

Conversación con Gemini

Exported: 15/6/2026 10:05:56

Jordi H.

Has dicho tengo una GPU L40s. quiero correr simultáneamente varias instancias de Ollama con distintos LLM. Cómo hago para que no se bloqueen entre sí y tengamos un escenario tipo tiempo compartid

Gemini

Para aprovechar una **L40s (48GB de VRAM)** con un escenario de "tiempo compartido" o concurrencia real, no necesitas lanzar varios procesos de Ollama por separado (lo cual suele ser ineficiente con la gestión de memoria). Desde la versión 0.2, Ollama ya soporta la carga de múltiples modelos y el procesamiento de peticiones en paralelo de forma nativa. La clave está en configurar correctamente las **variables de entorno** del servi

1. Configuración de Concurrencia Nativa

Para que Ollama gestione los recursos como un sistema multi-usuario/multi-instancia, debes definir estas tres variables antes de iniciar el servicio

- **OLLAMA_MAX_LOADED_MODELS**: Define cuántos modelos distintos pueden residir en la VRAM simultáneamente. Con 48GB, puedes poner un valor alto (ej. 4 o 6) dependiendo del tamaño de los modelos (si usas Llama 3 8B, caben muchos; si usas modelos de 70B, quizás solo uno o dos)
- **OLLAMA_NUM_PARALLEL**: Define cuántas peticiones simultáneas puede procesar cada modelo. Si pones 4, un mismo modelo podrá responder a 4 usuarios a la vez repartiendo el poder de cómputo de la L40s
- **OLLAMA_MAX_QUEUE**: El número de peticiones que Ollama pondrá en espera si el hardware está saturado antes de rechazar conexiones

Ejemplo de ejecución (Linux):

Bash

CODE

```
export OLLAMA_MAX_LOADED_MODELS=4
export OLLAMA_NUM_PARALLEL=4
ollama serve
```

2. Estrategia de "Tiempo Compartido" con Docker

Si prefieres un aislamiento total (instancias con puertos distintos para diferentes equipos o departamentos), la mejor forma es usar **Docker**. La L40s permite que varios contenedores accedan a ella, pero debes tener cuidado de que no intenten reservar toda la memoria a la vez. Puedes lanzar contenedores limitando su visibilidad, aunque en una sola GPU lo ideal es que compartan el recurso:

CODE

```
# Instancia para el equipo A (Puerto 11434)
docker run -d --gpus all \
-e OLLAMA_NUM_PARALLEL=2 \
-p 11434:11434 \
--name ollama_equipo_a ollama/ollama

# Instancia para el equipo B (Puerto 11435)
docker run -d --gpus all \
-p 11435:11434 \
--name ollama_equipo_b ollama/ollama
```

3. Consideraciones Técnicas para la L40s

La NVIDIA L40s es una bestia para inferencia, pero ten en cuenta estos puntos para evitar bloqueos:

- Cálculo de VRAM: Cada instancia de OLLAMA_NUM_PARALLEL multiplica el tamaño del KV Cache (el contexto). Si un modelo usa 5GB y el contexto de 8k usa 2GB, con parallel=4 estarás ocupando $5 + (2 \times 4) = 13$ GB aproximadamente
- KV Cache Quantization: Para maximizar el tiempo compartido, usa modelos que soporten cuantización de caché (como K/V de 4 o 8 bits), lo que permite meter más usuarios en los mismos 48GB
- Prioridad: Ollama no tiene un "planificador de prioridades" avanzado. Si una instancia satura los núcleos CUDA, la otra irá más lenta. Es un "tiempo compartido" por competencia de recursos

Nota: Si instalaste Ollama como servicio de systemd en Linux, debes editar el archivo de configuración en `/etc/systemd/system/ollama.service` y añadir las variables en la sección [Service] como

`Environment="OLLAMA_NUM_PARALLEL=4"`. ¿Te gustaría que te ayude a calcular cuántas instancias de un modelo específico (como Llama 3.3 70B o Mistral) podrías meter en esos 48GB de tu L4

Jordi H.

Has dicho para la estrategia con docker, debo instalar software de nvidia para hacer posible la compartición?

Gemini

Sí, para que Docker pueda "ver" y utilizar tu **L40s**, necesitas instalar obligatoriamente el **NVIDIA Container** to, los contenedores de Ollama no tendrán acceso a los núcleos CUDA y correrán (muy lentamente) en la CPU. Aquí tienes la hoja de ruta simplificada para dejarlo listo en una L40s sobre Linux (Ubuntu/Debian):

1. Requisitos previos

Debes tener instalados los **Drivers de NVIDIA** (recomendado versión 535 o superior para la L40s) y **Docker**. Puedes verificar los drivers con: Bas

CODE

```
nvidia-smi
```

2. Instalar el NVIDIA Container Toolkit

Este software es el puente entre Docker y tu GPU. Ejecuta estos comandos para configurar el repositorio e instalarlo: Bas

CODE

```
# Configurar el repositorio
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o
/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list
| \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

# Instalar el toolkit
sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
```

3. Configurar Docker para usar la GPU

Una vez instalado, debes decirle a Docker que use el "runtime" de NVIDIA:Bash

CODE

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

El secreto del "Tiempo Compartido": NVIDIA MPS

Si vas a correr **muchas instancias pequeñas** simultáneamente y quieres que compartan la GPU de forma eficiente (sin que una bloquee a las demás mientras espera a la CPU), te recomiendo activar **NVIDIA MP** ocess Service). **La L40s maneja muy bien el particionamiento de recursos. Sin MPS, el acceso a la GPU es p** ismo milisegundo, **aprovechando mejor los núcleos CUDA que queden libres.**Cómo activar MPS en el host:** Bash

CODE

```
# Iniciar el demonio de control de MPS
export CUDA_VISIBLE_DEVICES=0 # Ajusta el ID si tienes más GPUs
nvidia-cuda-mps-control -d
```

Resumen de la Arquitectura

Componente	Función
NVIDIA Driver	Comunicación básica con el hardware L40s.
NVIDIA Container Toolkit	Permite que Docker "mapee" la GPU dentro de los
Ollama Docker Image	Contiene el binario de Ollama y las librerías de inferencia.
MPS (Opcional)	Optimiza la concurrencia real entre múltiples

¿Cómo lanzar las instancias ahora?

Con el toolkit instalado, ya puedes usar el flag --gpus all en tus comandos de Docker:Bash

CODE

```
docker run -d --gpus all -v ollama:/root/.ollama -p 11434:11434 --name ollama-1 ollama/ollama
```

¿Te gustaría que te prepare un archivo docker-compose.yml para gestionar 3 o 4 instancias de Ollama con

Jordi H.

Has dicho debo pagar licencia por el mps?

Gemini

No, **no tienes que pagar licencia por usar MPS** (Multi-Process Service). Es una funcionalidad que viene incluida de forma gratuita en el stack de software de NVIDIA (CUDA Toolkit). Aquí hay que hacer una distinción importante para tu **L40**

1. MPS vs. vGPU (La diferencia clave)

- **MPS (Gratis):** Es una solución a nivel de software/driver. Permite que varios procesos de Linux (o contenedores Docker) envíen comandos a la GPU simultáneamente. Es ideal para "llenar los huecos" de la GPU cuando un solo modelo no la usa al 100%. Esto es lo que tú necesita
- **vGPU / NVIDIA AI Enterprise (De pago):** Esto se usa si quieres crear "máquinas virtuales" con porciones rígidas de la GPU (por ejemplo, en VMware o Hyper-V) con aislamiento total de hardware. Eso sí requiere licencias costosas, pero para correr Ollama en Docker sobre Linux, no es necesario

2. Cómo activar MPS para tus contenedores

Para que tus instancias de Ollama compartan la L40s de forma eficiente, sigue estos pasos en el servidor (fuera de Docker):

CODE

```
# 1. Asegúrate de que no haya procesos usando la GPU
nvidia-smi

# 2. Activa el modo de computación exclusiva (requerido para MPS)
sudo nvidia-smi -i 0 -c EXCLUSIVE_PROCESS

# 3. Inicia el demonio de control de MPS
export CUDA_VISIBLE_DEVICES=0
nvidia-cuda-mps-control -d
```

3. Ejemplo de Docker Compose (Tiempo Compartido)

Aquí tienes un archivo docker-compose.yml para levantar dos instancias de Ollama que convivan en la misma L40s sin pisarse, listas para tu escenario de tiempo compartido:

CODE

```
services:
ollama-general:
image: ollama/ollama
container_name: ollama_equipo_1
ports:
- "11434:11434"
volumes:
- ./ollama_data_1:/root/.ollama
deploy:
resources:
reservations:
devices:
- driver: nvidia
count: 1
capabilities: [gpu]
environment:
- OLLAMA_NUM_PARALLEL=2 # Permite 2 usuarios a la vez en esta instancia

ollama-dev:
image: ollama/ollama
container_name: ollama_equipo_2
ports:
- "11435:11434"
volumes:
- ./ollama_data_2:/root/.ollama
deploy:
resources:
reservations:
devices:
- driver: nvidia
count: 1
capabilities: [gpu]
environment:
- OLLAMA_NUM_PARALLEL=2
```

Un consejo de "amigo":

La **L40s** tiene arquitectura Ada Lovelace y **48GB** de VRAM, pero **no soporta MIG** (Multi-Instance GPU), que es la tecnología de las A100/H100 para partir la tarjeta físicamente. Por eso, **MPS es tu mejor aliado**: es la forma más profesional y gratuita de lograr que la L40s no se quede esperando a un solo proceso y pueda atender a varios a la vez. ¿Te gustaría que te ayude a configurar el límite de memoria para que una instancia no "secuestre" todos los 48GB y deje a la otra sin n

Jordi H.

Has dicho si en el próximo futuro adquiero una a100, deberé pagar por el software de compartición MIG?

Gemini

La respuesta corta es: **No, la tecnología MIG en sí no es de pago**, pero su uso en entornos profesionales suele requerir una licencia si quieres soporte oficial o virtualización avanzada. Aquí tienes el desglose para que no te lleves sorpresas con el presupuesto

1. El Hardware y el Driver (Gratis)

Si compras una **A100** (ya sea PCIe o SXM), la capacidad de hacer **MIG (Multi-Instance GPU)** viene "grabada"

en el silicio

- Puedes activar MIG usando el comando `nvidia-smi -mig 1`.
- Puedes crear hasta 7 particiones independientes (cada una con su propia memoria y computación aislada).
- Para correr Docker directamente sobre Linux (bare metal) usando MIG, no necesitas pagar licencias adicionales. Es una función del hardware y del driver gratuito

2. NVIDIA AI Enterprise (Suscripción / De pago)

NVIDIA ha movido gran parte de su ecosistema de centros de datos bajo el paraguas de **NVIDIA AI Enterprise**. Necesitarás pagar si

- Quieres soporte técnico oficial: Para resolver problemas críticos en producción.
- Usas Máquinas Virtuales (vGPU): Si en lugar de Docker usas un Hipervisor (como VMware vSphere o Proxmox) y quieres que cada Máquina Virtual vea una partición MIG como si fuera una GPU física, ahí es donde entra la licencia de NVIDIA vGPU / AI Enterprise

Comparativa: Tu L40s actual vs. A100 con MIG

Es fundamental entender que **MIG es mucho más rígido que MPS** (lo que usas en la L40s).

Característica	L40s (con MPS)	A100 (con MIG)
Aislamiento	Soft (lógico). Si un proceso falla	Hard (físico). Aislamiento total de
Flexibilidad	Alta. Puedes asignar el 10%, 50% o	Baja. Tienes que elegir "perfiles" fijos
Latencia	Puede variar según la carga de otros	Determinista. El rendimiento es
Costo Soft	\$0	\$0 (Docker) / \$\$\$

Mi recomendación para Ollama

Si el objetivo es correr **Ollama para varios usuarios**, la L40s que ya tienes es en muchos sentidos **mejor** que una A100 debido a su arquitectura más moderna (Ada Lovelace) y su mayor velocidad de reloj para inferencia. La A100 con MIG es superior solo si necesitas **garantías de seguridad (SLA)** donde un usuario no pueda bajo ninguna circunstancia afectar el rendimiento de otro (por ejemplo, si vendes acceso a la GPU a terceros). **¿Te gustaría que te ayude a configurar un perfil MIG específico en caso de que ya estés evaluand**
