Travaux Diriges

Administration Linux Avancee

Fil rouge : Infrastructure web haute disponibilite — MediaStream

8 TD progressifs — de l'installation a la supervision

Le fil rouge : MediaStream

Contexte general

Vous etes administrateur systeme chez **MediaStream**, une startup de diffusion de contenus video en ligne comptant 80 000 utilisateurs actifs par jour.

L'equipe technique a decide de migrer son infrastructure vers des serveurs Linux autogeres, en abandonnant un hebergement cloud couteux.

Votre mission : **concevoir, deployer, tuner et superviser** une infrastructure web haute disponibilite avec **Nginx** comme serveur web et **HAProxy** comme load balancer.

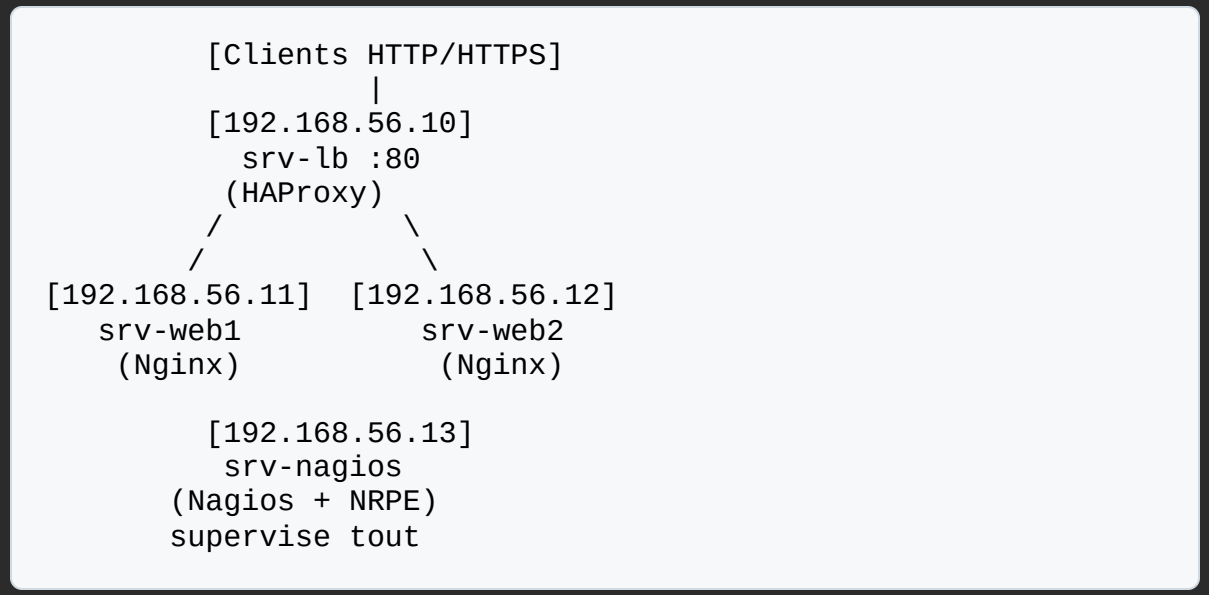
> Chaque TD correspond a une phase du projet. Le travail de chaque TD s'appuie sur le precedent.

Infrastructure cible

Machine	Role	IP	OS
`srv-lb`	Load balancer HAProxy	192.168.56.10/24	Rocky Linux 9
`srv-web1`	Serveur web Nginx (backend 1)	192.168.56.11/24	Rocky Linux 9
`srv-web2`	Serveur web Nginx (backend 2)	192.168.56.12/24	Rocky Linux 9
`srv-nagios`	Supervision Nagios	192.168.56.13/24	Rocky Linux 9

> Environnement : 4 VMs (KVM/VirtualBox/VMware). Chaque VM : 2 vCPU, 2 Go RAM, 2 disques de 20 Go minimum.

Architecture finale



> HAProxy distribue les requetes HTTP entre les deux backends Nginx en round-robin. Si l'un tombe, le trafic bascule automatiquement sur l'autre. Nagios surveille chaque composant en temps reel.

Prerequis techniques

- Hyperviseur installe (KVM + virt-manager, VirtualBox ou VMware)
- ISO Rocky Linux 9 / AlmaLinux 9 telechargee
- Acces reseau entre les VMs (reseau host-only ou bridge)
- Connaissance de base de Linux (navigation, edition, services)

Configuration reseau des VMs

Ajoutez dans `/etc/hosts` de **chaque machine** :

```
192.168.56.10  srv-lb      srv-lb.mediastream.lan
192.168.56.11  srv-web1    srv-web1.mediastream.lan
192.168.56.12  srv-web2    srv-web2.mediastream.lan
192.168.56.13  srv-nagios  srv-nagios.mediastream.lan
```

TD 1 — Installation et deploiement

TD 1 — Objectif

Contexte MediaStream : Le materiel est arrive. Vous devez installer les quatre serveurs de maniere **reproductible** avec un stockage **resilient**. Une reinstallation manuelle n'est pas acceptable pour une startup qui evolue vite.

A la fin de ce TD, vous aurez :

- Installe `srv-lb` manuellement une premiere fois
- Cree un fichier Kickstart a partir de cette installation
- Configure un stockage RAID1 + LVM adapte a chaque role
- Securise le chargeur de demarrage GRUB
- Deploye les trois autres serveurs via Kickstart

TD 1 — Etape 1 : Installation initiale de srv-lb

1. Creez une VM `srv-lb` avec **2 disques virtuels** de 20 Go chacun.
2. Installez Rocky Linux 9 en mode **minimal** (sans interface graphique).
3. Pendant l'installation, configurez :
   - Nom d'hote : `srv-lb.mediastream.lan`
   - Utilisateur admin : `msadmin` (membre du groupe `wheel`)
   - Reseau : IP statique `192.168.56.10/24`, passerelle `192.168.56.1`
   - Partitionnement **manuel** : reservez pour RAID+LVM (cf. etape suivante)

> L'objectif est d'abord de comprendre ce que fait l'installeur, puis de l'automatiser. Ne passez pas trop de temps ici : l'essentiel vient apres.

TD 1 — Etape 2 : Recuperer le Kickstart genere

4. Connectez-vous a `srv-lb` et recuperez le fichier Kickstart genere :

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

Ce fichier contient toutes les reponses donnees pendant l'installation :

- `%packages` : liste des paquets
- `network` : configuration reseau
- `part` / `raid` / `logvol` : schema de partitionnement

5. Copiez-le sur votre poste hote :

```
scp root@192.168.56.10:/root/anaconda-ks.cfg ./ks-original.cfg
```

TD 1 — Etape 3 : Creer le Kickstart RAID1 + LVM

6. Creez `ks-mediastream.cfg`. Commencez par les parametres globaux :

```
# Langue et fuseau
lang fr_FR.UTF-8
keyboard fr-latin1
timezone Europe/Paris --utc

# Reseau srv-lb
network --bootproto=static --ip=192.168.56.10 --netmask=255.255.255.0 \
        --gateway=192.168.56.1 --nameserver=8.8.8.8 \
        --hostname=srv-lb.mediastream.lan --activate

# Mode texte, pas de reboot auto pendant les tests
text
reboot
```

TD 1 — Etape 3 : Partitionnement RAID1 + LVM

Ajoutez le schema de partitionnement. Les deux disques (`sda` , `sdb`) sont miroires :

```
clearpart --all --initlabel

# Partition /boot hors du RAID (le bootloader ne gere pas le RAID)
part /boot --fstype=ext4 --size=1024 --ondisk=sda

# Membres RAID pour le LVM
part raid.01 --size=1 --grow --ondisk=sda
part raid.02 --size=1 --grow --ondisk=sdb

# RAID1 -> PV LVM
raid pv.01 --level=1 --device=md0 raid.01 raid.02
volgroup vg_lb pv.01

# Volumes logiques
logvol /     --vgname=vg_lb --name=lv_root --fstype=xfs  --size=8192
logvol /var  --vgname=vg_lb --name=lv_var  --fstype=xfs  --size=6144
logvol /home --vgname=vg_lb --name=lv_home --fstype=xfs  --size=2048
logvol swap  --vgname=vg_lb --name=lv_swap             --size=2048
```

> Notez l'utilisation de **XFS** : c'est le systeme de fichiers par defaut de Rocky Linux 9. Il excelle sur les workloads de fichiers nombreux (logs, contenu statique).

TD 1 — Etape 3 : Kickstart (suite)

Completez avec les paquets et le post-install :

```
# Mot de passe root (generer avec : python3 -c "import crypt; print(crypt.crypt('monpass'))")
rootpw --iscrypted <hash>
user --name=msadmin --groups=wheel --password=<hash> --iscrypted

# Paquets
%packages
@core
vim-enhanced
rsync
curl
%end

# Post-install : remplir /etc/hosts
%post
echo "192.168.56.11  srv-web1  srv-web1.mediastream.lan"  >> /etc/hosts
echo "192.168.56.12  srv-web2  srv-web2.mediastream.lan"  >> /etc/hosts
echo "192.168.56.13  srv-nagios srv-nagios.mediastream.lan" >> /etc/hosts
%end
```

TD 1 — Etape 4 : Valider et tester le Kickstart

7. Validez la syntaxe :

```
dnf install -y pykickstart
ksvalidator ks-mediastream.cfg
```

Resultat attendu : Aucune sortie = syntaxe valide.

8. Servez le fichier en HTTP depuis votre poste hote :

```
python3 -m http.server 8080
```

9. Creez `srv-lb-v2` avec 2 disques, bootez sur l'ISO et ajoutez au menu GRUB :

```
inst.ks=http://192.168.56.1:8080/ks-mediastream.cfg
```

L'installation se deroule sans intervention. Verifiez le resultat :

```
cat /proc/mdstat      # RAID1 actif ?
lvs                   # LV crees ?
```

TD 1 — Etape 5 : Verifier le RAID et le LVM

10. Verifiez l'etat du RAID :

```
cat /proc/mdstat
```

Resultat attendu :

```
md0 : active raid1 sdb1[1] sda2[0]
      19921920 blocks super 1.2 [2/2] [UU]
```

> `[UU]` : les deux disques sont actifs. `[U_]` signalerait un disque defaillant. Verifiez egalement avec `mdadm --detail /dev/md0`.

11. Verifiez les volumes logiques :

```
lvs
df -hT
```

Resultat attendu : `/`, `/var`, `/home` montes en XFS sur les bons LV.

TD 1 — Etape 6 : Securiser GRUB

12. Protegez le bootloader par mot de passe :

```
grub2-setpassword
```

Cette commande cree `/boot/grub2/user.cfg` avec le hash PBKDF2 du mot de passe.

13. Redemarrez et tentez d'editer une entree GRUB avec `e` : le mot de passe est demande.

> Le demarrage **normal** ne demande pas de mot de passe. Seule l'**edition des parametres de boot** est protegee. Cela empeche d'ajouter `init=/bin/bash` pour obtenir un shell root sans authentification.

TD 1 — Etape 7 : Deployer les trois autres serveurs

14. Generez les Kickstart pour chaque serveur en adaptant l'IP, le hostname et le nom du VG :

```
# srv-web1
sed -e 's/192.168.56.10/192.168.56.11/g' \
    -e 's/srv-lb/srv-web1/g' \
    -e 's/vg_lb/vg_web1/g' \
    ks-mediastream.cfg > ks-web1.cfg

# srv-web2
sed -e 's/192.168.56.10/192.168.56.12/g' \
    -e 's/srv-lb/srv-web2/g' \
    -e 's/vg_lb/vg_web2/g' \
    ks-mediastream.cfg > ks-web2.cfg

# srv-nagios
sed -e 's/192.168.56.10/192.168.56.13/g' \
    -e 's/srv-lb/srv-nagios/g' \
    -e 's/vg_lb/vg_nagios/g' \
    ks-mediastream.cfg > ks-nagios.cfg
```

15. Installez les trois VMs et verifiez la connectivite croisee (`ping`).

TD 1 — Checkpoint

- [ ] 4 serveurs installes et accessibles par SSH
- [ ] RAID1 actif sur chaque machine (`[UU]` dans `/proc/mdstat`)
- [ ] LVM configure : VG + 4 LV par machine
- [ ] GRUB protege par mot de passe sur tous les serveurs
- [ ] Ping croise entre toutes les machines : 0% packet loss

TD 2 — Configuration logicielle

TD 2 — Objectif

Contexte MediaStream : Les serveurs sont installes. Avant de deployer Nginx et HAProxy, vous devez maitriser la gestion des paquets, comprendre les dependances, et construire un depot local pour controler les versions en production.

A la fin de ce TD, vous aurez :

- Inspecte et disseque les paquets RPM de Nginx et HAProxy
- Analyse les dependances binaires
- Cree un depot RPM local sur `srv-web2`
- Installe Nginx et HAProxy depuis ce depot controle

TD 2 — Etape 1 : Anatomie du paquet Nginx

1. Sur `srv-web1`, telechargez le RPM sans l'installer :

```
dnf download nginx
```

2. Inspectez les metadonnees :

```
rpm -qpi nginx-*.rpm
```

Resultat attendu : Nom, version, release, architecture, description, URL upstream.

3. Listez les fichiers inclus dans le paquet :

```
rpm -qpl nginx-*.rpm
```

Resultat attendu : La liste complete des fichiers installes, dont `/usr/sbin/nginx`, `/etc/nginx/nginx.conf`, `/usr/lib/systemd/system/nginx.service`.

TD 2 — Etape 1 : Anatomie (suite)

4. Inspectez les scripts pre/post-installation :

```
rpm -qp --scripts nginx-*.rpm
```

Resultat attendu : Un script `%post` qui execute `systemctl preset nginx` pour activer ou non le service selon les presets du systeme. C'est ce script qui decide si Nginx demarre automatiquement apres installation.

5. Extraire le RPM sans installer (pour inspection) :

```
mkdir /tmp/nginx-extract
cd /tmp/nginx-extract
rpm2cpio ~/nginx-*.rpm | cpio -idmv
ls usr/sbin/
file usr/sbin/nginx
```

> `rpm2cpio` convertit l'archive RPM en archive `cpio`. C'est utile pour recuperer un fichier de configuration par defaut ecrase, sans reinstaller le paquet entier.

TD 2 — Etape 2 : Dependances et bibliotheques

6. Analysez les dependances declarees dans le RPM :

```
rpm -qpR nginx-*.rpm | head -20
```

Resultat attendu : Liste des dependances (`openssl` , `pcre` , `zlib` , `gd` , etc.).

7. Installez Nginx et examinez les bibliotheques liees dynamiquement :

```
dnf install -y nginx
ldd /usr/sbin/nginx
```

Resultat attendu : Une dizaine de lignes de type `libssl.so.3 => /lib64/libssl.so.3` .

8. Identifiez quel paquet RPM fournit une bibliotheque :

```
rpm -qf /lib64/libssl.so.3
```

Resultat attendu : `openssl-libs-3.x.x-x.el9.x86_64` — le paquet exact qui detient ce fichier.

TD 2 — Etape 3 : Le linker dynamique

9. Inspectez le cache du linker :

```
ldconfig -p | grep libpcre
```

Resultat attendu : Le chemin vers `libpcre2-8.so.0` (bibliotheque de regex utilisee par Nginx).

10. Comprenez pourquoi cette bibliotheque est critique :

```
# Simuler une bibliotheque manquante (ne pas faire en prod !)
LD_PRELOAD=/dev/null /usr/sbin/nginx -t
```

Resultat attendu : `nginx: [alert] could not open error log file` ou une erreur de bibliotheque. La directive `-t` teste la configuration sans demarrer.

> Le linker dynamique (`ld-linux.so`) resout les dependances a l'execution. Si une `.so` est absente ou incompatible, le binaire ne peut pas demarrer — meme si le RPM etait installe correctement. C'est pourquoi `ldd` et `rpm -qf` sont des outils de depannage essentiels.

TD 2 — Etape 4 : Creer le depot local

11. Sur `srv-web2`, creez le depot RPM local :

```
dnf install -y createrepo_c
mkdir -p /var/repo/mediastream/{base,nginx,haproxy}
```

12. Telechargez les paquets et leurs dependances :

```
# Nginx et ses dependances
dnf download --resolve --destdir=/var/repo/mediastream/nginx nginx

# HAProxy et ses dependances
dnf install -y epel-release
dnf download --resolve --destdir=/var/repo/mediastream/haproxy haproxy
```

13. Generez les metadonnees du depot :

```
createrepo_c /var/repo/mediastream/nginx
createrepo_c /var/repo/mediastream/haproxy
```

TD 2 — Etape 5 : Exposer le depot en HTTP

14. Installez httpd et exposez le depot :

```
dnf install -y httpd
cat > /etc/httpd/conf.d/repo.conf << 'EOF'
Alias /repo /var/repo/mediastream
<Directory /var/repo/mediastream>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
EOF

systemctl enable --now httpd
firewall-cmd --permanent --add-service=http && firewall-cmd --reload
```

15. Testez depuis `srv-lb` :

```
curl -s http://srv-web2.mediastream.lan/repo/ | grep -o 'nginx\|haproxy'
```

TD 2 — Etape 6 : Configurer les clients

16. Sur `srv-lb`, `srv-web1` et `srv-nagios`, ajoutez le fichier de depot :

```
cat > /etc/yum.repos.d/mediastream-local.repo << 'EOF'
[mediastream-nginx]
name=MediaStream Local - Nginx
baseurl=http://srv-web2.mediastream.lan/repo/nginx
enabled=1
gpgcheck=0

[mediastream-haproxy]
name=MediaStream Local - HAProxy
baseurl=http://srv-web2.mediastream.lan/repo/haproxy
enabled=1
gpgcheck=0
EOF
```

17. Verifiez que le depot est reconnu :

```
dnf repolist
dnf info nginx
```

Resultat attendu : `mediastream-nginx` et `mediastream-haproxy` apparaissent. `dnf info nginx` affiche `Depot : mediastream-nginx`.

TD 2 — Etape 7 : Installer depuis le depot

18. Sur `srv-web1` et `srv-web2`, installez Nginx **exclusivement depuis le depot local** :

```
dnf install --disablerepo="*" --enablerepo="mediastream-nginx" nginx
```

19. Sur `srv-lb`, installez HAProxy depuis le depot local :

```
dnf install --disablerepo="*" --enablerepo="mediastream-haproxy" haproxy
```

20. Verifiez les versions installees :

```
nginx -v
haproxy -v
```

> Travailler avec un depot local garantit que **tous les serveurs ont exactement la meme version** des logiciels. En production, cela evite les surprises dues a des mises a jour non maitrisees.

TD 2 — Checkpoint

- [ ] Nginx installe sur srv-web1 et srv-web2 (depuis le depot local)
- [ ] HAProxy installe sur srv-lb (depuis le depot local)
- [ ] `dnf repolist` affiche les deux repos mediastream sur les clients
- [ ] `ldd /usr/sbin/nginx` : toutes les dependances resolues (`not found` absent)
- [ ] `rpm -qf /lib64/libpcre2-8.so.0` retourne le nom du paquet source

TD 3 — Systemes de fichiers et stockage

TD 3 — Objectif

Contexte MediaStream : Les serveurs web vont stocker du contenu statique (images, videos, JS/CSS). Les logs d'acces Nginx peuvent croitre tres rapidement. Vous devez organiser le stockage intelligemment et etre capable de revenir en arriere apres un deploiement problematique.

A la fin de ce TD, vous aurez :

- Cree des volumes logiques dedies au contenu web et aux logs
- Compare ext4 et XFS sur un workload de logs web
- Etendu un LV a chaud pour faire face a la croissance
- Utilise les snapshots LVM comme filet de securite avant un deploiement

TD 3 — Etape 1 : Creer les LV dedies

1. Sur `srv-web1`, verifiez l'espace disponible :

```
vgs
```

2. Creez deux LV pour le contenu web et les logs :

```
lvcreate -L 3G -n lv_www  vg_web1
lvcreate -L 2G -n lv_logs vg_web1
```

3. Formater et monter :

```
mkfs.xfs /dev/vg_web1/lv_www
mkfs.xfs /dev/vg_web1/lv_logs

mkdir -p /var/www/html /var/log/nginx

mount /dev/vg_web1/lv_www  /var/www/html
mount /dev/vg_web1/lv_logs /var/log/nginx
```

TD 3 — Etape 1 : Rendre le montage permanent

4. Ajoutez les entrees dans `/etc/fstab` :

```
echo "/dev/vg_web1/lv_www   /var/www/html   xfs  defaults,noatime  0 0" >> /etc/fstab
echo "/dev/vg_web1/lv_logs  /var/log/nginx  xfs  defaults,noatime  0 0" >> /etc/fstab
```

5. Testez la validite de `/etc/fstab` sans redemarrer :

```
mount -a
df -hT | grep vg_web1
```

Resultat attendu :

```
/dev/mapper/vg_web1-lv_www    xfs     3.0G   33M  3.0G   2% /var/www/html
/dev/mapper/vg_web1-lv_logs   xfs     2.0G   28M  2.0G   2% /var/log/nginx
```

> L'option `noatime` supprime la mise a jour de l'horodatage d'acces lors de chaque lecture. Pour un serveur web servant des milliers de fichiers statiques, c'est un gain de performance mesurable (cf. TD 7).

TD 3 — Etape 2 : Comparer ext4 et XFS pour les logs web

6. Creez deux LV de test :

```
lvcreate -L 300M -n lv_test_ext4 vg_web1
lvcreate -L 300M -n lv_test_xfs  vg_web1
mkfs.ext4 /dev/vg_web1/lv_test_ext4
mkfs.xfs  /dev/vg_web1/lv_test_xfs
mkdir -p /mnt/test_{ext4,xfs}
mount /dev/vg_web1/lv_test_ext4 /mnt/test_ext4
mount /dev/vg_web1/lv_test_xfs  /mnt/test_xfs
```

7. Simulez des logs web (milliers de petits fichiers) :

```
time for i in $(seq 1 2000); do
    echo "192.168.1.$((RANDOM%256)) - - [$(date)] \"GET /video_$i HTTP/1.1\" 200 $((RANDOM%5000))" \
    > /mnt/test_ext4/access_$i.log
done

time for i in $(seq 1 2000); do
    echo "192.168.1.$((RANDOM%256)) - - [$(date)] \"GET /video_$i HTTP/1.1\" 200 $((RANDOM%5000))" \
    > /mnt/test_xfs/access_$i.log
done
```

TD 3 — Etape 2 : Resultats de comparaison

8. Comparez l'utilisation des inodes et de l'espace :

```
df -i /mnt/test_ext4 /mnt/test_xfs
tune2fs -l /dev/vg_web1/lv_test_ext4 | grep -E "Inode count|Block size|Reserved"
```

Resultats attendus :

Critere	ext4	XFS
Inodes totaux	Fixe a la creation	Alloues dynamiquement
Reserve systeme	5% de l'espace	0%
Meilleur pour	Partitions fixes, disques < 1 To	Gros fichiers, logs, > 1 To

9. Nettoyez :

```
umount /mnt/test_{ext4,xfs}
lvremove -f /dev/vg_web1/lv_test_{ext4,xfs}
```

TD 3 — Etape 3 : Snapshot avant deploiement

MediaStream s'apprete a deployer une nouvelle version du site. Vous allez creer un **snapshot** comme point de retour en arriere.

10. Creez une page web initiale (v1) :

```
cat > /var/www/html/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head><title>MediaStream</title></head>
<body>
  <h1>MediaStream</h1>
  <p>Version 1.0 — Production</p>
</body>
</html>
EOF
```

11. Creez le snapshot **avant** le deploiement :

```
lvcreate -L 500M -s -n snap_www_v1 /dev/vg_web1/lv_www
```

> Un snapshot LVM ne copie pas les donnees immediatement. Il enregistre uniquement les **blocs modifies** apres la creation du snapshot (copy-on-write). L'espace alloue (`500M`) doit etre suffisant pour absorber toutes les modifications jusqu'a la suppression du snapshot.

TD 3 — Etape 4 : Simuler un deploiement problematique

12. Deployez la "nouvelle version" (qui introduit un probleme) :

```
cat > /var/www/html/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head><title>MediaStream</title></head>
<body>
  <h1>MediaStream</h1>
  <p>Version 2.0 — BROKEN — erreur de deploiement</p>
  <!-- PROBLEME: fichiers manquants -->
</body>
</html>
EOF

# Simuler la suppression accidentelle du contenu
rm -f /var/www/html/style.css
```

13. Verifiez l'etat du snapshot :

```
lvs -a | grep snap
```

Resultat attendu : Le snapshot utilise maintenant quelques Ko (les blocs modifies).

TD 3 — Etape 5 : Restauration depuis le snapshot

14. Pour restaurer, fusionnez le snapshot avec le volume original :

```
# Demontez d'abord le volume (arret temporaire du service)
systemctl stop nginx
umount /var/www/html

# Fusionner le snapshot (restaure l'etat v1)
lvconvert --merge /dev/vg_web1/snap_www_v1
```

15. Remontez et verifiez :

```
mount /dev/vg_web1/lv_www /var/www/html
cat /var/www/html/index.html
systemctl start nginx
```

Resultat attendu : Le fichier affiche `Version 1.0 — Production`. Le deploiement problematique est annule en moins de 30 secondes.

TD 3 — Etape 6 : Extension a chaud

16. Simulez une saturation des logs (`/var/log/nginx` a 90%) :

```
dd if=/dev/zero of=/var/log/nginx/fill.log bs=1M count=1800
df -h /var/log/nginx
```

17. Etendez le LV **sans demonter** :

```
lvextend -L +1G /dev/vg_web1/lv_logs
xfs_growfs /var/log/nginx
df -h /var/log/nginx
```

Resultat attendu : La partition passe de 2 Go a 3 Go sans interruption de service.

> Avec XFS, `xfs_growfs` etend le systeme de fichiers a chaud. Avec ext4, on utilise `resize2fs`. Dans les deux cas, le service Nginx **ne doit pas etre coupe** pour cette operation.

18. Nettoyez :

```
rm /var/log/nginx/fill.log
```

TD 3 — Checkpoint

- [ ] LV `lv_www` (3 Go, XFS) monte sur `/var/www/html` avec `noatime`
- [ ] LV `lv_logs` (2 Go, XFS) monte sur `/var/log/nginx`
- [ ] Montages persistants dans `/etc/fstab` (valides avec `mount -a`)
- [ ] Snapshot cree et restauration validee (retour a v1 en < 60s)
- [ ] Extension a chaud de `lv_logs` sans interruption de service

TD 4 — Noyau et peripheriques

TD 4 — Objectif

Contexte MediaStream : Avec 80 000 utilisateurs actifs par jour, le serveur web va recevoir des milliers de connexions simultanees. Les parametres noyau par defaut sont calibres pour un usage general — pas pour un serveur HTTP haute charge. Vous devez adapter le noyau a ce workload.

A la fin de ce TD, vous aurez :

- Realise un inventaire materiel complet
- Charge et configure des modules noyau pour la production
- Applique un profil sysctl optimise pour un serveur HTTP
- Ecrit une regle udev pour peripheriques reseau

TD 4 — Etape 1 : Inventaire materiel

1. Sur `srv-lb`, faites un inventaire complet :

```
# CPU
lscpu | grep -E "^CPU\(s\)|Model name|Architecture"

# Memoire
free -h

# Disques
lsblk -o NAME,SIZE,TYPE,MOUNTPOINT

# Interfaces reseau
ip link show
```

2. Inventoriez le materiel PCI (carte reseau, controleur disque) :

```
lspci | grep -iE "ethernet|network|storage|RAID"
```

Resultat attendu : La carte reseau virtuelle apparait (ex : `VirtIO network device` ou `Intel 82540EM`).

TD 4 — Etape 1 : Inventaire reseau approfondi

3. Inspectez la carte reseau avec `ethtool` :

```
# Identifiez l'interface principale
ip route | grep default | awk '{print $5}'

# Puis remplacez <iface> par le nom trouve
ethtool <iface>
```

Resultat attendu :

```
Settings for eth0:
    Speed: 1000Mb/s
    Duplex: Full
    Link detected: yes
```

4. Listez les parametres reseau modifiables par le driver :

```
ethtool -k <iface> | grep -E "tcp-segmentation|scatter-gather|generic-receive"
```

> Ces fonctions de **decharge materielle** (hardware offloading) permettent au driver de traiter des segments TCP directement en hardware, reduisant la charge CPU. En virtualisation, elles sont souvent emulees par le driver virtio.

TD 4 — Etape 2 : Gestion des modules noyau

5. Identifiez le module du driver reseau :

```
ethtool -i <iface> | grep driver
```

6. Inspectez ce module :

```
# Remplacez <module> par le nom trouve (ex: virtio_net, e1000)
modinfo <module>
```

Resultat attendu : Description, auteur, parametres disponibles (`parm:`).

7. Listez les modules charges en filtrant les modules reseau :

```
lsmod | grep -E "^Module|net|eth|virt"
```

8. Simulez le dechargement et rechargement d'un module non critique :

```
modprobe -r dummy    # cree une interface reseau factice
modprobe dummy
ip link show dummy0  # l'interface apparait
```

TD 4 — Etape 3 : Explorer /sys et udev

9. Explorez les attributs de la carte reseau dans `/sys` :

```
ls /sys/class/net/
cat /sys/class/net/<iface>/speed
cat /sys/class/net/<iface>/mtu
cat /sys/class/net/<iface>/address
```

10. Surveillez les evenements udev en temps reel :

```
# Dans un terminal
udevadm monitor --environment --udev &

# Dans un autre, modifiez un parametre (ex: MTU)
ip link set <iface> mtu 1400
ip link set <iface> mtu 1500
```

Resultat attendu : Les evenements `UDEV [...] change /devices/...` s'affichent a chaque modification.

TD 4 — Etape 4 : Ecrire une regle udev

11. Creez une regle udev pour nommer une interface reseau de maniere predictible :

```
# Recupérez les attributs de la carte
udevadm info --query=all --path=/sys/class/net/<iface> | grep -E "ID_NET|ATTR"
```

12. Creez la regle :

```
cat > /etc/udev/rules.d/70-mediastream-net.rules << 'EOF'
# Renommer l'interface reseau principale en "ms-front"
# Adapter ATTR{address} a l'adresse MAC de votre carte
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="<votre_MAC>", \
    NAME="ms-front"
EOF
```

13. Testez sans redemarrer :

```
udevadm test /sys/class/net/<iface>
```

Resultat attendu : La regle est trouvee et le `NAME="ms-front"` est applique dans la simulation.

> Les noms d'interface predictibles (`eth0` , `ms-front`) facilitent les scripts d'administration. Sans udev, le nom peut changer apres un reboot si le noyau detecte les cartes dans un ordre different.

TD 4 — Etape 5 : Profil sysctl pour serveur HTTP haute charge

14. Examinez les valeurs par defaut critiques pour un serveur HTTP :

```
sysctl net.core.somaxconn
sysctl net.ipv4.tcp_fin_timeout
sysctl net.ipv4.ip_local_port_range
sysctl net.ipv4.tcp_tw_reuse
```

Valeurs par defaut typiques :

- `somaxconn = 4096` (file d'attente connexions en attente)
- `tcp_fin_timeout = 60` (secondes avant liberation d'une socket en FIN_WAIT)
- `ip_local_port_range = 32768 60999` (ports ephemeres disponibles)

TD 4 — Etape 5 : Appliquer le profil HTTP

15. Creez un fichier sysctl dedie a MediaStream :

```
cat > /etc/sysctl.d/90-mediastream-http.conf << 'EOF'
# File d'attente de connexions TCP (HAProxy accept)
net.core.somaxconn = 65535
net.core.netdev_max_backlog = 65535

# Delai de liberation des sockets en TIME_WAIT
net.ipv4.tcp_fin_timeout = 15
net.ipv4.tcp_tw_reuse = 1

# Plage de ports ephemeres elargie (pour connexions HAProxy -> backends)
net.ipv4.ip_local_port_range = 1024 65535

# Tampons socket TCP
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

# Nombre maximal de connexions ouvertes
fs.file-max = 200000
EOF
```

TD 4 — Etape 5 : Application et verification

16. Appliquez et verifiez :

```
sysctl -p /etc/sysctl.d/90-mediastream-http.conf
sysctl net.core.somaxconn
sysctl net.ipv4.tcp_fin_timeout
```

Resultats attendus :

```
net.core.somaxconn = 65535
net.ipv4.tcp_fin_timeout = 15
```

17. Verifiez la persistance apres reboot :

```
reboot
# Apres reconnexion :
sysctl net.core.somaxconn
```

Resultat attendu : `65535` (valeur persistee).

> Sur un serveur sous HAProxy, `somaxconn` est critique : c'est la taille de la file d'attente des connexions TCP acceptees par le noyau avant que HAProxy ne les traite. Une valeur trop faible cause des `SYN_RECV` abandonnes sous charge.

TD 4 — Checkpoint

- [ ] Inventaire complet : CPU, RAM, disques, carte reseau (`lspci` , `ethtool`)
- [ ] Module reseau identifie avec `ethtool -i` et inspecte avec `modinfo`
- [ ] Regle udev creee et testee avec `udevadm test`
- [ ] Profil sysctl `/etc/sysctl.d/90-mediastream-http.conf` applique
- [ ] `sysctl net.core.somaxconn` retourne `65535` apres reboot

TD 5 — Maintenance et metrologie

TD 5 — Objectif

Contexte MediaStream : L'infrastructure entre en production. Vous devez mettre en place les outils de surveillance operationnelle : journalisation centralisee des acces web, detection de modification non autorisee des configurations, et suivi des metriques systeme sous charge.

A la fin de ce TD, vous aurez :

- Centralise les logs d'acces Nginx sur un serveur dedie
- Configure AIDE pour detecter toute modification de `/etc/nginx` et `/etc/haproxy`
- Utilise les outils de metrologie pour analyser la charge generee par le serveur web
- Mis en place une rotation automatique des logs Nginx

TD 5 — Etape 1 : Centraliser les logs Nginx (serveur)

1. Sur `srv-web2` (serveur de logs centraux), configurez rsyslog pour recevoir les logs distants :

```
# Editez /etc/rsyslog.conf et decommentez ces 4 lignes :
module(load="imudp")
input(type="imudp" port="514")
module(load="imtcp")
input(type="imtcp" port="514")
```

2. Ajoutez une regle de stockage par machine source :

```
cat > /etc/rsyslog.d/mediastream-remote.conf << 'EOF'
template(name="WebLogs" type="string"
    string="/var/log/remote/%HOSTNAME%/%PROGRAMNAME%.log")
if $fromhost-ip != '127.0.0.1' then {
    action(type="omfile" dynaFile="WebLogs")
    stop
}
EOF
```

TD 5 — Etape 1 : Demarrage et ouverture firewall

3. Demarrez rsyslog et ouvrez le port :

```
systemctl restart rsyslog
firewall-cmd --permanent --add-port=514/tcp --add-port=514/udp
firewall-cmd --reload
```

4. Creez le repertoire d'accueil des logs et verifiez les permissions :

```
mkdir -p /var/log/remote
chmod 755 /var/log/remote
```

TD 5 — Etape 2 : Configurer Nginx pour envoyer ses logs

5. Sur `srv-web1`, configurez Nginx pour ecrire ses logs dans syslog :

```
# Editez /etc/nginx/nginx.conf
# Dans le bloc http {}, remplacez les directives access_log et error_log :
```

```
http {
    # Envoyer access_log vers syslog (mediastream-access comme tag)
    access_log syslog:server=srv-web2.mediastream.lan:514,
               facility=local7,tag=nginx-access,severity=info
               combined;

    # Conserver aussi un log local pour le debug
    access_log /var/log/nginx/access.log combined;
    error_log  /var/log/nginx/error.log warn;
}
```

6. Testez et rechargez :

```
nginx -t && systemctl reload nginx
logger -t nginx-access "Test de centralisation des logs"
```

TD 5 — Etape 2 : Verification de la centralisation

7. Sur `srv-web2`, verifiez la reception des logs :

```
ls /var/log/remote/
tail -f /var/log/remote/srv-web1/nginx-access.log
```

8. Generez du trafic depuis `srv-lb` pour alimenter les logs :

```
for i in $(seq 1 20); do
    curl -s http://192.168.56.11/ > /dev/null
done
```

Resultat attendu : Les acces apparaissent en temps reel dans `/var/log/remote/srv-web1/nginx-access.log` sur `srv-web2`.

> `@@` en rsyslog signifie TCP (transport fiable avec acquittement). Utilisez TCP pour les logs de production — vous ne voulez pas perdre des entrees en cas de pic reseau.

TD 5 — Etape 3 : AIDE — Surveillance d'integrite

9. Sur `srv-lb`, installez AIDE :

```
dnf install -y aide
```

10. Editez `/etc/aide.conf` pour limiter la surveillance aux fichiers critiques :

```
# Ajoutez ces regles en debut de fichier (apres les macros de groupes)
cat >> /etc/aide.conf << 'EOF'
# Surveiller la configuration HAProxy
/etc/haproxy    NORMAL
/etc/haproxy/haproxy.cfg    CONTENT+ATTR

# Surveiller les certificats si presents
/etc/pki/tls    NORMAL
EOF
```

11. Initialisez la base de reference :

```
aide --init
mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

> La commande `--init` peut prendre 1 a 3 minutes. Elle calcule les hachages SHA-256 de tous les fichiers surveilles et les stocke dans la base de reference.

TD 5 — Etape 4 : AIDE — Simuler une modification

12. Modifiez la configuration HAProxy (simulation d'une modification non autorisee) :

```
echo "# Modification non autorisee" >> /etc/haproxy/haproxy.cfg
```

13. Lancez un audit AIDE :

```
aide --check 2>&1 | head -40
```

Resultat attendu :

```
AIDE found differences between database and filesystem!!

Changed files:
changed: /etc/haproxy/haproxy.cfg

Detailed information about changes:
                 Old             New
SHA256           : abc123...     def456...
Mtime            : ...           ...
```

14. Annulez la modification et reinitialise la base :

```
# Supprimer la ligne ajoutee
sed -i '/Modification non autorisee/d' /etc/haproxy/haproxy.cfg
aide --init && mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

TD 5 — Etape 5 : Automatiser AIDE

15. Planifiez l'audit AIDE quotidien avec un rapport par mail :

```
cat > /etc/cron.d/aide-check << 'EOF'
# Audit AIDE tous les jours a 3h du matin
0 3 * * * root /usr/sbin/aide --check 2>&1 | \
    mail -s "[AIDE] Audit $(hostname) $(date +%Y-%m-%d)" root
EOF
```

16. Testez l'execution manuelle du cron :

```
run-parts --test /etc/cron.d/
```

> En l'absence de serveur mail local, redirigez la sortie vers un fichier de log : `>> /var/log/aide-report.log 2>&1`. Le principe reste le meme : une trace horodatee de chaque audit.

TD 5 — Etape 6 : Metrologie sous charge HTTP

17. Sur `srv-web1`, installez les outils de metrologie :

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

18. Demarrez une collecte en arriere-plan et generez de la charge :

```
# Terminal 1 : collecte toutes les 2 secondes
vmstat 2 30 > /tmp/vmstat-charge.txt &

# Terminal 2 : generez du trafic HTTP
for i in $(seq 1 500); do
    curl -s http://localhost/ > /dev/null &
done
wait
```

19. Analysez la collecte :

```
cat /tmp/vmstat-charge.txt
```

TD 5 — Etape 6 : Interpreter vmstat sous charge HTTP

Colonnes cles de vmstat sous charge HTTP :

Colonne	Signification	Valeur inquietante
`r` (run queue)	Processus prets a tourner	> nombre de CPU
`si/so` (swap in/out)	Echanges avec le swap	> 0 en continu
`bi/bo` (block I/O)	Lecture/ecriture disque	Eleve si logs verbeux
`us` (user CPU)	CPU user (Nginx workers)	Normal si eleve
`sy` (system CPU)	CPU noyau (appels systeme)	Anormal si > us
`wa` (I/O wait)	CPU en attente I/O	> 10% = disque trop lent

> Si `wa` est eleve mais `us` est faible pendant une charge HTTP, les logs sont probablement le goulot d'etranglement. Solution : utiliser `noatime` (deja fait en TD 3) ou basculer sur des logs asynchrones (`access_log ... buffer=32k`).

TD 5 — Etape 7 : Rotation des logs Nginx

20. Creez une configuration logrotate pour Nginx :

```
cat > /etc/logrotate.d/nginx-mediastream << 'EOF'
/var/log/nginx/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 nginx adm
    sharedscripts
    postrotate
        # Signaler a Nginx de reuvrir ses fichiers de log
        if [ -f /var/run/nginx.pid ]; then
            kill -USR1 $(cat /var/run/nginx.pid)
        fi
    endscript
}
EOF
```

21. Testez la rotation :

```
logrotate -vf /etc/logrotate.d/nginx-mediastream
ls -lh /var/log/nginx/
```

Resultat attendu : `access.log.1` cree, `access.log` vide, service Nginx toujours actif.

TD 5 — Checkpoint

- [ ] Logs Nginx centralises sur srv-web2 (rsyslog TCP port 514)
- [ ] Reception verifiee : `/var/log/remote/srv-web1/nginx-access.log` se remplit
- [ ] AIDE initialise sur srv-lb avec surveillance de `/etc/haproxy`
- [ ] Modification detectee par `aide --check` (hash different)
- [ ] Cron AIDE planifie a 3h du matin
- [ ] Logrotate Nginx : rotation testee avec `-vf`, service toujours actif

TD 6 — Boot et depannage

TD 6 — Objectif

Contexte MediaStream : Scenario catastrophe : un administrateur a perdu le mot de passe root de `srv-lb` apres une modification de GRUB. La configuration HAProxy a ete corrompue par une mauvaise manipulation. Vous devez retablir la situation selon des procedures documentees.

A la fin de ce TD, vous aurez :

- Sauvegarde et analyse le MBR de `srv-lb`
- Recupere un mot de passe root perdu via le mode rescue
- Repare une configuration GRUB corrompue
- Diagnostique et repare une configuration HAProxy invalide
- Documente les procedures de reprise

TD 6 — Etape 1 : Sauvegarder le MBR

1. Sur `srv-lb`, sauvegardez le MBR complet :

```
dd if=/dev/sda of=/root/mbr-srv-lb.bin bs=512 count=1
```

2. Sauvegardez uniquement le code bootloader (sans la table de partitions) :

```
dd if=/dev/sda of=/root/bootloader-srv-lb.bin bs=1 count=440
```

3. Analysez la signature de boot :

```
hexdump -C /root/mbr-srv-lb.bin | tail -3
```

Resultat attendu :

```
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa
```

> Les octets `55 AA` en position 510-511 sont la **signature MBR**. Sans eux, le BIOS considere le disque comme non bootable. Sauvegardez le MBR sur `srv-web2` via `scp` pour en avoir une copie hors de la machine.

TD 6 — Etape 1 : Structure du MBR

4. Copiez la sauvegarde sur `srv-web2` :

```
scp /root/mbr-srv-lb.bin msadmin@srv-web2:/home/msadmin/
```

5. Examinez les 440 premiers octets (code bootloader) :

```
hexdump -C /root/bootloader-srv-lb.bin | head -5
```

Resultat attendu : Code binaire correspondant au stage 1 de GRUB 2.

> Les 440 premiers octets = **GRUB stage 1** (code de demarrage). Les octets 446-509 = **table de partitions** (4 entrees primaires de 16 octets). Les octets 510-511 = signature `55 AA`. Modifier les 440 premiers octets detruit le bootloader sans toucher les partitions.

TD 6 — Etape 2 : Configuration GRUB

6. Examinez la configuration GRUB actuelle :

```
cat /etc/default/grub
```

Elements importants :

```
GRUB_TIMEOUT=5
GRUB_DEFAULT=saved
GRUB_CMDLINE_LINUX="... rd.lvm.lv=vg_lb/lv_root ..."
```

7. Explorez les entrees de demarrage generees :

```
grep -A5 "menuentry" /boot/grub2/grub.cfg | head -20
```

> `rd.lvm.lv=vg_lb/lv_root` dans la ligne de commande noyau indique a l'initramfs quel volume LVM activer pour trouver le systeme de fichiers racine. Sans ce parametre, le systeme ne demarre pas (panic noyau : "Unable to mount root fs").

TD 6 — Etape 3 : Recuperer un mot de passe root perdu

8. Simulez la perte du mot de passe root (changez-le pour en perdre la trace) :

```
# Notez l'ancien mot de passe quelque part, puis :
passwd root
# Entrez un nouveau mot de passe "perdu"
```

9. Redemarrez et au menu GRUB, appuyez sur `e` pour editer l'entree de boot.

10. Trouvez la ligne commencant par `linux` et ajoutez a la fin :

```
rd.break
```

11. Appuyez sur `Ctrl+X` pour booter.

TD 6 — Etape 3 : Shell d'urgence rd.break

12. Dans le shell d'urgence, le systeme est monte en lecture seule sur `/sysroot`. Remontez-le en lecture/ecriture :

```
mount -o remount,rw /sysroot
```

13. Entrez dans l'environnement du systeme (chroot) :

```
chroot /sysroot
```

14. Changez le mot de passe root :

```
passwd root
# Entrez le nouveau mot de passe
```

15. Indiquez a SELinux de reetiquer les fichiers au prochain demarrage :

```
touch /.autorelabel
exit
exit
```

Le systeme redemmarre. Apres le reetiketage SELinux (1-2 minutes), connectez-vous avec le nouveau mot de passe.

TD 6 — Etape 4 : Reparer GRUB apres corruption

16. Simulez une corruption du bootloader :

```
dd if=/dev/zero of=/dev/sda bs=1 count=440
```

> **Attention** : Cette commande detruit le stage 1 de GRUB. La machine ne pourra plus booter normalement jusqu'a la reparation. Executez-la uniquement dans un environnement de TP ou vous pouvez facilement acceder a la console VM.

17. Redemarrez : la machine affiche `GRUB>` (shell GRUB minimal) ou une erreur de boot.

18. Bootez depuis l'ISO Rocky Linux en mode **rescue** :

```
Troubleshooting -> Rescue a Rocky Linux system
```

TD 6 — Etape 4 : Reparation depuis le mode rescue

19. Dans l'environnement rescue, le systeme est monte sous `/mnt/sysimage`. Entrez en chroot :

```
chroot /mnt/sysimage
```

20. Reinstallez GRUB sur le disque :

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

21. Quittez le chroot et redemarrez :

```
exit
reboot
```

Resultat attendu : Le menu GRUB apparait normalement. Le systeme demarre.

> `grub2-install` reinstalle les 440 octets du stage 1 et le stage 1.5 dans l'espace entre le MBR et la premiere partition. `grub2-mkconfig` regenere `/boot/grub2/grub.cfg` en lisant `/etc/default/grub` et les scripts dans `/etc/grub.d/`.

TD 6 — Etape 5 : Diagnostiquer une config HAProxy invalide

22. Simulez une corruption de la configuration HAProxy :

```
# Introduire une directive invalide
echo "invalid_directive 999" >> /etc/haproxy/haproxy.cfg
systemctl restart haproxy
```

Resultat attendu : `systemctl restart haproxy` echoue.

23. Diagnostiquez avec les outils appropriés :

```
# Verification de syntaxe HAProxy (sans demarrer)
haproxy -c -f /etc/haproxy/haproxy.cfg

# Logs systemd
journalctl -u haproxy -n 30 --no-pager

# Etat detaille
systemctl status haproxy
```

Resultat attendu de `haproxy -c` : Numero de ligne et nature de l'erreur.

TD 6 — Etape 5 : Resolution et procedure documentee

24. Corrigez la configuration :

```
# Supprimer la ligne invalide
sed -i '/invalid_directive/d' /etc/haproxy/haproxy.cfg

# Valider avant de redemarrer
haproxy -c -f /etc/haproxy/haproxy.cfg

# Redemarrer seulement si la config est valide
systemctl start haproxy
```

25. Redigez la procedure de reprise (a inclure dans votre wiki interne) :

```
PROCEDURE : HAProxy ne demarre pas
1. haproxy -c -f /etc/haproxy/haproxy.cfg  -> identifier la ligne erreur
2. Corriger l'erreur identifiee
3. haproxy -c -f ... (re-valider)
4. systemctl start haproxy
5. curl -v http://localhost/ (valider le service)
6. Verifier les backends dans les stats :8080/haproxy-stats
```

TD 6 — Checkpoint

- [ ] MBR sauvegarde (`dd`) et copie sur srv-web2 (`scp`)
- [ ] Mot de passe root recupere via `rd.break` (procedure executee)
- [ ] GRUB repare depuis le mode rescue (`grub2-install` + `grub2-mkconfig`)
- [ ] `haproxy -c -f` : outil de validation maitrise
- [ ] Procedure de reprise HAProxy redigee (au moins 5 etapes)

TD 7 — Optimisation des performances

TD 7 — Objectif

Contexte MediaStream : Le site est en production mais les temps de reponse sont mediocres sous charge. La direction exige un benchmark avant/apres pour justifier les optimisations. Vous devez mesurer, identifier les goulots d'etranglement, optimiser, puis mesurer a nouveau.

A la fin de ce TD, vous aurez :

- Etabli une baseline de performances avec Apache Benchmark
- Optimise la configuration Nginx (workers, buffers, compression)
- Configure HAProxy pour la haute charge
- Mesure l'impact de chaque optimisation
- Analyse la consommation memoire du processus Nginx

TD 7 — Etape 1 : Installer les outils de benchmark

1. Sur votre poste hote ou `srv-nagios`, installez Apache Benchmark :

```
dnf install -y httpd-tools  # fournit 'ab'
```

2. Verifiez l'installation :

```
ab -V
```

Resultat attendu : `This is ApacheBench, Version 2.3 ...`

TD 7 — Etape 2 : Creer une page de test representative

3. Sur `srv-web1` et `srv-web2`, creez un contenu de test realiste :

```
# Page principale (~10 Ko)
python3 -c "
content = '<html><body><h1>MediaStream</h1>'
content += '<p>' + 'A' * 9500 + '</p>'
content += '</body></html>'
print(content)
" > /var/www/html/index.html

# Image factice (100 Ko)
dd if=/dev/urandom bs=1024 count=100 | base64 > /var/www/html/banner.jpg

# Script JS factice (20 Ko)
python3 -c "print('// MediaStream JS\\n' + 'var x = ' + '\"data\";\\n' * 1000)" \
    > /var/www/html/app.js
```

4. Verifiez que Nginx sert ces fichiers :

```
curl -sI http://localhost/index.html | grep -E "Content-Length|Content-Type"
```

TD 7 — Etape 3 : Baseline avant optimisation

5. Mesurez les performances **avant** toute optimisation :

```
# 1000 requetes, 50 connexions concurrentes, via HAProxy
ab -n 1000 -c 50 -g /tmp/ab-baseline.tsv http://192.168.56.10/index.html
```

Releves a noter dans un tableau de bord :

```
Requests per second:    XXX.XX [#/sec]
Time per request:       XX.XXX [ms] (mean)
Time per request:       XX.XXX [ms] (mean, across all concurrent requests)
Transfer rate:          XXXX.XX [Kbytes/sec] received

Percentage of the requests served within a certain time (ms)
  50%     XX
  90%     XX
  99%     XX
```

> Notez **toutes** ces valeurs. Ce sont vos metriques de reference. Sans baseline, il est impossible de prouver que vos optimisations ont eu un effet.

TD 7 — Etape 4 : Analyser la memoire Nginx

6. Analysez la consommation memoire des workers Nginx :

```
dnf install -y smem
smem -t -k -P nginx
```

Resultat attendu :

```
  PID User     Command                         Swap      USS      PSS      RSS
 1234 root     nginx: master process              0     1.1M     1.4M     5.6M
 1235 nginx    nginx: worker process              0     1.9M     2.6M     6.3M
 ...
```

7. Observez les zones memoire du master :

```
NGINX_PID=$(pgrep -o nginx)
cat /proc/$NGINX_PID/status | grep -E "^VmRSS|^VmSize|^Threads"
```

> **USS** (Unique Set Size) est la memoire reellement exclusive a ce processus. **PSS** (Proportional) est la metrique la plus juste en production car elle repartit la memoire partagee. `smem -t` donne le total reel utilise par Nginx, contrairement a `ps aux` qui somme le RSS et surestime la consommation.

TD 7 — Etape 5 : Optimiser la configuration Nginx

8. Sur `srv-web1` et `srv-web2`, sauvegardez la config par defaut :

```
cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.orig
```

9. Appliquez le profil de production MediaStream :

```
cat > /etc/nginx/nginx.conf << 'EOF'
user nginx;

# Auto-detecte le nombre de CPU (evite de le mettre a la main)
worker_processes auto;

# Limite de fichiers ouverts par worker
worker_rlimit_nofile 65535;

error_log /var/log/nginx/error.log warn;
pid /run/nginx.pid;

events {
    # Connexions simultanees par worker
    worker_connections 4096;
    # Accepter plusieurs connexions a la fois (evite les allers-retours)
    multi_accept on;
    # Methode I/O la plus efficace sur Linux
    use epoll;
}
EOF
```

TD 7 — Etape 5 : Optimiser le bloc HTTP

```
cat >> /etc/nginx/nginx.conf << 'EOF'

http {
    # Envoi direct depuis le kernel (zero-copy)
    sendfile on;
    tcp_nopush on;       # Accumule les donnees avant envoi (avec sendfile)
    tcp_nodelay on;      # Envoie immediatement les petits paquets

    # Keepalive : reutiliser les connexions TCP existantes
    keepalive_timeout 65;
    keepalive_requests 100;

    # Buffers : evite les allers-retours avec le disque pour les petites requetes
    client_body_buffer_size 16k;
    client_header_buffer_size 1k;
    client_max_body_size 8m;

    # Compression gzip pour le contenu HTML/JS/CSS
    gzip on;
    gzip_min_length 1000;
    gzip_types text/plain text/css application/javascript application/json;
    gzip_comp_level 4;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    access_log /var/log/nginx/access.log combined;
}
EOF
```

TD 7 — Etape 6 : Configurer le vhost Nginx

10. Creez le vhost MediaStream :

```
cat > /etc/nginx/conf.d/mediastream.conf << 'EOF'
server {
    listen 80;
    server_name _;
    root /var/www/html;
    index index.html;

    # Cache navigateur pour les assets statiques
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 7d;
        add_header Cache-Control "public, immutable";
    }

    # Page principale
    location / {
        try_files $uri $uri/ =404;
    }

    # Health check pour HAProxy (pas de log)
    location /health {
        access_log off;
        return 200 "OK\n";
        add_header Content-Type text/plain;
    }
}
EOF
```

11. Validez et rechargez :

TD 7 — Etape 7 : Optimiser HAProxy

12. Sur `srv-lb`, optimisez la configuration HAProxy pour la production :

```
# Dans le bloc global, ajoutez :
cat >> /etc/haproxy/haproxy.cfg << 'EOF'

# Tuning global pour haute charge
tune.maxaccept 100
tune.ssl.default-dh-param 2048
EOF
```

13. Dans le bloc `defaults`, verifiez et ajustez :

```
# Les timeouts suivants evitent des connexions "fantomes" qui consomment des ressources :
# timeout connect : delai pour etablir la connexion TCP avec le backend
# timeout client  : delai d'inactivite cote client
# timeout server  : delai d'inactivite cote serveur
```

14. Ajoutez la compression dans le backend :

```
# Dans le bloc backend, ajoutez :
# compression algo gzip
# compression type text/html text/plain application/javascript
```

TD 7 — Etape 8 : Mesure apres optimisation

15. Appliquez la configuration Nginx optimisee et relancez le benchmark :

```
systemctl reload nginx  # sur srv-web1 et srv-web2
haproxy -c -f /etc/haproxy/haproxy.cfg && systemctl reload haproxy

# Meme benchmark qu'a l'etape 3
ab -n 1000 -c 50 -g /tmp/ab-apres.tsv http://192.168.56.10/index.html
```

16. Comparez les resultats :

```
# Requests per second (avant vs apres)
grep "^Requests" /tmp/ab-baseline.tsv 2>/dev/null || echo "Comparez manuellement"
```

Tableau de bord attendu :

Metrique	Avant	Apres	Delta
Req/sec	_	_	+_%
Temps moyen (ms)	_	_	-_%
99e percentile (ms)	_	_	-_%

> Si les gains sont faibles en VM locale, c'est normal : le goulot d'etranglement est souvent la vitesse du disque virtuel, pas la config logicielle. Sur un vrai serveur avec SSD NVMe, `sendfile` et `gzip` font une difference significative.

TD 7 — Checkpoint

- [ ] Baseline enregistree (req/sec, temps moyen, p99)
- [ ] `smem -t -k -P nginx` execute et valeurs notees
- [ ] Configuration Nginx optimisee : `worker_processes auto`, `epoll`, `sendfile`, `gzip`
- [ ] Vhost MediaStream avec route `/health` (retourne 200 OK)
- [ ] Benchmark post-optimisation execute
- [ ] Tableau comparatif avant/apres complete

TD 8 — Supervision avec Nagios

TD 8 — Objectif

Contexte MediaStream : L'infrastructure est en production et optimisee. La derniere brique : une supervision proactive capable de detecter les problemes avant les utilisateurs — qu'il s'agisse d'un backend Nginx tombe, d'HAProxy sature, ou d'un disque `/var/www` plein.

A la fin de ce TD, vous aurez :

- Installe et configure Nagios sur `srv-nagios`
- Declare les 4 machines de l'infrastructure
- Configure NRPE pour les checks distants (Nginx, HAProxy)
- Ecrit un plugin custom pour surveiller l'etat des backends via les stats HAProxy
- Simule 3 scenarios de pannes et verifie leur detection

TD 8 — Etape 1 : Installer Nagios

1. Sur `srv-nagios`, installez Nagios et ses plugins :

```
dnf install -y epel-release
dnf install -y nagios nagios-plugins-all nagios-plugins-nrpe httpd php
```

2. Configurez le mot de passe de l'interface web :

```
htpasswd -c /etc/nagios/passwd nagiosadmin
```

3. Demarrez les services :

```
systemctl enable --now nagios httpd
firewall-cmd --permanent --add-service=http
firewall-cmd --reload
```

4. Testez l'acces : ouvrez `http://192.168.56.13/nagios/` dans un navigateur.

Resultat attendu : La page de login Nagios s'affiche.

TD 8 — Etape 2 : Creer les templates MediaStream

5. Creez les templates de base :

```
cat > /etc/nagios/objects/mediastream-templates.cfg << 'EOF'
define host {
    name                    ms-server
    use                     linux-server
    check_interval          5
    retry_interval          1
    max_check_attempts      3
    notification_interval   30
    register                0
}

define service {
    name                    ms-service
    use                     generic-service
    check_interval          5
    retry_interval          1
    max_check_attempts      3
    notification_interval   30
    register                0
}
EOF
```

> `register 0` signifie que ce sont des **templates** (modeles), pas des objets reels. Ils servent de base via `use`. Changer `check_interval` dans le template suffit a modifier la frequence de **tous** les services qui en heritent.

TD 8 — Etape 3 : Declarer les hotes

6. Creez les definitions d'hotes MediaStream :

```
cat > /etc/nagios/objects/mediastream-hosts.cfg << 'EOF'
define hostgroup {
    hostgroup_name  ms-web-backends
    alias           Backends Web MediaStream
    members         srv-web1,srv-web2
}

define hostgroup {
    hostgroup_name  ms-all-servers
    alias           Tous les serveurs MediaStream
    members         srv-lb,srv-web1,srv-web2
}

define host {
    use         ms-server
    host_name   srv-lb
    alias       Load Balancer HAProxy
    address     192.168.56.10
}

define host {
    use         ms-server
    host_name   srv-web1
    alias       Serveur Web Nginx 1
    address     192.168.56.11
}

define host {
    use         ms-server
    host_name   srv-web2
    alias       Serveur Web Nginx 2
    address     192.168.56.12
}
EOF
```

TD 8 — Etape 4 : Services reseau de base (sans NRPE)

7. Ajoutez les checks reseau de base :

```
cat > /etc/nagios/objects/mediastream-services-net.cfg << 'EOF'
# Ping de toutes les machines
define service {
    use                 ms-service
    hostgroup_name      ms-all-servers
    service_description PING
    check_command       check_ping!100.0,20%!500.0,60%
}

# HTTP sur le frontend HAProxy (la porte d'entree des utilisateurs)
define service {
    use                 ms-service
    host_name           srv-lb
    service_description HTTP Frontend
    check_command       check_http!-H srv-lb.mediastream.lan -p 80 -u /health
}

# HTTP direct sur chaque backend (hors HAProxy)
define service {
    use                 ms-service
    hostgroup_name      ms-web-backends
    service_description HTTP Backend Direct
    check_command       check_http!-p 80 -u /health
}
EOF
```

TD 8 — Etape 5 : Installer NRPE sur les serveurs

8. Sur `srv-lb`, `srv-web1` et `srv-web2`, installez NRPE :

```
dnf install -y nrpe nagios-plugins-all
```

9. Configurez NRPE pour autoriser `srv-nagios` :

```
sed -i 's/^allowed_hosts=.*/allowed_hosts=127.0.0.1,::1,192.168.56.13/' \
    /etc/nagios/nrpe.cfg
```

10. Demarrez NRPE et ouvrez le pare-feu :

```
systemctl enable --now nrpe
firewall-cmd --permanent --add-port=5666/tcp
firewall-cmd --reload
```

11. Testez depuis `srv-nagios` :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.10
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.11
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.12
```

Resultat attendu : `NRPE v4.x.x` sur les trois machines.

TD 8 — Etape 5 : Commandes NRPE par machine

12. Sur `srv-web1` et `srv-web2`, ajoutez les commandes NRPE pour Nginx :

```
cat > /etc/nrpe.d/mediastream.cfg << 'EOF'
# Disques
command[check_disk_root]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /
command[check_disk_www]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /var/www/html
command[check_disk_logs]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /var/log/nginx

# Systeme
command[check_load]=/usr/lib64/nagios/plugins/check_load -w 3,2,1.5 -c 6,4,3
command[check_mem]=/usr/lib64/nagios/plugins/check_swap -w 30% -c 10%

# Nginx
command[check_nginx_proc]=/usr/lib64/nagios/plugins/check_procs -C nginx -w 2:20 -c 1:50
command[check_nginx_http]=/usr/lib64/nagios/plugins/check_http -H localhost -p 80 -u /health
EOF

systemctl restart nrpe
```

TD 8 — Etape 6 : Plugin custom — Etat des backends HAProxy

13. Sur `srv-lb`, creez un plugin NRPE pour surveiller les backends HAProxy :

```
cat > /usr/lib64/nagios/plugins/check_haproxy_csv << 'EOF'
#!/bin/bash
# Surveille les backends HAProxy via la sortie CSV des stats
# Usage: check_haproxy_csv [backend_name]

BACKEND="${1:-novatech_backends}"
STATS_SOCKET="/var/lib/haproxy/stats"

# Recupere les stats via le socket Unix HAProxy
DATA=$(echo "show stat" | socat stdio $STATS_SOCKET 2>/dev/null)

if [ $? -ne 0 ] || [ -z "$DATA" ]; then
    echo "CRITICAL - Cannot connect to HAProxy socket $STATS_SOCKET"
    exit 2
fi

DOWN=$(echo "$DATA" | awk -F',' -v b="$BACKEND" '$1==b && $2!="FRONTEND" && $2!="BACKEND" && $18=="DOWN"' | wc -l)
UP=$(echo "$DATA"   | awk -F',' -v b="$BACKEND" '$1==b && $2!="FRONTEND" && $2!="BACKEND" && $18=="UP"'   | wc -l)
TOTAL=$((DOWN + UP))

if   [ "$TOTAL" -eq 0 ]; then
    echo "UNKNOWN - No servers found in backend $BACKEND"
    exit 3
elif [ "$DOWN" -eq 0 ]; then
    echo "OK - All $TOTAL servers UP ($BACKEND)"
    exit 0
elif [ "$DOWN" -lt "$TOTAL" ]; then
    echo "WARNING - $DOWN/$TOTAL servers DOWN ($BACKEND)"
    exit 1
else
    echo "CRITICAL - ALL $TOTAL servers DOWN ($BACKEND)"
    exit 2
fi
EOF

chmod +x /usr/lib64/nagios/plugins/check_haproxy_csv
dnf install -y socat
```

TD 8 — Etape 6 : Enregistrer le plugin dans NRPE

14. Sur `srv-lb`, ajoutez les commandes NRPE pour HAProxy :

```
cat > /etc/nrpe.d/mediastream-lb.cfg << 'EOF'
# Etat des backends via socket HAProxy
command[check_haproxy_backends]=/usr/lib64/nagios/plugins/check_haproxy_csv novatech_backends
command[check_haproxy_proc]=/usr/lib64/nagios/plugins/check_procs -C haproxy -w 1:5 -c 1:10

# Disques
command[check_disk_root]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /
command[check_disk_var]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /var

# Systeme
command[check_load]=/usr/lib64/nagios/plugins/check_load -w 3,2,1.5 -c 6,4,3
EOF

systemctl restart nrpe
```

15. Testez le plugin depuis `srv-nagios` :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.10 -c check_haproxy_backends
```

Resultat attendu : `OK - All 2 servers UP (novatech_backends)`

TD 8 — Etape 7 : Services NRPE dans Nagios

16. Sur `srv-nagios`, creez les services NRPE :

```
cat > /etc/nagios/objects/mediastream-services-nrpe.cfg << 'EOF'
# --- srv-lb (HAProxy) ---
define service {
    use                 ms-service
    host_name           srv-lb
    service_description HAProxy Backends
    check_command       check_nrpe!check_haproxy_backends
}
define service {
    use                 ms-service
    host_name           srv-lb
    service_description HAProxy Process
    check_command       check_nrpe!check_haproxy_proc
}

# --- backends web (commun) ---
define service {
    use                 ms-service
    hostgroup_name      ms-web-backends
    service_description Nginx Process
    check_command       check_nrpe!check_nginx_proc
}
define service {
    use                 ms-service
    hostgroup_name      ms-web-backends
    service_description Nginx HTTP
    check_command       check_nrpe!check_nginx_http
}
define service {
    use                 ms-service
    hostgroup_name      ms-web-backends
    service_description Disk /var/www
    check_command       check_nrpe!check_disk_www
}
define service {
    use                 ms-service
    hostgroup_name      ms-web-backends
    service_description Disk /var/log/nginx
    check_command       check_nrpe!check_disk_logs
}
EOF
```

TD 8 — Etape 7 : Validation Nagios

17. Enregistrez les fichiers dans nagios.cfg et validez :

```
# Verifiez que cfg_dir=/etc/nagios/objects est bien present dans nagios.cfg
grep "cfg_dir=/etc/nagios/objects" /etc/nagios/nagios.cfg

# Validation complete
nagios -v /etc/nagios/nagios.cfg
```

Resultat attendu : `Things look okay - No serious problems were detected`

18. Rechargez Nagios et observez l'interface :

```
systemctl reload nagios
```

Attendez 2-3 cycles (10-15 minutes) que tous les services passent en vert.

TD 8 — Etape 8 : Scenario 1 — Panne d'un backend Nginx

19. Arretez Nginx sur `srv-web1` :

```
# Sur srv-web1 :
systemctl stop nginx
```

20. Forcez un check immediat depuis `srv-nagios` :

```
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.11 -c check_nginx_proc
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.10 -c check_haproxy_backends
```

Resultats attendus :

- `srv-web1 Nginx Process` : **CRITICAL** — nombre de processus = 0
- `srv-lb HAProxy Backends` : **WARNING** — 1/2 servers DOWN

> **Pourquoi WARNING et pas CRITICAL ?** Parce que HAProxy a encore un backend actif (srv-web2) et que le service reste accessible aux utilisateurs. La distinction WARNING/CRITICAL est cle pour prioriser les interventions : WARNING = degradation, CRITICAL = interruption.

TD 8 — Etape 8 : Scenario 2 — Disque `/var/www` plein

21. Remplissez `/var/www/html` sur `srv-web2` :

```
# Sur srv-web2 :
dd if=/dev/zero of=/var/www/html/bigfile bs=1M count=2800
df -h /var/www/html
```

22. Verifiez le check disque :

```
# Depuis srv-nagios :
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.12 -c check_disk_www
```

Resultat attendu : `DISK CRITICAL - free space: /var/www/html ...` (< 10% restants).

23. Nettoyez et verifiez le retour a la normale :

```
# Sur srv-web2 :
rm /var/www/html/bigfile
```

Resultat attendu : Apres le prochain cycle de check, `Disk /var/www` repasse en **OK** (vert).

TD 8 — Etape 8 : Scenario 3 — HAProxy down

24. Arretez completement HAProxy :

```
# Sur srv-lb :
systemctl stop haproxy
```

25. Depuis `srv-nagios`, observez la cascade d'alertes :

```
# Verifiez le frontend HTTP
/usr/lib64/nagios/plugins/check_http -H 192.168.56.10 -p 80 -u /health

# Verifiez le process HAProxy
/usr/lib64/nagios/plugins/check_nrpe -H 192.168.56.10 -c check_haproxy_proc
```

Resultats attendus :

- `HTTP Frontend` : **CRITICAL** — connexion refusee
- `HAProxy Process` : **CRITICAL** — 0 processus detectes
- `HAProxy Backends` : **CRITICAL** — socket HAProxy inaccessible

26. Redemarrez HAProxy et verifiez le retour en vert :

```
# Sur srv-lb :
haproxy -c -f /etc/haproxy/haproxy.cfg && systemctl start haproxy
```

TD 8 — Checkpoint

- [ ] Nagios accessible sur http://192.168.56.13/nagios/
- [ ] 3 hotes declares + 2 hostgroups (ms-web-backends, ms-all-servers)
- [ ] NRPE installe et actif sur srv-lb, srv-web1, srv-web2
- [ ] Plugin `check_haproxy_csv` fonctionne : `OK - All 2 servers UP`
- [ ] Tous les services en OK (vert) au repos
- [ ] Scenario 1 : srv-web1 arrete -> WARNING HAProxy Backends + CRITICAL Nginx
- [ ] Scenario 2 : disque /var/www plein -> CRITICAL Disk detecte
- [ ] Scenario 3 : HAProxy stoppe -> cascade CRITICAL sur 3 services

Synthese du projet MediaStream

Recapitulatif de l'infrastructure

A l'issue des 8 TD, vous avez construit une infrastructure web haute disponibilite complete :

Composant	TD	Ce qui a ete fait
Installation reproductible	TD 1	Kickstart + RAID1 + LVM + GRUB securise
Gestion logicielle	TD 2	Depot local + Nginx + HAProxy maitrise
Stockage resilient	TD 3	LV dedies + XFS + snapshots + extension a chaud
Tuning systeme	TD 4	sysctl HTTP + udev + modules noyau
Surveillance	TD 5	Logs centralises + AIDE + logrotate + vmstat
Plan de reprise	TD 6	MBR + GRUB + HAProxy config + procedures
Performances	TD 7	Baseline + Nginx/HAProxy optimises + mesure
Supervision	TD 8	Nagios + NRPE + plugin HAProxy + alerting

Competences acquises

- **Deployer** : Installation automatisee, RAID, LVM, 4 serveurs en < 1h avec Kickstart
- **Maîtriser les paquets** : RPM, dependances, depot prive, controle des versions
- **Stocker** : XFS pour les logs, snapshots LVM comme filet de securite
- **Comprendre le noyau** : Modules, udev, sysctl pour HTTP haute charge
- **Securiser** : GRUB, AIDE, logs centralises, firewall
- **Diagnostiquer** : `haproxy -c`, `journalctl`, `vmstat`, `smem`, `ab`
- **Reparer** : MBR, GRUB, mot de passe root, config HAProxy invalide
- **Optimiser** : Benchmark avant/apres, tuning Nginx/HAProxy mesure
- **Superviser** : Nagios, NRPE, plugin custom, alerting differencie

Specificites de l'infrastructure web HA

Problematique	Solution deployee
Point unique de defaillance	HAProxy + 2 backends Nginx
Crash d'un backend	Bascule automatique (fall 2 / rise 3)
Deploiement problematique	Snapshot LVM avant deploy
Logs non maitrisables	LV dedie + logrotate + centralisation
Saturation des connexions TCP	sysctl : somaxconn, tcp_fin_timeout
Detection des pannes	Nagios + plugin check_haproxy_csv

Pour aller plus loin

- **Keepalived** : VIP flottante sur srv-lb pour eliminer le SPOF du load balancer lui-meme
- **SSL/TLS** : Termination HTTPS sur HAProxy avec Let's Encrypt
- **HAProxy ACLs** : Routage par URL, par header, par methode HTTP
- **Nginx cache** : `proxy_cache` pour les backends dynamiques
- **Prometheus + Grafana** : Metriques temps reel (complement moderne de Nagios)
- **Ansible** : Automatiser le deploiement de toute cette infrastructure en 1 commande

> L'infrastructure MediaStream est desormais tolerante aux pannes, optimisee et supervisee. Un incident sur un backend est **detecte en 5 minutes** et **sans impact utilisateur** grace au failover HAProxy.

Checklist globale du projet

Progression

- [ ] **TD 1** : 4 serveurs installes, RAID+LVM, GRUB securise
- [ ] **TD 2** : Depot local, Nginx et HAProxy installes et verifies
- [ ] **TD 3** : LV dedies, snapshot, restauration, extension a chaud
- [ ] **TD 4** : sysctl HTTP, regle udev, module reseau inspecte
- [ ] **TD 5** : Logs centralises, AIDE sur HAProxy, logrotate Nginx
- [ ] **TD 6** : MBR sauvegarde, GRUB repare, config HAProxy validee
- [ ] **TD 7** : Baseline etablie, Nginx/HAProxy optimises, delta mesure
- [ ] **TD 8** : Supervision complete, plugin custom, 3 scenarios valides

> Chaque TD valide = une brique de plus dans l'infrastructure de production MediaStream.