Installation NVIDIA GPU sur Proxmox avec passthrough LXC

Rémi Albertucci

Table des matières

I Contexte et prérequis.....3

I.1 Environnement cible.....3

I.2 Ce qu'on va faire.....3

I.3 Prérequis système.....3

II Installation du driver NVIDIA sur l'hôte Proxmox.....4

II.1 Identifier le bon driver.....4

II.2 Installation des dépendances.....4

II.3 Téléchargement et installation du driver.....4

II.4 Vérification.....5

III Création du LXC et passthrough GPU.....6

III.1 Créer le conteneur LXC.....6

III.2 Éditer la configuration LXC.....6

Méthode 1 : cgroup2 + dev (recommandée, environnement de production).....6

Méthode 2 : mount.entry (alternative, non recommandée en prod).....7

III.3 Vérification dans le LXC.....7

IV Installation du driver NVIDIA dans le LXC.....8

V Installation de Docker dans le LXC.....9

Il s'agit de l'installation « classique » de docker, que vous pourrez retrouver à l'adresse :.....9

https://docs.docker.com/engine/install/debian/.....9

V.1 Suppression des anciennes versions.....9

V.2 Ajout du repo officiel Docker.....9

V.3 Installation.....9

V.4 Vérification.....10

VI Configuration du NVIDIA Container Toolkit.....11

VI.1 Ajout du repo et installation.....11

VI.2 Configuration du runtime Docker 11

VI.3 Test du runtime NVIDIA.....12

VII Fix automatique des cgroup majors NVIDIA.....13

VII.1 Contexte du problème.....13

VII.2 Script /usr/local/bin/fix-nvidia-lxc.sh.....13

VII.3 Service systemd /etc/systemd/system/nvidia-lxc-fix.service.....14

VII.4 Activation du service.....15

VIII Fix manuel (si besoin urgent) 16

IX Vérification globale.....17

IX.1 Commandes de vérification.....17

IX.2 Résultat attendu (nvidia-smi)…17

IX.3 Séquence de démarrage complète 17

X Notes et points d'attention.….18

I Contexte et prérequis

I.1 Environnement cible

Ce guide couvre l'installation complète d'un GPU NVIDIA sur un hôte Proxmox VE, avec passthrough vers un conteneur LXC, installation de Docker et configuration du NVIDIA Container Toolkit.

- Hyperviseur : Proxmox VE (testé avec kernel 6.x)
- GPU : NVIDIA Quadro P1000 (applicable à tout GPU NVIDIA compatible CUDA)
- Conteneur : LXC Debian (non-privileged ou privileged selon config cgroup)
- Objectif final : transcodage HEVC/CUDA via ffmpeg dans le LXC, avec support Docker

I.2 Ce qu'on va faire

- Installer le driver NVIDIA propriétaire sur l'hôte Proxmox
- Configurer le passthrough des devices NVIDIA vers le LXC (/dev/nvidia*)
- Installer le driver NVIDIA dans le LXC (sans kernel modules)
- Installer Docker et le NVIDIA Container Toolkit
- Mettre en place un fix automatique des cgroup majors (problème post-reboot)

I.3 Prérequis système

- Accès root sur l'hôte Proxmox
- GPU NVIDIA présent et reconnu par le BIOS/UEFI
- Connexion internet depuis l'hôte et depuis le LXC
- Kernel headers disponibles pour la version en cours

Effectuer une sauvegarde ou snapshot Proxmox avant toute manipulation des drivers.

II Installation du driver NVIDIA sur l'hôte Proxmox

II.1 Identifier le bon driver

Consulter la page officielle NVIDIA pour choisir le driver adapté au GPU et au kernel :

https://www.nvidia.com/en-us/drivers/unix/

Exemple utilisé dans ce guide : driver 580.126.18 pour Linux x86_64.

II.2 Installation des dépendances

Sur l'hôte Proxmox, installer les headers kernel et les outils de compilation :

```
apt install -y proxmox-default-headers proxmox-headers-$(uname -r) gcc make dkms
apt install linux-headers-$(uname -r) -y
```

II.3 Téléchargement et installation du driver

Télécharger le fichier .run depuis le lien NVIDIA, puis l'installer avec support DKMS (permet la recompilation automatique après une mise à jour kernel)

Télécharger le driver (adapter l'URL selon le GPU/version choisie)

```
wget https://fr.download.nvidia.com/XFree86/Linux-x86_64/580.126.18/NVIDIA-Linux-x86_64-580.126.18.run
chmod +x NVIDIA-Linux-x86_64-580.126.18.run
```

Installation avec DKMS et désactivation de Nouveau (drivers préinstallés) :

```
./NVIDIA-Linux-x86_64-580.126.18.run --dkms --disable-nouveau \
--kernel-module-type proprietary --no-install-libglvnd
```

Alternative simplifiée (avec questions interactives)

```
./$(ls -t NVIDIA*.run | head -n 1) --dkms
```

L'option --dkms garantit que le module sera recompilé automatiquement après une mise à jour du kernel Proxmox.

II.4 Vérification

Après l'installation, vérifier que le GPU est bien détecté :

```
nvidia-smi
```

Résultat attendu :

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.126.18             Driver Version: 580.126.18     CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Quadro P1000                   Off |   00000000:01:00.0 Off |                  N/A |
| 52%   51C    P0            N/A  /  N/A  |       0MiB /   4096MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Lister les devices créés

```
ls -la /dev/nvidia* /dev/nvidia-caps/*
```

Noter les numéros majors affichés, ils seront utilisés pour la config LXC

```
crw-rw-rw- 1 root root 195,   0 May  1 23:06 /dev/nvidia0
cr-------- 1 root root 511,   1 May  1 23:06 /dev/nvidia-caps/nvidia-cap1
cr--r--r-- 1 root root 511,   2 May  1 23:06 /dev/nvidia-caps/nvidia-cap2
crw-rw-rw- 1 root root 195, 255 May  1 23:06 /dev/nvidiactl
crw-rw-rw- 1 root root 195, 254 May  1 23:06 /dev/nvidia-modeset
crw-rw-rw- 1 root root 508,   0 May  1 23:06 /dev/nvidia-uvm
crw-rw-rw- 1 root root 508,   1 May  1 23:06 /dev/nvidia-uvm-tools

/dev/nvidia-caps:
total 0
drwxr-xr-x  2 root root     80 May  1 23:06 .
drwxr-xr-x 22 root root   4800 May  3 07:03 ..
cr--------  1 root root 511, 1 May  1 23:06 nvidia-cap1
cr--r--r--  1 root root 511, 2 May  1 23:06 nvidia-cap2
root@proxmox:~# ls -la /dev/nvidia* /dev/nvidia-caps/*
crw-rw-rw- 1 root root 195,   0 May  1 23:06 /dev/nvidia0
cr-------- 1 root root 511,   1 May  1 23:06 /dev/nvidia-caps/nvidia-cap1
cr--r--r-- 1 root root 511,   2 May  1 23:06 /dev/nvidia-caps/nvidia-cap2
crw-rw-rw- 1 root root 195, 255 May  1 23:06 /dev/nvidiactl
crw-rw-rw- 1 root root 195, 254 May  1 23:06 /dev/nvidia-modeset
crw-rw-rw- 1 root root 508,   0 May  1 23:06 /dev/nvidia-uvm
crw-rw-rw- 1 root root 508,   1 May  1 23:06 /dev/nvidia-uvm-tools

/dev/nvidia-caps:
total 0
drwxr-xr-x  2 root root     80 May  1 23:06 .
drwxr-xr-x 22 root root   4800 May  3 07:03 ..
cr--------  1 root root 511, 1 May  1 23:06 nvidia-cap1
cr--r--r--  1 root root 511, 2 May  1 23:06 nvidia-cap2
```

III Création du LXC et passthrough GPU

III.1 Créer le conteneur LXC

Créer le conteneur LXC depuis l'interface Proxmox (ou via pct create). Retenir l'ID du conteneur (ex. 104). Le conteneur peut être unprivileged, la config cgroup gère les permissions.

III.2 Éditer la configuration LXC

Ouvrir le fichier de configuration du conteneur — remplacer XXX par l'ID du LXC :

```
nano /etc/pve/lxc/XXX.conf
```

Deux méthodes sont disponibles pour exposer les devices NVIDIA au conteneur :

Méthode 1 : cgroup2 + dev (recommandée, environnement de production)

Cette méthode donne un contrôle fin sur les permissions et limite l'accès aux seuls devices nécessaires. Adapter les numéros majors aux valeurs réelles affichées par ls -la /dev/nvidia* :

```
lxc.cgroup2.devices.allow: c 195:* rwm
lxc.cgroup2.devices.allow: c 507:* rwm
lxc.cgroup2.devices.allow: c 510:* rwm
```

Exposer les devices dans le conteneur

```
dev0: /dev/nvidia0,gid=0,uid=0,mode=0666
dev1: /dev/nvidiactl,gid=0,uid=0,mode=0666
dev2: /dev/nvidia-uvm,gid=0,uid=0,mode=0666
dev3: /dev/nvidia-uvm-
tools,gid=0,uid=0,mode=0666
dev4: /dev/nvidia-caps/nvidia-
cap1,gid=0,uid=0,mode=0666
dev5: /dev/nvidia-caps/nvidia-
cap2,gid=0,uid=0,mode=0666
```

Les numéros majors de nvidia-uvm et nvidia-caps changent à chaque reboot. Voir Chapitre 7 pour le fix automatique.

Méthode 2 : mount.entry (alternative, non recommandée en prod)

Plus simple à configurer, mais donne accès à tous les devices cgroup (lxc.cgroup2.devices.allow: a). À réserver aux environnements de test ou machines dédiées :

```
# ATTENTION : accès global à tous les devices cgroup — ne pas utiliser en prod partagé
lxc.cgroup2.devices.allow: a
lxc.mount.entry: /dev/nvidia0 dev/nvidia0 none bind,optional,create=file
lxc.mount.entry: /dev/nvidiactl dev/nvidiactl none bind,optional,create=file
lxc.mount.entry: /dev/nvidia-uvm dev/nvidia-uvm none bind,optional,create=file
lxc.mount.entry: /dev/nvidia-uvm-tools dev/nvidia-uvm-tools none bind,optional,create=file
lxc.mount.entry: /dev/nvidia-caps/nvidia-cap1 dev/nvidia-caps/nvidia-cap1 none bind,optional,create=file
lxc.mount.entry: /dev/nvidia-caps/nvidia-cap2 dev/nvidia-caps/nvidia-cap2 none bind,optional,create=file
```

III.3 Vérification dans le LXC

Démarrer le conteneur et vérifier que les devices sont bien accessibles :

```
pct start XXX
pct enter XXX

# Dans le conteneur :
ls -la /dev/nvidia*
```

Si les devices sont listés, le passthrough est opérationnel.

IV Installation du driver NVIDIA dans le LXC

Le driver doit être installé dans le LXC avec la même version que sur l'hôte, mais sans compiler de module kernel (le module tourne déjà sur l'hôte).

Dans le conteneur LXC :

```
wget https://fr.download.nvidia.com/XFree86/Linux-x86_64/580.126.18/NVIDIA-Linux-x86_64-580.126.18.run
chmod +x NVIDIA-Linux-x86_64-580.126.18.run
```

L'option --no-kernel-modules est obligatoire dans un LXC

```
./NVIDIA-Linux-x86_64-580.126.18.run --no-kernel-modules
```

Vérifier que le GPU est accessible depuis le LXC :

```
nvidia-smi
```

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.126.18             Driver Version: 580.126.18     CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Quadro P1000                   Off |   00000000:01:00.0 Off |                  N/A |
| 52%   51C    P0            N/A  /  N/A  |       0MiB /   4096MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

La version du driver dans le LXC DOIT correspondre exactement à la version installée sur l'hôte Proxmox.

V Installation de Docker dans le LXC

Il s'agit de l'installation « classique » de docker, que vous pourrez retrouver à l'adresse :

https://docs.docker.com/engine/install/debian/

V.1 Suppression des anciennes versions

```
sudo apt remove $(dpkg --get-selections docker.io
docker-compose docker-doc \
podman-docker containerd runc | cut -f1)
```

V.2 Ajout du repo officiel Docker

```
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL
https://download.docker.com/linux/debian/gpg
\
-o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Ajouter le dépôt
sudo tee /etc/apt/sources.list.d/docker.sources
<<EOF
Types: deb
URIs:
https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo
"$VERSION_CODENAME")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF
sudo apt update
```

V.3 Installation

```
sudo apt install docker-ce docker-ce-cli
containerd.io \
```

```
docker-buildx-plugin docker-compose-plugin -y
```

V.4 Vérification

```
systemctl status docker
```

Le service doit être active (running)

VI Configuration du NVIDIA Container Toolkit

Le NVIDIA Container Toolkit permet aux conteneurs Docker d'accéder au GPU. Une configuration spécifique est nécessaire dans un environnement LXC.

VI.1 Ajout du repo et installation

```
apt update && apt install -y gpg curl --no-install-recommends

curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey \
| gpg --dearmor > /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg

cat <<EOF > /etc/apt/sources.list.d/nvidia-container-toolkit.sources
Types: deb
URIs: http://nvidia.github.io/libnvidia-container/stable/deb/amd64/
Suites: /
Components:
Signed-By: /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
EOF

apt update && apt install -y nvidia-container-toolkit
```

VI.2 Configuration du runtime Docker

Configurer le runtime NVIDIA comme runtime par défaut pour Docker, puis activer l'option `no-cgroups`. Essentiel en LXC car Docker n'a pas accès direct aux cgroups :

```
nvidia-ctk runtime configure --runtime=docker --set-as-default

# Option CRITIQUE en LXC : désactive la gestion cgroups par nvidia-container-cli
```

```
# (les cgroups sont déjà gérés par la config LXC sur l'hôte)
nvidia-ctk config -i --set nvidia-container-cli.no-cgroups=true

systemctl restart docker
```

Sans l'option no-cgroups=true, les conteneurs Docker ne pourront pas accéder au GPU depuis un LXC (erreur d'accès cgroup).

VI.3 Test du runtime NVIDIA

On lance un conteneur de test avec accès GPU :

```
docker run --rm --gpus all nvidia/cuda:13.0.1-base-ubuntu22.04 nvidia-smi
```

Le résultat doit afficher le GPU Quadro P1000

```
Unable to find image 'nvidia/cuda:13.0.1-base-ubuntu22.04' locally
13.0.1-base-ubuntu22.04: Pulling from nvidia/cuda
60d98d907669: Pull complete
7e9d3b636b44: Pull complete
3a2ba8ed1759: Pull complete
5eaf68e8e556: Pull complete
c03b8ec8dd33: Pull complete
Digest: sha256:5543ce69b0d2d65b0c4ad978bd26f4cd67e4fccc629c262902ced0502e23cd7f
Status: Downloaded newer image for nvidia/cuda:13.0.1-base-ubuntu22.04
Sun May  3 13:25:06 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.126.18             Driver Version: 580.126.18     CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Quadro P1000                   Off |   00000000:01:00.0 Off |                  N/A |
| 34%   38C    P8            N/A  /  N/A  |      45MiB /   4096MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

VII Fix automatique des cgroup majors NVIDIA

VII.1 Contexte du problème

Le kernel Linux attribue dynamiquement les numéros majors des devices NVIDIA à chaque chargement du module nvidia_uvm. Ces numéros changent après un reboot ou une mise à jour du driver, mais la configuration LXC conserve les anciens numéros, causant un blocage cgroup.

Symptôme dans le LXC après un reboot ou mise à jour driver :

```
[AVHWDeviceContext] cu->cuInit(0) failed ->
CUDA_ERROR_UNKNOWN: unknown error
Device creation failed: -542398533.
[dec:hevc] No device available for decoder: device
type cuda needed for codec hevc.
Hardware device setup failed for decoder: Generic
error in an external library
```

Seul le major 195 (nvidia0/nvidiactl) est stable. Les majors nvidia_uvm et nvidia-caps sont attribués dynamiquement.

VII.2 Script /usr/local/bin/fix-nvidia-lxc.sh

Ce script lit les numéros majors réels au moment du boot et met à jour automatiquement la configuration LXC :

```
#!/bin/bash
# Fix automatique des cgroup majors NVIDIA
pour LXC 104

LXC_CONF="/etc/pve/lxc/104.conf"

modprobe nvidia_uvm 2>/dev/null

# Attendre jusqu'à 30s que les devices soient
disponibles
for i in $(seq 1 30); do
UVM_MAJOR=$(ls -la /dev/nvidia-uvm
2>/dev/null | awk '{print $5}' | tr -d ',')
CAPS_MAJOR=$(ls -la /dev/nvidia-caps/nvidia-
```

```
cap1 2>/dev/null | awk '{print $5}' | tr -d ',')
[[ -n "$UVM_MAJOR" && -n "$CAPS_MAJOR" ]]
&& break
echo "Attente devices NVIDIA... ($i/30)"
sleep 1
done

if [[ -z "$UVM_MAJOR" || -z "$CAPS_MAJOR" ]];
then
echo "ERREUR: devices NVIDIA introuvables
après 30s"
exit 1
fi

echo "Majors détectés -> nvidia-uvm:
$UVM_MAJOR | nvidia-caps: $CAPS_MAJOR"

# Supprimer les anciens majors dynamiques
(>=200) et réécrire les bons
sed -i '/lxc.cgroup2.devices.allow: c [2-9][0-9][0-
9]:\* rwm/d' "$LXC_CONF"

cat >> "$LXC_CONF" << CONF
lxc.cgroup2.devices.allow: c $UVM_MAJOR:*
rwm
lxc.cgroup2.devices.allow: c $CAPS_MAJOR:*
rwm
CONF

echo "✓ $LXC_CONF mis à jour"
```

Pour rendre le script exécutable :

```
chmod +x /usr/local/bin/fix-nvidia-
lxc.sh
```

VII.3 Service systemd /etc/systemd/system/nvidia-lxc-fix.service

Ce service s'exécute avant le démarrage du conteneur LXC à chaque boot Proxmox :

```
[Unit]
```

```
Description=Fix NVIDIA cgroup majors pour LXC 104
After=systemd-modules-load.service nvidia-persistenced.service dev-nvidia\x2duvm.device
Wants=dev-nvidia\x2duvm.device
Before=pve-container@104.service

[Service]
Type=oneshot
ExecStart=/usr/local/bin/fix-nvidia-lxc.sh
RemainAfterExit=yes
StandardOutput=journal
StandardError=journal

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

VII.4 Activation du service

```
systemctl daemon-reload
systemctl enable nvidia-lxc-fix.service
systemctl start nvidia-lxc-fix.service
systemctl status nvidia-lxc-fix.service
```

VIII Fix manuel (si besoin urgent)

Si le conteneur est déjà démarré et que le GPU ne fonctionne pas (ex. après une mise à jour driver sans reboot), relancer le service et redémarrer le conteneur suffit :

```
systemctl restart nvidia-lxc-fix.service
pct reboot 104
```

Pour voir les logs du service en temps réel :

```
# Logs du service
journalctl -u nvidia-lxc-fix.service

# Logs en temps réel au boot
journalctl -fu nvidia-lxc-fix.service
```

IX Vérification globale

IX.1 Commandes de vérification

Vérifier les majors dans la conf LXC :

```
grep 'cgroup2.*[2-9][0-9][0-9]'
/etc/pve/lxc/104.conf
```

Redémarrer le conteneur et vérifier le GPU :

```
pct reboot 104 && sleep 10 && pct exec 104 --
nvidia-smi
```

IX.2 Résultat attendu (nvidia-smi)

```
+-----------------------------------------------------
------------+
| NVIDIA-SMI 580.126.18 Driver Version:
580.126.18 CUDA Version: 13.0|
+-----------------------------------------------------
------------+
| GPU Name Persistence-M| Bus-Id Disp.A |
Volatile Uncorr. ECC |
| 0 Quadro P1000 Off | 0MiB / 4096MiB | 0%
Default |
+-----------------------------------------------------
------------+
```

IX.3 Séquence de démarrage complète

Voici l'ordre des événements à chaque boot Proxmox avec le fix automatique en place :

- Boot Proxmox → modules NVIDIA chargés (majors attribués dynamiquement)
- nvidia-lxc-fix.service attend que /dev/nvidia-uvm soit disponible (boucle 30s max)
- Le script lit les vrais majors et met à jour /etc/pve/lxc/104.conf
- pve-container@104.service démarre => cgroups corrects => GPU accessible
- Dans le LXC : nvidia-smi et Docker avec --gpus all fonctionnent normalement

X Notes et points d'attention

- Ce problème de majors dynamiques se reproduit après TOUTE mise à jour du driver NVIDIA sur l'hôte.
- La boucle d'attente de 30s dans le script est essentielle : sans elle, le service démarre avant que les devices soient disponibles.
- Le major 195 (nvidia0/nvidiactl) est stable et n'est pas touché par le script
- L'option --no-kernel-modules est obligatoire lors de l'installation du driver dans le LXC, ne pas l'oublier après une mise à jour.
- La directive no-cgroups=true dans la config nvidia-container-toolkit est spécifique aux LXC : ne pas l'activer sur une VM ou un hôte bare-metal.
- Pour adapter ce guide à un autre LXC, remplacer 104 par l'ID du conteneur cible dans le script et le service systemd.
- En cas de mise à jour majeure de Proxmox (changement de kernel), les modules DKMS seront recompilés automatiquement grâce à l'option –dkms.
- Attention, certaine smise à jour peuvent tout de même casser le service si le module Nvidia n’est pas compilé pour la nouvelle version du kernel