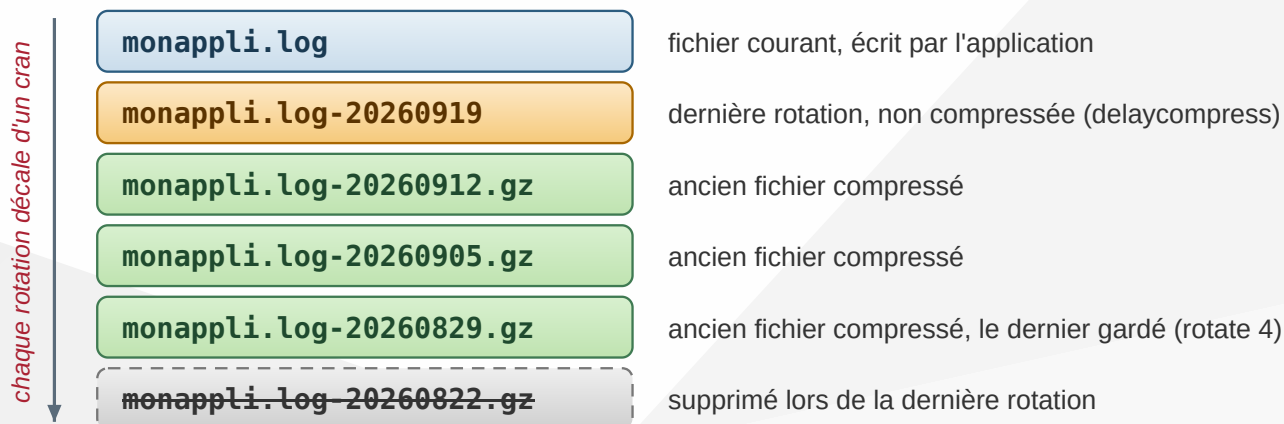
i Fiabiliser l'envoi

Une coupure de connexion peut faire perdre des messages avec TCP seul. Le protocole RELP (`omrelp` et `imrelp`, paquet `rsyslog-relp`) et une file d'attente sur disque (`queue.*`) fiabilisent l'envoi : un exemple figure en commentaire à la fin de `/etc/rsyslog.conf`, et la suite de cette annexe le détaille.

logrotate : faire tourner les logs

Les fichiers de log ne font que grossir. **logrotate** les fait *tourner* : il renomme le fichier courant, en recrée un vide, compresse les anciens et supprime les plus vieux.

Cycle de vie d'un fichier de log



i Lancement par un *timer*

Avant RHEL 9, **logrotate** était lancé par le script `/etc/cron.daily/logrotate`. C'est aujourd'hui un timer systemd, `logrotate.timer`, exécuté une fois par jour.

logrotate : où est la configuration ?

Fichier	Rôle
<code>/etc/logrotate.conf</code>	valeurs par défaut , puis <code>include</code> des règles de <code>/etc/logrotate.d</code>
<code>/etc/logrotate.d/*</code>	une règle par service, déposée par son paquet
<code>/var/lib/logrotate/logrotate.status</code>	état : date de la dernière rotation de chaque fichier

```
# Extrait de /etc/logrotate.conf
weekly
rotate 4
create
dateext
#compress
include /etc/logrotate.d
```

Le lancement quotidien se vérifie avec le timer :

```
$ systemctl list-timers logrotate.timer
```

logrotate : les directives à connaître

Directive	Effet
<code>daily</code> <code>weekly</code> <code>monthly</code>	fréquence de la rotation
<code>size 10M</code>	rotation dès que le fichier dépasse une taille
<code>rotate 4</code>	nombre d'anciens fichiers conservés
<code>compress</code> <code>delaycompress</code>	compresser les anciens (<i><code>delaycompress</code> : à la rotation suivante</i>)
<code>dateext</code>	suffixe daté (<code>-20260919</code>) plutôt que numéroté (<code>.1</code>)
<code>create 0640 root root</code>	recréer le fichier vide avec ce mode et ce propriétaire
<code>missingok</code> <code>notifempty</code>	ignorer un fichier absent, ne pas faire tourner un fichier vide
<code>copytruncate</code>	copier puis vider le fichier, si le programme ne sait pas le rouvrir
<code>postrotate</code> <code>endscript</code>	commande lancée après la rotation, écrite entre ces deux mots

logrotate : écrire sa règle

Cas d'un fichier `monappli.log` écrit par `rsyslog` :

```
# /etc/logrotate.d/monappli
/var/log/monappli.log {
    weekly
    rotate 4
    dateext
    compress
    delaycompress
    missingok
    notifempty
    create 0640 root root
    postrotate
        /usr/bin/systemctl reload rsyslog.service >/dev/null 2>&1 || true
    endscript
}
```

⚠ Un programme qui garde son fichier ouvert

Après la rotation, un programme qui n'a pas rouvert son fichier continue d'écrire dans l'ancien. Le bloc `postrotate` lui demande de le rouvrir.

logrotate : tester sa règle

```
# logrotate -d /etc/logrotate.d/monappli    # simulation
logrotate -f /etc/logrotate.d/monappli    # rotation forcée
ls -l /var/log/monappli.log*
```

```
≡ -rw-r-----. 1 root root    0 Sep 19 14:02 /var/log/monappli.log
  -rw-r--r--. 1 root root 1284 Sep 19 13:58 /var/log/monappli.log-20260919
```

💡 Simuler avant de forcer

`logrotate -d` affiche chaque décision (rotation, compression, suppression) sans toucher aux fichiers ni au fichier d'état.

`logrotate -f` force la rotation même si l'échéance n'est pas venue.

Pour aller plus loin

`man logrotate`, `man journald.conf` et la [documentation de rsyslog](#).

La suite de l'annexe traite la centralisation avancée : RELP, Graylog, `systemd-journal-remote` et les backends (Loki, ELK...).

Formats et transports de syslog

	RFC 3164 (BSD syslog, 2001)	RFC 5424 (syslog moderne, 2009)
Forme	<code><PRI>MMM jj hh:mm:ss</code> <code>HÔTE TAG: message</code>	<code><PRI>1 HORODATAGE HÔTE APP PROCID</code> <code>MSGID [DONNÉES-STRUCTURÉES] message</code>
Horodatage	sans année ni fuseau	ISO 8601, avec fuseau
Données structurées	non	oui, entre crochets

`PRI = facility × 8 + severity`, comme dans le module.

Transport	Remarque
UDP 514	historique : aucun accusé de réception, des messages peuvent se perdre
TCP 514	connexion établie, mais un message en cours d'envoi se perd si elle tombe
TLS 6514	TCP chiffré (RFC 5425)
RELp	TCP avec accusé de réception de chaque message, <code>rsyslog</code> seul

Centralisation fiable avec RELP

Paquet `rsyslog-relp` sur les deux machines. Le **récepteur** charge `imrelp` (fichier `/etc/rsyslog.d/00-relp.conf`, [documentation](#)) :

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

L'**émetteur** charge `omrelp` (fichier `/etc/rsyslog.d/99-relp-out.conf`, [documentation](#)), avec une file d'attente sur disque :

```
≡ module(load="omrelp")

if $syslogseverity <= 6 then {
  action(
    type="omrelp"
    target="10.205.49.2"
    port="2514"
    queue.type="LinkedList"
    queue.filename="relp_queue"
    queue.spoolDirectory="/var/spool/rsyslog"
    queue.maxDiskSpace="100m"
    queue.saveOnShutdown="on"
    action.resumeRetryCount="-1"
    action.resumeInterval="30"
  )
}
```

REL : SELinux et pare-feu

Sur AlmaLinux 10, `rsyslog` ne peut pas écouter sur `2514` : SELinux ne reconnaît pas ce port comme un port syslog. Au démarrage, le journal de `rsyslog` signale « error while binding relp tcp socket on port 2514 », et `ausearch -m avc` montre un `name_bind` refusé.

```
# dnf install policycoreutils-python-utils
semanage port -a -t syslogd_port_t -p tcp 2514
firewall-cmd --permanent --add-port=2514/tcp
firewall-cmd --reload
systemctl restart rsyslog
ss -tln | grep 2514
```

La file d'attente de l'émetteur

Avec `queue.*`, l'émetteur garde les messages sur disque tant que le récepteur ne répond pas (`action.resumeRetryCount="-1"` : réessais sans limite, toutes les 30 secondes).

Centraliser vers un serveur Graylog

Le gabarit `gelf` met chaque message au format attendu par Graylog, puis une action l'envoie en UDP (fichier `/etc/rsyslog.d/99-gelf-to-graylog.conf`) :

```
≡ template(name="gelf" type="list") {  
    constant(value="{\"version\": \"1.1\", \"")  
    constant(value=\"\"host\": \"")  
    property(name="hostname")  
    constant(value="\", \"short_message\": \"")  
    property(name="msg" format="json")  
    constant(value="\", \"timestamp\": \"")  
    property(name="timegenerated" dateformat="unixtimestamp")  
    constant(value="\", \"level\": \"")  
    property(name="syslogseverity")  
    constant(value="\"}")  
}  
  
if $syslogseverity <= 6 then {  
    action(type="omfwd" target="localhost" port="12201" protocol="udp" template="gelf")  
}
```

🎯 Cible

`localhost` est une valeur d'exemple : l'adresse du serveur Graylog et le port de son entrée GELF UDP (`12201` par défaut) la remplacent.

Centraliser avec journal-upload et journal-remote

Les deux côtés demandent le paquet `systemd-journal-remote`.

Sur le récepteur

`systemd-journal-remote.service` est lancé à la demande par `systemd-journal-remote.socket`. La configuration passe en partie par `/etc/systemd/journal-remote.conf`, mais l'écoute se règle sur la ligne de commande du service :

```
# systemctl edit systemd-journal-remote.service
```

```
[Service]
```

```
ExecStart=
```

```
ExecStart=/usr/lib/systemd/systemd-journal-remote --listen-http=-3 --output=/var/log/journal/remote/
```

La première ligne `ExecStart=` vide efface celle du paquet, reprise ensuite avec `--output` : l'écoute se fait en HTTP (`--listen-http`) et non en HTTPS (`--listen-https`, valeur du paquet). Le port `19532` est celui de la socket.

Sur l'émetteur

Éditer `/etc/systemd/journal-upload.conf` :

```
[Upload]  
URL=http://10.205.49.2:19532
```

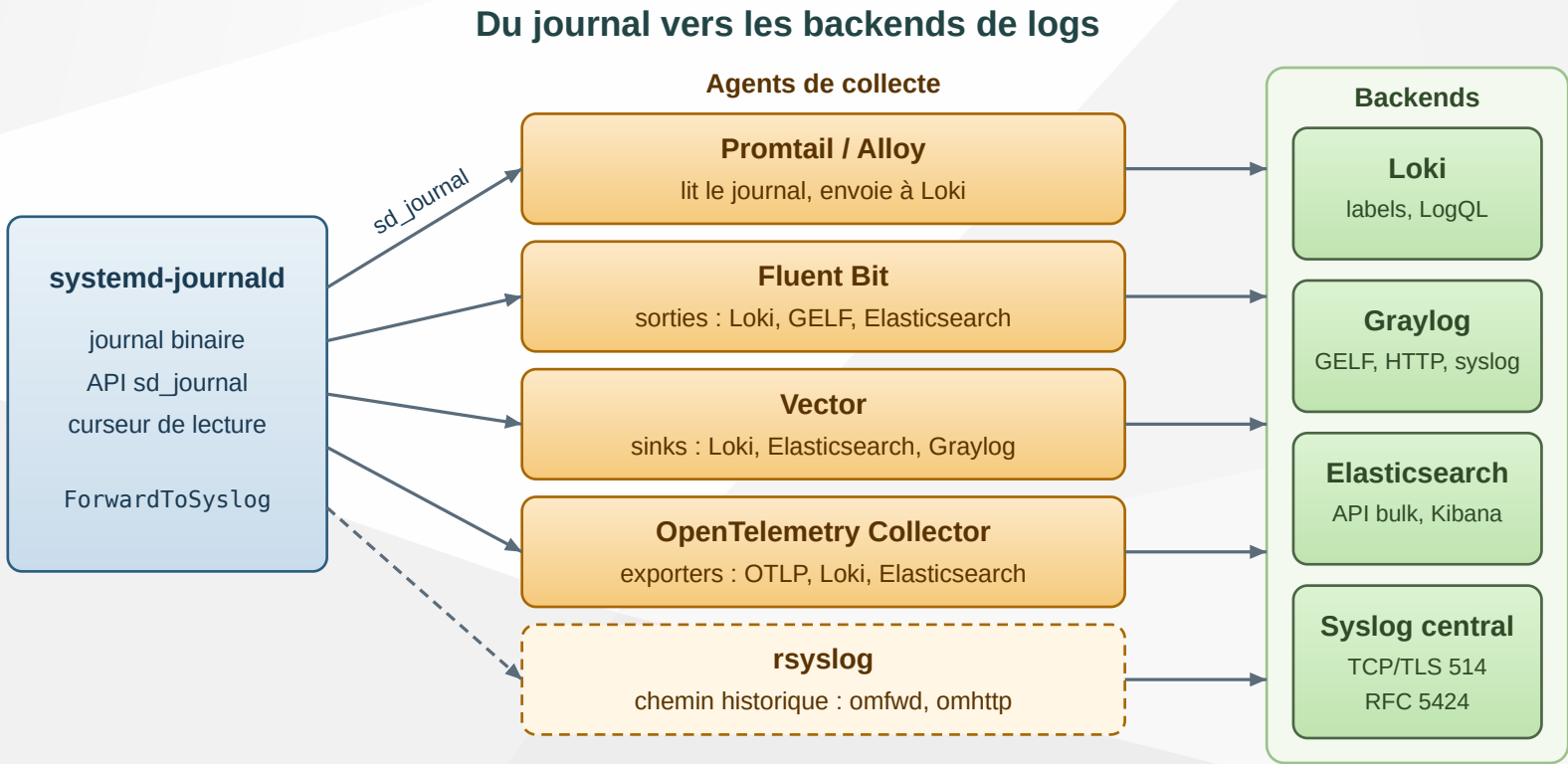
Puis démarrer le service :

```
# systemctl enable --now systemd-journal-upload.service
```

Pour visualiser les logs côté récepteur :

```
# journalctl --facility user --directory /var/log/journal/remote -f
```

Chemins vers des backends de log



Bonnes pratiques

- Une règle `rsyslog` validée (`rsyslogd -N1`) avant de redémarrer le service.
- `logrotate` configuré pour chaque application qui écrit dans `/var/log`, la règle testée avec `logrotate -d`.
- Un transport qui accuse réception (RELP, HTTP) plutôt que de l'UDP seul.
- SELinux et pare-feu ouverts avant de démarrer le récepteur.
- Transport chiffré (HTTPS) hors d'un réseau de confiance.
- Port du récepteur ouvert au seul réseau des émetteurs.
- Espace disque du récepteur surveillé : il reçoit les journaux de tout le parc.

Miroir de dépôts local

reposync, publication HTTP, synchronisation

De quoi parle-t-on

Un **miroir local** est une copie, sur un serveur interne, des dépôts d'une distribution : les serveurs du parc s'y servent au lieu de joindre Internet. Cette annexe montre comment le créer avec `dnf reposync`, le publier et le tenir à jour.

Au programme

- pourquoi un miroir, et la différence avec un dépôt « maison »,
- recopier un dépôt amont avec `reposync`,
- le servir en HTTP et le déclarer côté client,
- la synchronisation planifiée, et le cas de RHEL.

Pourquoi un miroir local ?

- **Bande passante** : un téléchargement pour tout le parc, au lieu d'un par serveur.
- **Reproductibilité** : tous les serveurs installent **la même version** d'un paquet, même des mois plus tard.
- **Maîtrise des correctifs** : on choisit **quand** une mise à jour amont devient disponible en interne (test, validation, déploiement).
- **Environnements fermés** : un serveur sans accès Internet (DMZ, zone industrielle) ne joint que le miroir.

i Le principe, en un schéma



Le miroir n'est mis à jour que si on le décide : c'est lui qui fixe le rythme des correctifs.

Miroir ou dépôt « maison »

	Dépôt « maison »	Miroir
Contenu	Ses propres paquets (build interne, RPM tiers)	Copie d'un dépôt amont (BaseOS, AppStream...)
Métadonnées	Générées localement par <code>createrepo_c</code>	Recopiées telles quelles, ou régénérées après filtrage
Outil	<code>createrepo_c</code>	<code>dnf reposync</code>

 Les deux se combinent

Dans un même dépôt local, un miroir des paquets officiels peut être complété par des composants internes. La structure d'un dépôt et la génération des métadonnées sont décrites à la slide « Anatomie d'un dépôt RPM ».

reposync : recopier un dépôt

`dnf reposync` (paquet `dnf-plugins-core`) télécharge **tous les paquets d'un dépôt actif**, avec ses métadonnées.

```
# dnf install dnf-plugins-core
dnf reposync --repoid=baseos --repoid=appstream \
  --download-metadata -p /var/www/html/mirror/almalinux/10/
```

Les identifiants de dépôt (`--repoid`) sont ceux de `dnf repolist`. Les mises à jour ultérieures ne retéléchargent que les nouveautés :

```
# dnf reposync --repoid=baseos --repoid=appstream \
  --download-metadata --delete -p /var/www/html/mirror/almalinux/10/
```

Option	Effet
<code>--download-metadata</code>	copie les métadonnées telles que signées en amont, sans les régénérer
<code>--delete</code>	supprime localement les paquets retirés du dépôt amont
<code>--newest-only</code>	ne télécharge que la version la plus récente de chaque paquet
<code>-p</code>	dossier de destination (un sous-dossier par <code>--repoid</code>)

Pièges de reposync

⚠ `--newest-only` avec `--download-metadata`

Les métadonnées recopiées décrivent aussi les anciennes versions, qui n'ont pas été téléchargées : `dnf` échoue dès qu'il en réclame une. Après la synchronisation, `createrepo_c --update` sur chaque dépôt corrige les métadonnées, mais elles ne sont alors plus celles signées en amont.

✎ Sans `--download-metadata`

Il faut générer les métadonnées avec `createrepo_c` après la synchronisation. C'est la méthode à retenir quand on **filtre** les paquets copiés.

i Place disque

Un dépôt complet est volumineux : un premier essai sur un seul dépôt, mesuré avec `du -sh`, permet de dimensionner le volume avant de tout recopier.

Servir le miroir en HTTP

Le client `dnf` ne fait que des requêtes `GET` : un simple serveur web suffit, avec le dossier recopié sous sa racine.

```
# dnf install httpd
systemctl enable --now httpd
firewall-cmd --permanent --add-service=http
firewall-cmd --reload
curl -I http://localhost/mirror/almalinux/10/baseos/repo/repodata/repomd.xml
```

La réponse `200 OK` sur `repomd.xml` confirme que le dépôt est lisible.

⚠ Contexte SELinux

Un dossier déplacé avec `mv` garde son ancien contexte, et `httpd` ne peut pas le lire. `restorecon -Rv /var/www/html/mirror` rétablit celui de la racine web. Un dossier créé par `reposync` sous `/var/www/html` hérite du contexte de son dossier parent.

❗ Pas de miroir sans signature

`gpgcheck=0` est réservé aux essais. En production, le client garde `gpgcheck=1` et la clé de la distribution.

Déclarer le miroir côté client

On remplace les serveurs publics par le miroir : dans le fichier `.repo` du dépôt, le `baseurl` pointe vers lui.

```
# /etc/yum.repos.d/almalinux-miroir.repo
[miroir-baseos]
name=Miroir interne BaseOS
baseurl=http://miroir.interne.lan/mirror/almalinux/10/baseos/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-AlmaLinux-10
```

Les dépôts publics d'origine sont désactivés (`enabled=0`) pour que le client n'en sorte pas.

```
# dnf repolist
dnf makecache
```

💡 Un dépôt par identifiant

Un second bloc `[miroir-appstream]` pointe vers `.../appstream/`, de la même façon.

Synchroniser avec un timer systemd

Le rythme des correctifs est celui du miroir : on le synchronise quand on est prêt à les déployer. Un service et un timer lancent la recopie.

```
# /etc/systemd/system/miroir-sync.service
[Unit]
Description=Synchronisation du miroir AlmaLinux 10

[Service]
Type=oneshot
ExecStart=/usr/bin/dnf reposync --repoid=baseos --repoid=appstream \
    --download-metadata --delete -p /var/www/html/mirror/almalinux/10/
```

```
# /etc/systemd/system/miroir-sync.timer
[Unit]
Description=Synchronisation hebdomadaire du miroir

[Timer]
OnCalendar=Sun 03:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
# systemctl daemon-reload
systemctl enable --now miroir-sync.timer
systemctl list-timers miroir-sync.timer
```


Cas de RHEL

	RHEL	AlmaLinux, Rocky Linux, CentOS Stream
Accès aux dépôts	abonnement (RHSM, <code>subscription-manager register</code>)	dépôts publics, sans abonnement
Miroir	<code>reposync</code> sur le serveur enregistré, avec ses certificats d'abonnement	<code>reposync</code> tel que vu dans cette annexe
Correctifs à grande échelle	Satellite : serveur de gestion de correctifs, miroir et suivi de conformité	miroir maison

 Satellite

Sur un parc RHEL important, Red Hat propose Satellite plutôt qu'un miroir à construire. Le principe reste celui de cette annexe : un point de distribution interne qui fixe le rythme des mises à jour.

Bonnes pratiques

- Un miroir sert au parc, pas à une machine : prévoir place disque et bande passante.
- `gpgcheck=1` côté client, clé de la distribution distribuée avec le parc.
- Synchronisation planifiée (timer), journal relu (`journalctl -u miroir-sync`).
- Option `--newest-only` suivie de `createrepo_c --update`, ou métadonnées régénérées.
- SELinux : `restorecon -Rv` après un déplacement de dossier.
- Un dépôt de test synchronisé avant le dépôt de production.

Montée de version majeure

de la version 9 à la version 10, sur place

De quoi parle-t-on

Une **montée de version majeure** fait passer un serveur de la version 9 à la version 10 sans le réinstaller : données, configuration et services restent en place, le système est remplacé. Elle se distingue des mises à jour courantes (`dnf upgrade`), qui restent dans la même version majeure.

Au programme

- montée sur place ou réinstallation : comment choisir,
- les chemins pris en charge selon la distribution,
- la préparation, puis `leapp preupgrade` et son rapport,
- l'exécution, les vérifications, ELevate pour AlmaLinux et CentOS Stream.

Deux stratégies

	Montée sur place	Réinstallation et restauration
Principe	le système est remplacé sur la machine, avec l'outil <code>leapp</code>	une machine neuve est installée, la configuration et les données y sont rejouées
Atouts	données et réglages conservés, pas de machine supplémentaire	partir d'un système propre, retour arrière simple (l'ancienne machine reste intacte)
Limites	environnement complexe, risque d'échec en cours de route	installation (voir slide « Principe : rejouer automatiquement une installation ») et configuration à rejouer
Convient pour	serveur peu personnalisé, ou qu'on sait difficile à reconstruire	parc homogène déjà décrit par un Kickstart ou une automatisation

 Le choix se fait sur la reconstructibilité

Un serveur dont la configuration est entièrement décrite (Kickstart, automatisation) se réinstalle plus sûrement qu'il ne se monte en version. La montée sur place sert surtout aux serveurs qu'on ne sait plus reconstruire.

Les chemins pris en charge

Distribution	Montée de la version 9 à la version 10
RHEL	<code>leapp</code> , outil officiel Red Hat, d'une version majeure à la suivante seulement (de 9 à 10, pas de 8 à 10)
AlmaLinux, CentOS Stream	ELevate , projet de la communauté AlmaLinux construit sur <code>leapp</code>
Rocky Linux	pas de montée directe documentée au moment de la rédaction : la réinstallation est la voie recommandée (à revérifier dans la documentation de la distribution)

 Version source et version cible

Sur RHEL, la version mineure de départ détermine la version cible. Selon la documentation Red Hat : de 9.6 vers 10.0, et de 9.8 vers 10.2. La documentation de chaque version précise les couples pris en charge.

 Cas non pris en charge

D'après la documentation Red Hat : stockage multipath en réseau (SAN en Fibre Channel, FCoE, Infiniband), Stratis, passage de BIOS à UEFI pendant la montée, architectures hors x86-64-v3 pour x86.

Préparer la montée

1. Sauvegarder de façon à pouvoir revenir en arrière : image de la machine avec ReaR (voir slide « ReaR : sauvegarder une machine entière »), snapshot de VM ou de volume LVM, copie de la configuration.

2. Mettre le système à jour dans sa version actuelle, puis redémarrer :

```
# dnf update
reboot
```

3. Sur RHEL, fixer la version et les dépôts de départ (abonnement valide) :

```
# subscription-manager repos \
    --enable rhel-9-for-x86_64-baseos-rpms \
    --enable rhel-9-for-x86_64-appstream-rpms
subscription-manager release --set 9.6
```

4. Retirer ce qui gêne : verrous de versions (`dnf versionlock clear`), systèmes de gestion de configuration qui ne doivent pas intervenir pendant la montée (Puppet, Salt, Chef), systèmes de fichiers inutiles (démontés et mis en commentaire dans `/etc/fstab`).

Installer leapp

```
# dnf install leapp-upgrade
```

`leapp` est un cadre de mise à niveau : le paquet `leapp-upgrade` apporte les règles propres à RHEL 9 vers RHEL 10. Il sert deux fois : à **analyser** (sans rien modifier), puis à **exécuter**.

Tester sur une copie

Chaque environnement est différent (applications, pilotes, configuration). Le projet ELevate recommande de rejouer la montée sur une VM ou un clone avant la production, et ce conseil vaut pour tous les chemins.

Pré-analyse : leapp preupgrade

```
# leapp preupgrade --target 10.0
```

La commande analyse la machine **sans la mettre à niveau** : elle produit un rapport dans `/var/log/leapp/leapp-report.txt`. On la relance après chaque correction, jusqu'à obtenir un rapport sans blocage.

🔗 Le paramètre `--target`

Il désigne la version cible (voir slide « Les chemins pris en charge »). La valeur `10.0` convient à une machine en 9.6.

Lire le rapport :

Niveau	Sens
Inhibitor	bloque la montée : à corriger avant d'aller plus loin
High, Medium, Low	risque d'importance décroissante (paquet retiré, service modifié), à traiter ou à accepter en connaissance de cause
Info	à lire, sans action requise

Chaque entrée du rapport propose une remédiation (commande ou changement de configuration).

Exécuter : `leapp upgrade`

```
# leapp upgrade --target 10.0  
reboot
```

Au redémarrage, la machine démarre dans un **environnement de mise à niveau** (un `initramfs` dédié) qui remplace les paquets, puis redémarre sur le système de la version cible. Pendant cette phase, SELinux est positionné en mode permissif.

⚠ Une machine indisponible

Le serveur n'est pas utilisable pendant la montée. La durée dépend du nombre de paquets et des performances du disque : une fenêtre de maintenance est à prévoir, avec un accès console (VM ou écran distant) pour suivre l'opération.

Vérifier après la montée

```
$ cat /etc/os-release  
getenforce  
rpm -qa | grep el9
```

- `/etc/os-release` confirme la version.
- `getenforce` donne le mode de SELinux (permissif pendant la montée) : à contrôler, et à rétablir si besoin.
- `rpm -qa | grep el9` repère les paquets restés en version 9 : à remplacer ou à retirer à la main.

Les journaux de la montée restent dans `/var/log/leapp/` (`leapp-upgrade.log` et `leapp-report.txt`). On vérifie ensuite les services (`systemctl --failed`), le réseau, le pare-feu et les applications.

ELevate : AlmaLinux et CentOS Stream

Le projet **ELevate** ajoute à `leapp` les règles et les données des distributions dérivées. Le principe est celui de RHEL : préparer, analyser, exécuter.

```
# dnf update
reboot
dnf install -y \
    http://repo.almalinux.org/elevate/elevate-release-latest-el$(rpm --eval %rhel).noarch.rpm
dnf install -y leapp-upgrade leapp-data-almalinux
leapp preupgrade
leapp upgrade
reboot
```

Le paquet de données (`leapp-data-almalinux`) est celui de la distribution de départ : les autres distributions ont leur équivalent, indiqué dans le guide du projet. Le rapport est au même endroit que sur RHEL.

Sources

[Guide de démarrage ELevate](#) et [Upgrading from RHEL 9 to RHEL 10](#) : les commandes et les paquets évoluent avec les versions, la documentation du jour fait foi.

Bonnes pratiques

- Sauvegarde ou snapshot testé avant de commencer.
- Montée répétée sur un clone avant la production.
- `leapp preupgrade` relancé jusqu'à un rapport sans inhibiteur.
- Fenêtre de maintenance et accès console prévus.
- Vérification de la version, de SELinux et des paquets restés en version 9.
- Réinstallation envisagée pour un serveur dont la configuration est décrite.

Accents, locales et jeux de caractères

diagnostiquer et corriger les caractères mal affichés

De quoi parle-t-on

Un fichier ne contient que des **octets** : c'est une convention (le jeu de caractères) qui dit quel octet est quelle lettre. Quand l'émetteur, le fichier et le récepteur ne s'accordent pas, les accents se transforment en `Ã©` ou en `?`. Cette annexe donne le vocabulaire et les outils pour trouver où la convention diverge.

Au programme

- le vocabulaire : jeu de caractères, encodage, locale,
- les variables `LANG` et `LC_*`, et leur réglage sur le système,
- identifier et convertir l'encodage d'un fichier ou d'un nom de fichier,
- systèmes de fichiers étrangers, terminal et SSH,
- le diagnostic par symptôme.

Jeu de caractères, encodage, locale

Terme	Sens	Exemples
Jeu de caractères (<i>charset</i>)	l'ensemble des caractères connus	ASCII, Latin-1, Unicode
Encodage	la façon de les écrire en octets	UTF-8, ISO-8859-1 (Latin-1), ISO-8859-15 (Latin-9, avec le signe €)
Locale	les conventions d'une langue et d'un pays : encodage, tri, dates, nombres	fr_FR.UTF-8
Page de codes (<i>codepage</i>)	l'ancien nom d'un encodage sous DOS et Windows	437, 850, 1252

i UTF-8 est l'encodage courant

Il couvre tous les caractères d'Unicode. Un caractère accentué y occupe **deux octets**, contre un seul en Latin-1 : c'est la source de la plupart des erreurs d'affichage.

Locale : les variables

Une locale se choisit par des variables d'environnement, une par catégorie de comportement.

Variable	Catégorie
LC_CTYPE	classification et encodage des caractères
LC_COLLATE	ordre de tri
LC_TIME, LC_NUMERIC, LC_MONETARY	dates, nombres, monnaie
LC_MESSAGES	langue des messages
LANG	valeur par défaut de toutes les catégories
LC_ALL	force toutes les catégories, quelles que soient les autres variables

Ordre de priorité : LC_ALL, puis la variable LC_* de la catégorie, puis LANG.

\$

locale
locale -a

locale affiche les valeurs en cours, locale -a les locales **installées** sur la machine.

Effet sur le tri : un exemple

Le même tri, sous deux locales :

```
$ printf 'b\ na\né\nz\n' | LC_ALL=C sort | tr '\n' ' '
printf 'b\ na\né\nz\n' | LC_ALL=fr_FR.UTF-8 sort | tr '\n' ' '

```

```
≡ a b z é
  a b é z

```

Sous la locale `C`, le tri suit la valeur des octets : `é` arrive après `z`. Sous `fr_FR.UTF-8`, il suit l'alphabet.

💡 La locale `C` pour les scripts

Un script qui compare ou trie des textes fixe `LC_ALL=C` pour un résultat identique sur toutes les machines. Un script qui affiche des messages à l'utilisateur laisse la locale de la session.

Régler la locale du système

Le réglage de la machine est dans `/etc/locale.conf`, lu au démarrage. L'outil est `localectl`.

```
# dnf install glibc-langpack-fr
localectl set-locale LANG=fr_FR.UTF-8
localectl
```

- `glibc-langpack-fr` apporte la locale française. Le paquet `glibc-all-langpacks` les apporte toutes.
- `localectl` sans argument affiche la locale et la disposition du clavier.
- Un utilisateur peut surcharger la locale dans son `~/.bashrc` (`export LANG=fr_FR.UTF-8`).

La disposition du clavier de la console se règle de même (`localectl set-keymap fr`).

⚠ Locale absente

Une locale demandée mais non installée produit un avertissement à chaque commande : `warning: setlocale: LC_ALL: cannot change locale (fr_FR.UTF-8)`. Le message disparaît en installant le paquet de langue.

Identifier l'encodage d'un fichier

```
$ file -i u8.txt l15.txt
```

```
u8.txt: text/plain; charset=utf-8  
l15.txt: text/plain; charset=iso-8859-1
```

`file` **devine** à partir du contenu. Il ne distingue pas `ISO-8859-1` de `ISO-8859-15` : les deux ne diffèrent que par quelques octets (dont `€`). Pour voir les octets réels :

```
$ od -c l15.txt
```

```
0000000 311   l 350   v   e   340       N   o 353   l  \n
```

En Latin-1, chaque accent est un seul octet (`311` pour `É`, `350` pour `è`, en octal). En UTF-8, il en faut deux.

Convertir un fichier : `iconv`

```
$ iconv -f UTF-8 -t ISO-8859-15 u8.txt > l15.txt
iconv -f ISO-8859-15 -t UTF-8 l15.txt
iconv -l
```

- `-f` : encodage d'origine, `-t` : encodage cible, `-l` : liste des encodages connus.
- Le résultat sort sur la sortie standard : on le redirige vers un **autre** fichier. Rediriger vers le fichier d'origine le viderait, le shell l'ouvrant en écriture avant la lecture.

```
$ iconv -f UTF-8 -t ISO-8859-1 <<< "5 €"
```

```
≡ iconv: illegal input sequence at position 2
```

Latin-1 n'a pas le signe `€` : la conversion s'arrête. `-t ISO-8859-15` convient, `-t ASCII//TRANSLIT` remplace par un équivalent (`Eleve a Noel`), `-c` supprime les caractères sans équivalent.

Convertir les noms de fichiers : `convmv`

`iconv` convertit le **contenu**. Les **noms** de fichiers sont stockés tels quels, sans indication d'encodage : on les convertit avec `convmv`.

```
$ convmv -f iso-8859-1 -t utf8 -r ~/archive  
convmv -f iso-8859-1 -t utf8 -r --notest ~/archive
```

- Sans `--notest`, `convmv` **simule** : il affiche les renommages prévus.
- Avec `--notest`, il renomme.
- `-r` parcourt les sous-dossiers.

Où trouver `convmv`

C'est un paquet à part (`dnf info convmv` indique son dépôt, souvent EPEL sur les dérivées de RHEL). Il s'emploie typiquement après une copie depuis une machine ancienne (archive Latin-1, partage Windows).

Archives et noms

Une archive `tar` ou `zip` créée sous un autre encodage garde ses noms tels quels : les accents s'affichent mal à l'extraction. On corrige avec `convmv` sur le dossier extrait.

Systèmes de fichiers étrangers

Les noms de fichiers d'un système FAT (clé USB) ou d'un partage Windows sont convertis au **montage**, selon des options.

```
# mount -t vfat -o utf8,codepage=850 /dev/sdb1 /mnt/usb
```

```
# /etc/fstab
/dev/sdb1 /mnt/usb vfat utf8,codepage=850,noauto 0 0
```

Option	Rôle
utf8	convertit les noms vers UTF-8 à la lecture et à l'écriture
codepage=	page de codes des noms courts (437 par défaut)
iocharset=	encodage utilisé pour convertir les noms, à la place de utf8

 utf8 plutôt que iocharset=utf8

Sur FAT, la documentation du noyau recommande l'option utf8 : iocharset=utf8 rendrait le système de fichiers sensible à la casse. Les options propres à chaque système de fichiers sont décrites dans mount(8).

Terminal et SSH

Les accents passent par trois maillons : le **programme** (locale du processus), le **terminal** (son encodage) et, à distance, **SSH**.

- Le terminal doit utiliser le même encodage que la locale du processus : UTF-8 des deux côtés dans le cas courant.
- SSH peut transmettre `LANG` et `LC_*` du client vers le serveur : option `SendEnv` côté client, `AcceptEnv` côté serveur (voir `ssh_config(5)` et `sshd_config(5)`).
- Une machine distante sans la locale du client affiche l'avertissement vu plus haut : on installe le paquet de langue, ou on règle `LANG` sur une locale présente.

💡 Tester sans changer la machine

`LANG=C.UTF-8 commande` ou `LC_ALL=fr_FR.UTF-8 commande` isole le réglage à une seule commande, ce qui permet de comparer les affichages.

Diagnostic par symptôme

Symptôme	Cause probable	Action
NoÃ«l, LÃ©o	texte UTF-8 lu comme Latin-1	régler l'encodage du terminal ou de l'éditeur sur UTF-8
Noël, Léo	texte Latin-1 lu comme UTF-8	convertir avec <code>iconv -f ISO-8859-1 -t UTF-8</code>
? à la place des accents	caractère sans équivalent dans l'encodage cible, ou locale <code>C</code>	locale UTF-8, ou encodage cible capable de le représenter
<code>setlocale: cannot change locale</code>	locale demandée non installée	<code>dnf install glibc-langpack-fr</code>
tri qui place é après z	<code>LC_COLLATE</code> en <code>C</code> (valeur d'octets)	locale <code>fr_FR.UTF-8</code> pour la session
noms de fichiers illisibles	noms encodés autrement que la locale	<code>convmv</code> , ou options de montage

Le texte `NoÃ«l` s'obtient en lisant du UTF-8 comme du Latin-1 :

```
$ iconv -f ISO-8859-1 -t UTF-8 <<< "Noël"

≡ NoÃ«l
```

Bonnes pratiques

- UTF-8 partout : locale du système, terminal, éditeur, fichiers.
- Locale vérifiée avec `locale` et `localectl`, paquet de langue installé.
- Encodage d'un fichier contrôlé avec `file -i` puis `od -c` en cas de doute.
- Conversion avec `iconv` vers un **nouveau** fichier.
- `convmv` en simulation avant `--notest`.
- `LC_ALL=C` dans les scripts qui comparent ou trient.
- Options de montage `utf8` et `codepage` pour les clés USB et partages anciens.

Intégrité du système

rpm -V, AIDE et auditd

De quoi parle-t-on

L'**intégrité** d'un système répond à une question : *les fichiers sont-ils ceux qu'on y a mis ?* Elle sert à détecter une modification non voulue (erreur, panne de disque, intrusion) et à retrouver ce qui a changé. Cette annexe présente trois outils de la famille RedHat, qui répondent chacun à une question différente.

Au programme

- `rpm -V` : les fichiers des paquets sont-ils intacts ?
- AIDE : quels fichiers ont changé depuis un état de référence ?
- `auditd` : qui a modifié un fichier, et quand ?

Trois outils, trois questions

Outil	Question	Couvre	À installer
<code>rpm -v</code>	ce fichier est-il conforme à son paquet ?	les fichiers livrés par des paquets	non (système de base)
AIDE	qu'est-ce qui a changé depuis la référence ?	tous les fichiers choisis, paquets ou non	<code>aide</code>
<code>auditd</code>	qui l'a modifié, avec quelle commande ?	les fichiers surveillés par une règle	installé de base

💡 Ils se complètent

AIDE signale **qu'un** fichier a changé, `rpm -v` dit s'il s'écarte de son paquet, `auditd` dit **qui** l'a modifié.

rpm -V : vérifier un paquet

`rpm -V` compare les fichiers installés aux informations du paquet (taille, droits, propriétaire, somme de contrôle, date). Sans écart, il n'affiche rien.

```
# rpm -V openssh-server  
rpm -Vf /usr/bin/true  
rpm -Va  
rpm -Va --noconfig
```

- `-V paquet` vérifie un paquet, `-Vf fichier` le paquet qui contient ce fichier.
- `-Va` vérifie **tous** les paquets installés : c'est plus long, tout le système est lu.
- `--noconfig` écarte les fichiers de configuration, que l'administrateur modifie légitimement.

⚠ À exécuter avec les droits de `root`

Un utilisateur ordinaire ne peut pas lire tous les fichiers : `rpm -V` affiche alors `?` à la place de la somme de contrôle et `missing ... (Permission denied)`. Le résultat est inexploitable sans `sudo`.

Lire la sortie de `rpm -V`

```
≡ S.5....T.  c /etc/ssh/sshd_config
```

Neuf positions, un point (.) pour un attribut conforme, puis le type du fichier et son chemin.

Lettre	Écart constaté
S	taille (<i>size</i>)
M	droits ou type (<i>mode</i>)
5	somme de contrôle du contenu
D	numéro de périphérique
L	lien symbolique
U, G	propriétaire, groupe
T	date de modification (<i>time</i>)
P	capacités (<i>capabilities</i>)
?	attribut illisible (droits insuffisants)

Le type : `c` configuration, `d` documentation, `g` fichier fantôme (*ghost*, créé à l'exécution), `l` licence, `r` fichier « lisez-moi ». Un fichier absent s'affiche `missing`.

Exemple : une ligne ajoutée, des droits modifiés

```
$ echo "# test" | sudo tee -a /etc/ssh/sshd_config  
sudo rpm -V openssh-server
```

```
≡ S.5....T.  c /etc/ssh/sshd_config
```

Taille, somme de contrôle et date ont changé. Si on modifie aussi les droits (`chmod 644`), la ligne devient `SM5....T.`. Rétablir le contenu d'origine (`cp -a` d'une sauvegarde, puis `restorecon -v`) ramène la sortie à vide.

i Un système sain n'est pas toujours muet

Sur une machine installée normalement, `rpm -Va` affiche une poignée de lignes (fichiers de configuration modifiés par l'installation, droits d'un `/root`). Elles servent de **référence** : seules les lignes nouvelles sont à examiner.

Limites de rpm -V et signatures

rpm -V s'appuie sur la **base RPM de la machine** (`/var/lib/rpm`). Quelqu'un qui contrôle la machine peut modifier un fichier **et** la base qui le décrit : le contrôle ne voit alors rien.

Deux contrôles à partir d'un paquet de confiance :

```
$ dnf download bash
sudo rpm -Vp bash-*.rpm
rpm -K bash-*.rpm
```

```
≡ bash-5.2.26-6.el10.x86_64.rpm: digests signatures OK
```

- `rpm -Vp paquet.rpm` compare les fichiers installés au **paquet téléchargé**, pas à la base locale.
- `rpm -K` vérifie la signature du paquet avec les clés des distributions (`rpm -q gpg-pubkey`).

✎ Ce que `rpm -V` ne couvre pas

Les fichiers qui ne viennent d'aucun paquet (scripts maison, `/home`, données) : c'est le rôle d'AIDE.

AIDE : le principe

AIDE (*Advanced Intrusion Detection Environment*) enregistre dans une **base de référence** l'état des fichiers choisis (empreintes SHA, droits, propriétaires, dates, contexte SELinux). Un contrôle compare le disque à cette base et rapporte les écarts.

```
# dnf install aide
```

La configuration est `/etc/aide.conf` :

```
≡ NORMAL = R+sha512-m-c  
CONTENT = ftype+sha512  
PERMS = ftype+p+u+g+acl+selinux+xattrs  
  
/boot    NORMAL  
/usr     NORMAL  
/etc/passwd$    NORMAL  
/opt     CONTENT  
!/usr/src
```

Des **règles** (`NORMAL`...) nomment un jeu d'attributs à comparer. Chaque ligne associe un chemin à une règle, et `!` exclut un chemin.

i Que surveille la configuration par défaut ?

`/boot`, `/bin`, `/sbin`, `/lib`, `/lib64`, `/usr`, `/root` et une liste de fichiers de `/etc` : le système, pas `/home` ni les données.

Initialiser, contrôler, mettre à jour

```
# aide --init  
mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz  
aide --check
```

- `aide --init` construit la base dans `aide.db.new.gz`. Sur une machine de test, 58 592 entrées en une quinzaine de secondes.
- La base de référence doit s'appeler `aide.db.gz` : on **renomme** le fichier produit.
- `aide --check` compare le disque à la base et affiche un rapport. Il se termine par `AIDE found NO differences between database and filesystem` quand tout est conforme.

Après un **changement voulu**, la base est reconstruite :

```
# aide --update  
mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

⚠ Initialiser sur un système de confiance

La base décrit l'état **au moment de l'initialisation** : si le système est déjà compromis, la référence l'est aussi. On l'initialise juste après l'installation et la configuration.

Lire le rapport d'AIDE

```
Summary:
  Total number of entries:      58592
  Added entries:                1
  Removed entries:              0
  Changed entries:              4

Added entries:
f+++++: /etc/tp-nouveau.conf

Changed entries:
f = p...  ...A... : /etc/hosts
f > ...   ..H.... : /etc/ssh/sshd_config
```

Marque	Sens
f+++	fichier ajouté
f---	fichier supprimé
p	droits modifiés
A	ACL modifiées
H	empreinte modifiée

La suite du rapport détaille chaque écart, avec l'ancienne et la nouvelle valeur.

Code de retour d'aide --check

Le code de retour est un **masque de bits**, utilisable par un script ou une supervision.

Code	Sens
0	aucun écart
1	fichiers ajoutés
2	fichiers supprimés
4	fichiers modifiés
5	ajout et modification (1 + 4), comme dans l'exemple précédent

```
$ aide --check ; echo $?
```

Planifier le contrôle

Un contrôle utile est un contrôle **régulier**. Le paquet livre la rotation de `/var/log/aide/aide.log` mais pas de planification : un timer systemd (ou une tâche cron) lance `aide --check` chaque nuit. L'unité passe en échec quand AIDE trouve des écarts (code de retour non nul), ce que `systemctl --failed` ou la supervision signale.

```
# /etc/systemd/system/aide-check.service
[Unit]
Description=Contrôle d'intégrité AIDE

[Service]
Type=oneshot
ExecStart=/usr/sbin/aide --check
```

Le timer associé suit le modèle de la slide « Synchroniser avec un timer systemd » de l'annexe sur le miroir de dépôts (`OnCalendar=daily`).

Protéger la base

! La base est une cible

Un intrus qui peut écrire la base peut effacer ses traces. On en garde une **copie hors de la machine** (support en lecture seule, serveur distant) et on compare cette copie à l'originale avant un contrôle de confiance.

🔧 Du bruit normal

Certains fichiers changent seuls (`/etc/resolv.conf` géré par NetworkManager, `/etc/hostname`). On les exclut dans `/etc/aide.conf` (`!/etc/resolv.conf`) pour que seuls les vrais écarts restent visibles.

auditd : surveiller un fichier

Le service `auditd` enregistre dans le journal d'audit des **événements du noyau**, dont l'accès à un fichier. Une règle de surveillance (`-w`) l'active :

```
# auditctl -w /etc/ssh/sshd_config -p wa -k sshd_conf
auditctl -l
```

Option	Rôle
<code>-w chemin</code>	fichier ou dossier surveillé
<code>-p wa</code>	accès surveillés : <code>w</code> écriture, <code>a</code> changement d'attributs (<code>r</code> lecture, <code>x</code> exécution)
<code>-k clé</code>	étiquette de recherche

Une modification du fichier produit alors des événements, lus par `ausearch`. L'option `-i` traduit les numéros en noms :

```
# ausearch -k sshd_conf -i
aureport -k
```

```
≡ # date time key success exe auid event
1. 10/06/26 23:51:37 sshd_conf yes /usr/sbin/auditctl 1000 1445
2. 10/06/26 23:51:37 sshd_conf yes /usr/bin/tee 1000 1456
```

La première ligne est la création de la règle, la seconde l'écriture dans le fichier.

Qui est l'auteur : `audit`

Dans l'événement `SYSCALL` d'une modification faite avec `sudo` :

```
≡ audit=alma uid=root ... exe=/usr/bin/tee
```

- `uid` : l'utilisateur effectif (`root`, à cause de `sudo`).
- `audit` : l'utilisateur de **connexion** d'origine (*audit uid*), conservé à travers `sudo` et `su`.
- `exe` : la commande qui a écrit.

C'est l'`audit` qui répond à « qui ? » : un `uid=root` seul n'identifie personne.

i Règle sur la durée

`auditctl -w` agit jusqu'au prochain redémarrage. Pour la rendre permanente, la règle s'écrit dans `/etc/audit/rules.d/` puis `augenrules --load` la charge :

```
# echo '-w /etc/ssh/sshd_config -p wa -k sshd_conf' > /etc/audit/rules.d/50-sshd.rules
augenrules --load
```

🔧 Règles « à l'ancienne »

`auditctl -w` affiche `Old style watch rules are slower`. Il reste adapté à quelques fichiers ; au-delà, des règles sur les appels système (`-a always,exit -F path=...`) sont plus rapides.

Quel outil pour quelle situation

Situation	Premier geste
un service se comporte bizarrement après une mise à jour	<code>rpm -V paquet</code>
un fichier de configuration est suspect	<code>rpm -Vf fichier</code> , puis <code>aide --check</code>
un fichier hors paquet a changé	<code>aide --check</code>
il faut savoir qui a modifié un fichier	<code>ausearch -k clé -i</code>
contrôle régulier d'un parc	AIDE planifié, base conservée hors machine

Une enquête enchaîne souvent les trois : AIDE signale le fichier, `rpm -V` dit si son paquet l'explique, `ausearch` attribue la modification.

 **Aucun outil ne remplace la sauvegarde**

Ces outils **détectent**. Ils ne réparent rien : le retour à l'état connu passe par la réinstallation du paquet ou la restauration (voir slide « ReaR : restaurer ou cloner »).

Bonnes pratiques

- `rpm -V` avec `sudo`, sur un paquet précis, `--noconfig` pour un résultat lisible.
- `rpm -Vp` ou `rpm -K` à partir d'un paquet téléchargé en cas de doute sur la base locale.
- AIDE initialisé sur un système sain, juste après l'installation.
- Base d'AIDE copiée hors de la machine, contrôle planifié.
- Faux positifs exclus dans `/etc/aide.conf`, pas ignorés.
- Règles `auditd` posées sur les fichiers sensibles (`/etc/ssh/sshd_config`, `/etc/passwd`, `/etc/sudoers`), rendues permanentes dans `rules.d`.
- Alerte lue avec `audit` pour identifier l'auteur.