1. Proxmox : utiliser Ollama dans un conteneur LXC avec des GPU Nvidia

Lien : https://shionn.github.io/draft/nvidia-proxmox-lxc-passthrought-ollama.html

1.1. Présentation de Ollama

Ollama permet :

- d'installer un modèle LLM
- de configurer automatiquement tous les paramètres techniques
- dispose d'une interface simple pour discuter avec l'IA

Les données restent locales

Un LLM (Large Language Model) est un programme informatique qui a été entraîné sur d'énormes quantités de texte (livres, articles, code source, pages web...). Grâce à cet entraînement, il a appris :

- Les règles de la langue : grammaire, orthographe, syntaxe
- Les connaissances du monde : histoire, science, actualités (jusqu'à sa date de publication)
- Les patterns de raisonnement : comment résoudre des problèmes, structurer une réponse
- Les conventions du code : syntaxe des langages, bonnes pratiques, patterns courants

Quand une question est posée à un LLM, il ne **cherche** pas la réponse dans une base de données. Il génère la réponse mot par mot, en prédisant quel mot est le plus probable après le précédent, compte tenu de votre question et de tout ce qu'il a appris.

Ollama peut faire fonctionner des LLM uniquement en RAM (512 Gio dans le serveur) mais si des GPU NVIDIA sont installés avec au moins 8 Gio de VRAM, Ollama les utilisera automatiquement. Les deux cartes installées possèdent chacune 16 Gio de VRAM.

La quantité de RAM est le facteur déterminant de la rapidité des réponses du LLM.

Quand un modèle IA (LLM) est lancé avec Ollama, celui-ci est entièrement chargé en mémoire. Si votre ordinateur n'a pas assez de RAM, le modèle ne pourra pas être chargé, ou sera très lent car il devra utiliser le disque dur (**swap**).

RAM disponible	Modèles utilisables	Expérience
4 GB	❌ Insuffisant	Ollama refusera de lancer les modèles
8 GB	Modèles légers (3B)	Fonctionnel mais limité
16 GB	Modèles moyens (7B)	Bonne expérience pour la plupart des usages
32 GB	Grands modèles (13B)	Excellente expérience
64 GB+	Très grands modèles (70B)	Usage professionnel

1.1.1. Espace disque typique occupé par chaque modèle IA :

Modèle	Taille sur disque
Llama 3.2 (3B)	~2 GB
Mistral (7B)	~4 GB
CodeLlama (7B)	~4 GB
Llama 3.1 (70B)	~40 GB

1.1.2. Les paramètre d'un modèle de langage (LLM)

Les modèles de langage (LLM) disponibles sont caractérisés par le nombre de paramètres gérés : **3B** signifie **3 Billion parameters**, soit 3 milliards de paramètres ou nombre de **connexions neuronales** dans le modèle.

Plus un modèle a de paramètres, plus il peut :

- capturer des nuances linguistiques complexes
- mémoriser des patterns
- généraliser à partir d'exemples
- fournir des réponses plus fines

1.1.3. Comparaison rapide

Taille	RAM minimale	Usage typique
1-3B	4-8 GB	Bon équilibre : Questions simples, résumés, traduction

Taille	RAM minimale	Usage typique
7B	8-16 GB	Très bon généraliste : Code, rédaction, raisonnement
13B	16-32 GB	Analyse complexe, créativité
70B	64 GB+	Niveau proche du SOTA (State Of The Art, état de l'art) : Recherche, usage professionnel,
500B+	300 Go et plus de 2 To de RAM, selon la précision.	Modèles géants (GPT-5, etc.) : Raisonnement avancé

Pour un usage personnel, 3B est un excellent compromis.

1.2. Installer Ollama

- utiliser un conteneur avec les pilotes Nvidia
- installer les prérequis

```
apt install -y curl zstd pciutils
```

- installer Ollama

```
# curl -fsSL https://ollama.com/install.sh | sh

>>> Cleaning up old version at /usr/local/lib/ollama
>>> Installing ollama to /usr/local
>>> Downloading ollama-linux-amd64.tar.zst
######################################################################################################################
## 100.0%
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink '/etc/systemd/system/default.target.wants/ollama.service' ->
'/etc/systemd/system/ollama.service'.
>>> NVIDIA GPU installed.
```

1.3. Vérifier l'installation

* visualiser la version

```
ollama --version
```

⇒ vous devriez visualiser le numéro de version comme ollama version 0.14.1

- Vérifiez que le service tourne

```
systemctl status ollama
```

⇒ Vous devriez voir Active: active (running) en vert.

1.4. Tester le modèle en console

```
# ollama run qwen2.5-coder:7b
```

1.4.1. Poser des questions en langage naturel

Il suffit simplement de taper une question comme :

- Comment créer une API REST en Python ?

Le modèle répond dans le terminal.

Générer du code (tous langages) Comme il s'agit d'un modèle coder, on peux lui demander :

- Écris une fonction en JavaScript qui trie une liste d'objets par date.

Ou même des projets complets :

- Génère un Dockerfile pour une application FastAPI.

Expliquer du code :

- Explique ce que fait ce script :

<ton code ici>

Il te donnera une explication détaillée.

Déboguer ou améliorer du code

- Voici mon code, il plante. Trouve l'erreur.

ou

- Optimise cette fonction pour la rendre plus rapide.

Travailler en conversation continue

- Ollama garde l'état de la conversation tant que le processus est lancé. Il est alors possible d'enchaîner les prompts :
- Maintenant rends le code compatible Python 3.12.

Quitter proprement

- /bye

ou simplement CTRL + C.

1.4.2. Utiliser le modèle dans un script (API locale Ollama)

- depuis une autre console, pour appeler l'API :

```
curl http://localhost:11434/api/generate \
  -d '{ "model": "qwen2.5-coder:7b", "prompt": "Écris une classe Python." }'
```

1.4.3. Utiliser dans VS Code ou un éditeur

Beaucoup d'extensions permettent de configurer Ollama comme LLM local. **Qwen2.5-Coder** peux alors être utilisé comme assistant de code directement dans l'IDE.

1.4.4. Lancer en mode serveur

```
ollama serve
```

Le modèle devient accessible à d'autres outils (LM Studio, Continue, Cursor, etc.).

1.5. Installer Open WebUI

1.5.1. Installer Nodejs

```
apt install -y nodejs npm
```

1.5.2. Installer les prérequis

```
apt install python3-pip python3-venv  => à enlever
```

1.5.3. Créer un environnement virtuel pour OpenWebUI

1.5.4. Installation de pyenv

- Installer pyenv

```
curl https://pyenv.run | bash
```

- Ajouter les lignes suivantes à la fin du fichier **.bashrc** du compte root :

```
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

- Recharger le shell :

```
exec $SHELL
```

1.5.5. Installer Python 3.11

```
apt install -y \
  build-essential \
  libssl-dev \
  zlib1g-dev \
  libbz2-dev \
  libreadline-dev \
  libsqlite3-dev \
  libffi-dev \
  liblzma-dev \
  uuid-dev \
  tk-dev \
  libncursesw5-dev \
  libxml2-dev \
  libxmlsec1-dev \
  libgdbm-dev \
  libnss3-dev \
  libdb-dev \
  xz-utils
wget https://www.python.org/ftp/python/3.11.7/Python-3.11.7.tgz
tar -xzf Python-3.11.7.tgz
cd Python-3.11.7

./configure --enable-optimizations

make -j$(nproc)

make altinstall
```

- Vérifier que SQLite fonctionne

```
python3.11 -c "import sqlite3; print(sqlite3.sqlite_version)"
```

```
=> doit afficher 3.46.1
```

- Test d'utilisation ????

```
cd ~
pyenv global 3.11.7
python3 --version   # Python 3.11.7Afficher plus de lignes
```

1.5.6. Exécuter open-webui

- Créer l'environnement virtuel avec la nouvelle version Python dans le dossier **/opt/open-webui** :

```
mkdir /opt/open-webui
cd /opt/open-webui
python3.11 -m venv venv
source venv/bin/activate
```

```
pip install --upgrade pip
```

- Installer Open-WebUI :

```
pip install open-webui
```

- lancer Open-Webui

```
open-webui serve
```

- Accéder à Open-Webui depuis un navigateur à l'URL http://adresseopenwebui:8080

1.6. Lancer automatiquement Open WebUI au démarrage du conteneur

1.6.1. Créer un compte dédié à open-webui

```
useradd -r -s /usr/sbin/nologin -d /opt/open-webui openwebui
```

Commentaires :

- -r → compte système
- -s /usr/sbin/nologin → impossible de se connecter
- -d /opt/open-webui → son dossier d'application
- openwebui → nom du compte

- Donner les droits sur le dossier Open WebUI :

```
chown -R openwebui:openwebui /opt/open-webui
```

 2. Pourquoi tu n'as p

1.6.2. Créer un service systemd

- créer le fichier **/etc/systemd/system/openwebui.service** avec le contenu suivant

```
[Unit]
Description=Open WebUI service
After=network.target

[Service]
Type=simple
User=openwebui
Group=openwebui
WorkingDirectory=/opt/openwebui
ExecStart=/opt/open-webui/venv/bin/open-webui serve --host 0.0.0.0 --port 8080
Restart=always
Environment="PATH=/opt/open-webui/venv/bin:/usr/local/bin:/usr/bin"
Environment="PYTHONUNBUFFERED=1"

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

Problème	Solution
Le service démarre trop tôt	After=network-online.target + ExecStartPre sleep 5
venv pas disponible au boot	sleep 5
réseau absent au boot	network-online.target

* Activer et démarrer le service

```
systemctl daemon-reload
systemctl enable openwebui
systemctl start openwebui
systemctl status openwebui
```

1.6.3. Publier openwebui sur le port 80 avec caddy

- Installer Caddy

```
apt install -y caddy
```

- Configuration :

```
nano /etc/caddy/Caddyfile
```

- Décommenter la ligne pour avoir ce contenu

```
:80 {
    reverse_proxy 127.0.0.1:8080
}
```

- Recharger la configuration :

```
systemctl reload caddy
```

Open WebUI est accessible à l'URL http://adresseipLXC/

From:
https://siocours.lycees.nouvelle-aquitaine.pro/ - **Les cours du BTS SIO**

Permanent link:
https://siocours.lycees.nouvelle-aquitaine.pro/doku.php/reseau/cloud/proxmox/lxcnvidiaollama?rev=1768647776

Last update: **2026/01/17 11:02**