Inserm

La science pour la santé
From science to health

Guide d'utilisation

Mise en œuvre de vLLM avec 2 GPUs H100 sur le cluster HPC GPU

INSERM - Cloud self-service - Offre HPC

Objectifs du document

Ce tutoriel vise à vous guider pas-à-pas pour :

1	2
Configurer un environnement Python isolé sur le cluster HPC	**Installer des librairies au travers d'un serveur Nexus faisant office de proxy**
3	**4**
Exécuter vLLM en tirant parti de 2 GPUs H100	**Mettre en oeuvre les calculs depuis une console Linux, une interface graphique (GUI Linux) ou JupyterHub**

Table des matières

1 Eléments contextuels

2 Création d'un environnement virtuel Python

3 Mise en œuvre des calculs via une console Linus

4 Mise en œuvre des calculs via une GUI Linux

5 Mise en œuvre des calculs via jupyterhub

1 Eléments contextuels

Mise en œuvre de vLLM avec 2 GPUs H100 sur le cluster HPC GPU

CONTEXTE

Les LLM (Large Language Models) sont des **modèles d'intelligence artificielle très volumineux qui nécessitent beaucoup de puissance de calcul et de mémoire**, en particulier lors de l'inférence ou de l'adaptation de modèles. Le HPC permet d'**exploiter des GPU performants et de répartir les calculs sur plusieurs ressources**, ce qui réduit fortement les temps d'exécution, notamment pour des usages d'intelligence artificielle et de traitement de données scientifiques.

Parmi les possibilités de mise en œuvre de LLM, 2 solutions peuvent être mises en oeuvre : Ollama et vLLM.

Dans ce tutoriel, nous aborderons la solution vLLM.

EXEMPLES DE CAS D'USAGE VISÉS

- Tests et validation de modèles LLM
- Expérimentation GPU multi-cartes
- Prototypage et développement de workflows IA
- Évaluation de performances avant déploiement applicatif
- Utilisation interactive (notebooks) ou batch (scripts)

Ressources de calcul

La plateforme HPC propose divers types de ressources GPU. Pour les cas d'usage VLLM, nous utiliserons la ressource ci-dessous :

Partition 2xH100-96

Composée de 2 nœuds de calculs fournissant chacun :

- 32 Cores CPU
- 2 GPUs H100 – 96Go

Dédiée en intégralité à l'utilisateur qui a réservé le nœud.

L'accès et la réservation de nœuds de calcul d'une partition se fait au travers de l'orchestrateur Slurm et de ses commandes.

2 Création d'un environnement virtuel Python

Contraintes de développement Python

Python 3.9
Version par défaut sur les nœuds de calcul HPC

Python 3.10 & 3.11
Disponibles au travers d'environnement virtuel python

L'environnement HPC étant restreint, les utilisateurs peuvent télécharger et installer des librairies au travers d'un serveur Nexus faisant office de proxy. L'utilisation de ce serveur est conditionnée par un mécanisme d'authentification.

Bonnes pratiques de développement Python

Depuis home directory, nous allons créer un environnement virtuel. Nous pouvons créer autant d'environnement que nécessaire (en principe un par projet).

Un environnement virtuel Python sert à isoler les dépendances d'un projet Python. Il crée un espace indépendant avec :

- Une version dédiée de Python
- Un dossier distinct pour installer des bibliothèques spécifiques au projet

Cela permet de :

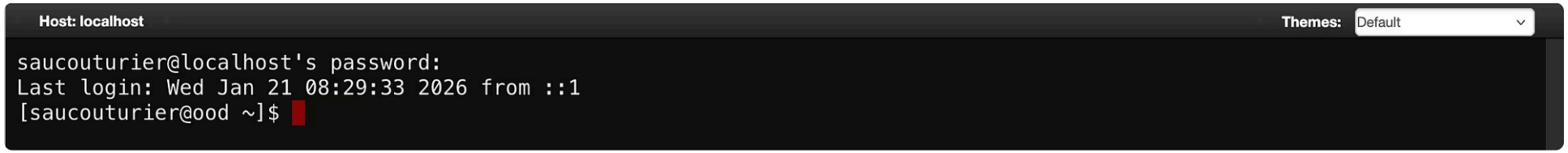
Éviter les conflits de dépendances

Ne pas polluer le système global

Gérer proprement les dépendances

Tester facilement sur différentes versions de bibliothèque

1) Cliquer sur le menu Clusters > HPC Shell Access pour accéder au home directory depuis la plateforme OOD HPC

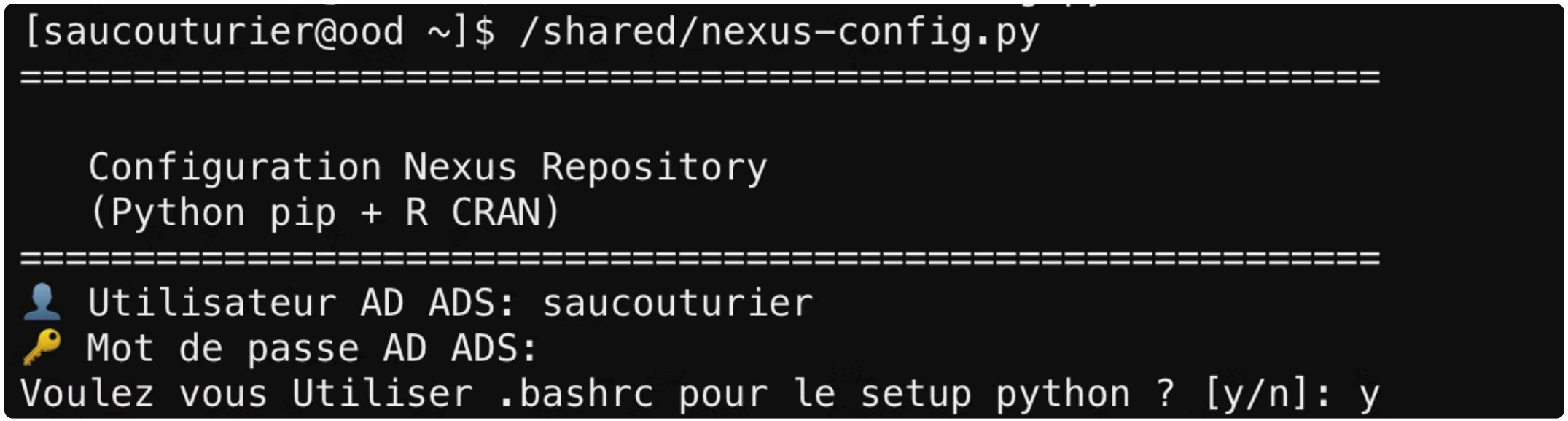


2) Exécuter la commande ci-dessous depuis la console pour configurer l'accès à un serveur Nexus afin de faciliter l'installation de bibliothèques Python et R :

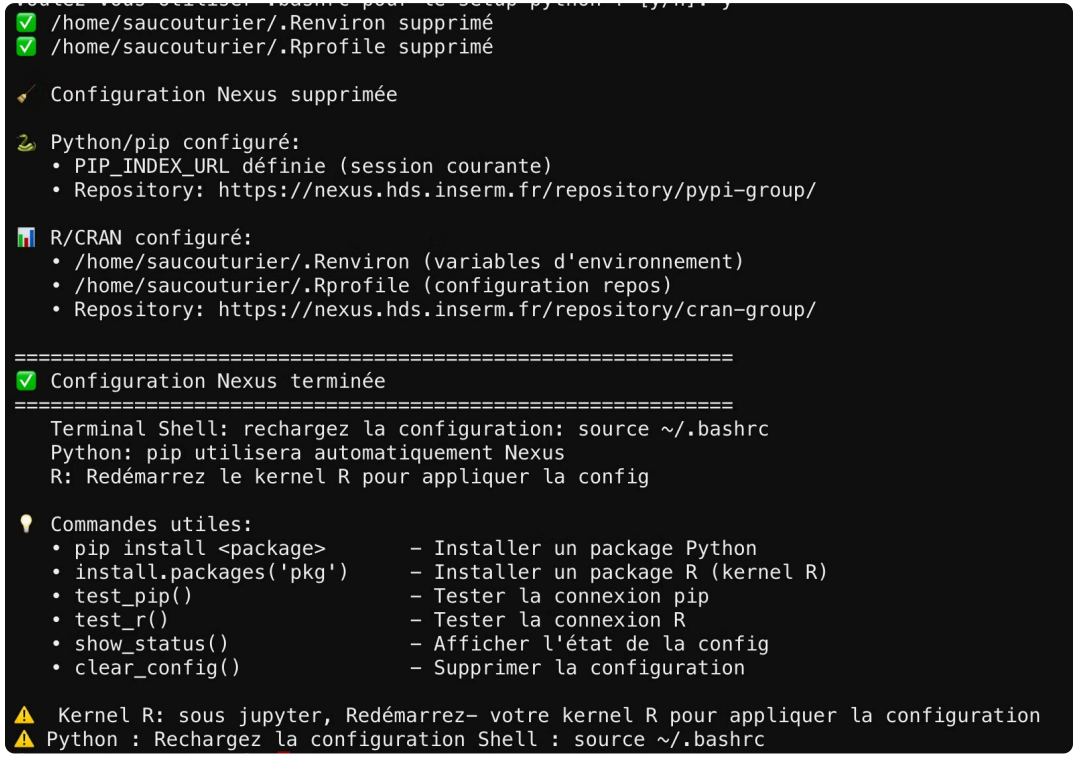
```
$>/shared/nexus-config.py
```

a. Saisir le nom utilisateur de votre compte ADS et son mot de passe associé

b. Taper y à la question "voulez-vous utiliser .bashrc pour le setup python ?"



La configuration se déroule :



c. Recharger la configuration Shell pour utiliser le nouveau paramétrage :

Dans le cadre de ce document, nous allons créer un environnement virtuel que nous allons appeler « vllm » et utilisant python 3.11. Cet environnement permettra d'installer les bibliothèques python d'un projet « VLLM ».

3) Exécuter la commande ci-dessous depuis la console :

```
$>/shared/Packages/python/3.11/bin/python3.11 -m venv /home/$USER/vllm
```

- /shared/Packages/python/3.11/bin/python3.11 : indique que nous allons utiliser la version 3.11 de python mise à disposition par le HPC via le répertoire /shared/packages/python/3.11
- vllm : nom de l'environnement virtuel python
- /home/$USER : préciser le chemin de votre home directory afin que notre venv puisse être utilisé dans le futur avec jupyterhub

4) Exécuter la commande ci-dessous pour activer le venv et vérifier la version de python3 active :

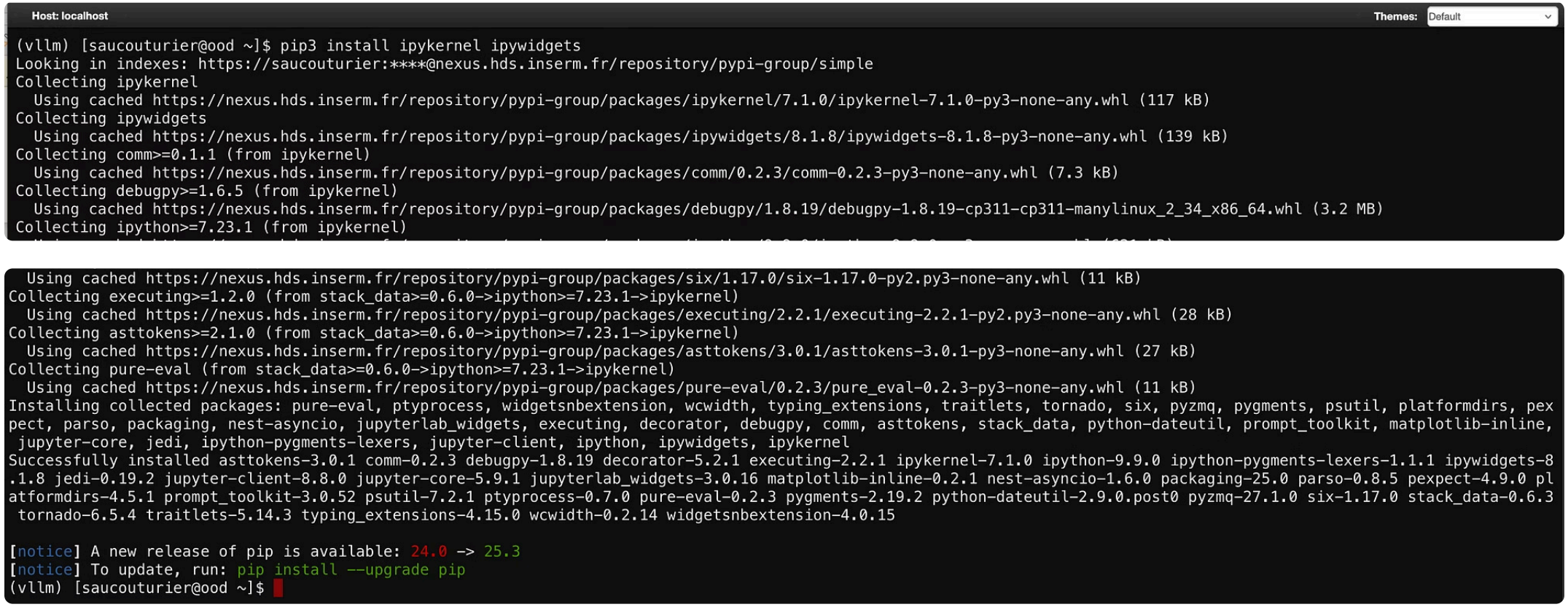
```
$> source vllm/bin/activate
$> python3 --version
```

A noter : la version python 3.11 versus 3.9 sans venv activé.

5) Exécuter la commande ci-dessous pour installer les librairies nécessaires à jupyterhub :

Afin de pouvoir facilement utiliser notre venv depuis jupyterhub, nous devons ajouter les librairies nécessaires : ipykernel ipywidgets

```
$> pip3 install ipykernel ipywidgets
```



6) Exécuter la commande ci-dessous pour rendre le venv visible depuis la GUI jupyterhub :

```
$> python3 -m ipykernel install --user --name=test-vllm --display-name "vllm"
```

```
(vllm) [saucouturier@ood ~]$ python3 -m ipykernel install --user --name=test-vllm --display-name "vllm"
Installed kernelspec test-vllm in /nfs/inserm/home_users/saucouturier/.local/share/jupyter/kernels/test-vllm
(vllm) [saucouturier@ood ~]$
```

7) Exécuter la commande ci-dessous pour installer les librairies nécessaires au projet « VLLM » :

Nous allons installer la libraririe vllm dans sa version 0.8.5.post1 avec toutes ses dépendances. L'installation peut prendre plusieurs minutes.

```
$> pip3 install vllm==0.8.5.post1
```

```
(vllm) [saucouturier@ood ~]$ pip3 install vllm==0.8.5.post1
Looking in indexes: https://saucouturier:****@nexus.hds.inserm.fr/repository/pypi-group/simple
Collecting vllm==0.8.5.post1
  Downloading https://nexus.hds.inserm.fr/repository/pypi-group/packages/vllm/0.8.5.post1/vllm-0.8.5.post1-cp38-abi3-manylinux1_x86_64.whl (326.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 326.4/326.4 MB 12.2 MB/s eta 0:00:00
Collecting cachetools (from vllm==0.8.5.post1)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/cachetools/6.2.4/cachetools-6.2.4-py3-none-any.whl (11 kB)
Requirement already satisfied: psutil in /home/saucouturier/vllm/lib/python3.11/site-packages (from vllm==0.8.5.post1) (7.2.1)
Collecting sentencepiece (from vllm==0.8.5.post1)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/sentencepiece/0.2.1/sentencepiece-0.2.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64
.whl (1.4 MB)
Collecting numpy (from vllm==0.8.5.post1)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/numpy/2.4.1/numpy-2.4.1-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.7 MB)
Collecting requests>=2.26.0 (from vllm==0.8.5.post1)
  Using cached https://nexus.hds.inserm.fr/repository/pypi-group/packages/requests/2.32.5/requests-2.32.5-py3-none-any.whl (64 kB)
Collecting tqdm (from vllm==0.8.5.post1)
```

```
Installing collected packages: triton, py-cpuinfo, nvidia-cusparselt-cu12, mpmath, fastrlock, zipp, wrapt, websockets, uvloop, urllib3, typing-inspection, tqdm, sympy, s
niffio, shellingham, sentencepiece, safetensors, rpds-py, rignore, regex, pyyaml, python-multipart, python-json-logger, python-dotenv, pydantic-core, pycountry, protobuf
, propcache, prometheus_client, pillow, partial-json-parser, opentelemetry-semantic-conventions-ai, nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-cur
and-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, numpy, ninja, networkx, multidict, msgspec, ms
gpack, mdurl, MarkupSafe, llvmlite, llguidance, lark, jiter, interegular, idna, httptools, hf-xet, h11, grpcio, fsspec, frozenlist, filelock, fastar, einops, dnspython,
distro, diskcache, dill, cloudpickle, click, charset_normalizer, certifi, cachetools, blake3, attrs, astor, annotated-types, annotated-doc, airportsdata, aiohappyeyeball
s, yarl, uvicorn, sentry-sdk, scipy, requests, referencing, pydantic, opentelemetry-proto, opencv-python-headless, nvidia-cusparse-cu12, nvidia-cudnn-cu12, numba, markdo
wn-it-py, jinja2, importlib_metadata, httpcore, googleapis-common-protos, gguf, email-validator, depyf, deprecated, cupy-cuda12x, anyio, aiosignal, watchfiles, tiktoken,
 starlette, rich, pydantic-settings, pydantic-extra-types, opentelemetry-exporter-otlp-proto-common, opentelemetry-api, nvidia-cusolver-cu12, lm-format-enforcer, jsonsch
ema-specifications, huggingface-hub, httpx, aiohttp, typer, torch, tokenizers, rich-toolkit, prometheus-fastapi-instrumentator, opentelemetry-semantic-conventions, opena
i, jsonschema, fastapi, xformers, transformers, torchvision, torchaudio, ray, outlines_core, opentelemetry-sdk, mistral_common, fastapi-cloud-cli, fastapi-cli, xgrammar,
 outlines, opentelemetry-exporter-otlp-proto-http, opentelemetry-exporter-otlp-proto-grpc, compressed-tensors, opentelemetry-exporter-otlp, vllm
Successfully installed MarkupSafe-3.0.3 aiohappyeyeballs-2.6.1 aiohttp-3.13.3 aiosignal-1.4.0 airportsdata-20250909 annotated-doc-0.0.4 annotated-types-0.7.0 anyio-4.12.
1 astor-0.8.1 attrs-25.4.0 blake3-1.0.8 cachetools-6.2.4 certifi-2026.1.4 charset_normalizer-3.4.4 click-8.3.1 cloudpickle-3.1.2 compressed-tensors-0.9.3 cupy-cuda12x-13
.6.0 deprecated-1.3.1 depyf-0.18.0 dill-0.4.1 diskcache-5.6.3 distro-1.9.0 dnspython-2.8.0 einops-0.8.1 email-validator-2.3.0 fastapi-0.128.0 fastapi-cli-0.0.20 fastapi-
cloud-cli-0.11.0 fastar-0.8.0 fastrlock-0.8.3 filelock-3.20.3 frozenlist-1.8.0 fsspec-2026.1.0 gguf-0.17.1 googleapis-common-protos-1.72.0 grpcio-1.76.0 h11-0.16.0 hf-xe
t-1.2.0 httpcore-1.0.9 httptools-0.7.1 httpx-0.28.1 huggingface-hub-0.36.0 idna-3.11 importlib_metadata-8.0.0 interegular-0.3.3 jinja2-3.1.6 jiter-0.12.0 jsonschema-4.26
.0 jsonschema-specifications-2025.9.1 lark-1.2.2 llguidance-0.7.30 llvmlite-0.44.0 lm-format-enforcer-0.10.12 markdown-it-py-4.0.0 mdurl-0.1.2 mistral_common-1.8.8 mpmat
h-1.3.0 msgpack-1.1.2 msgspec-0.20.0 multidict-6.7.0 networkx-3.6.1 ninja-1.13.0 numba-0.61.2 numpy-2.2.6 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvi
dia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-1
1.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-cusparselt-cu12-0.6.2 nvidia-nccl-cu12-2.21.5 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.4.127 openai-2.15.0 openc
v-python-headless-4.13.0.90 opentelemetry-api-1.26.0 opentelemetry-exporter-otlp-1.26.0 opentelemetry-exporter-otlp-proto-common-1.26.0 opentelemetry-exporter-otlp-proto
-grpc-1.26.0 opentelemetry-exporter-otlp-proto-http-1.26.0 opentelemetry-proto-1.26.0 opentelemetry-sdk-1.26.0 opentelemetry-semantic-conventions-0.47b0 opentelemetry-se
mantic-conventions-ai-0.4.13 outlines-0.1.11 outlines_core-0.1.26 partial-json-parser-0.2.1.1.post7 pillow-12.1.0 prometheus-fastapi-instrumentator-7.1.0 prometheus_clie
nt-0.24.1 propcache-0.4.1 protobuf-4.25.8 py-cpuinfo-9.0.0 pycountry-24.6.1 pydantic-2.12.5 pydantic-core-2.41.5 pydantic-extra-types-2.11.0 pydantic-settings-2.12.0 pyt
hon-dotenv-1.2.1 python-json-logger-4.0.0 python-multipart-0.0.21 pyyaml-6.0.3 ray-2.53.0 referencing-0.37.0 regex-2026.1.15 requests-2.32.5 rich-14.2.0 rich-toolkit-0.1
7.1 rignore-0.7.6 rpds-py-0.30.0 safetensors-0.7.0 scipy-1.17.0 sentencepiece-0.2.1 sentry-sdk-2.50.0 shellingham-1.5.4 sniffio-1.3.1 starlette-0.50.0 sympy-1.13.1 tikto
ken-0.12.0 tokenizers-0.22.2 torch-2.6.0 torchaudio-2.6.0 torchvision-0.21.0 tqdm-4.67.1 transformers-4.57.6 triton-3.2.0 typer-0.21.1 typing-inspection-0.4.2 urllib3-2.
6.3 uvicorn-0.40.0 uvloop-0.22.1 vllm-0.8.5.post1 watchfiles-1.1.1 websockets-16.0 wrapt-2.0.1 xformers-0.0.29.post2 xgrammar-0.1.18 yarl-1.22.0 zipp-3.23.0

[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: pip install --upgrade pip
```

La préparation du venv est terminée.

8) Exécuter la commande ci-dessous pour désactiver le venv temporairement :

```
$> deactivate
```

```
(vllm) [hpc_admin@ood ~]$ deactivate
[hpc_admin@ood ~]$
```

A partir de ce point, la ré-activation du venv « vllm » nous permettra d'utiliser la librairie vllm dans notre code python.

Utilisation du code python depuis les noeuds de calculs GPU

Dans les prochains chapitres, nous allons voir comment utiliser notre code python depuis les nœuds de calculs GPU.

Plusieurs options sont possibles, selon vos habitudes :

Mise en oeuvre des calculs via une console Linux

Détaillée au chapitre 3

Mise en oeuvre des calculs via une GUI Linux

Détaillée au chapitre 4

Mise en oeuvre des calculs via jupyterhub

Détaillée au chapitre 5

A noter : Jupyterhub offre plus de fonctionnalités pour le debugage et test du code

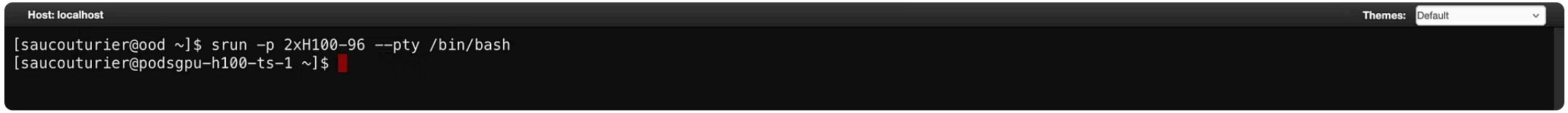
3 Mise en œuvre des calculs via une console Linux

1) Cliquer sur le menu Clusters > HPC Shell Access pour accéder au home directory depuis la plateforme OOD HPC

2) Exécuter la commande ci-dessous pour réserver un nœud de calcul de la partition 2xH100-96 :

```
$> srun -p 2xH100-96 --pty /bin/bash
```

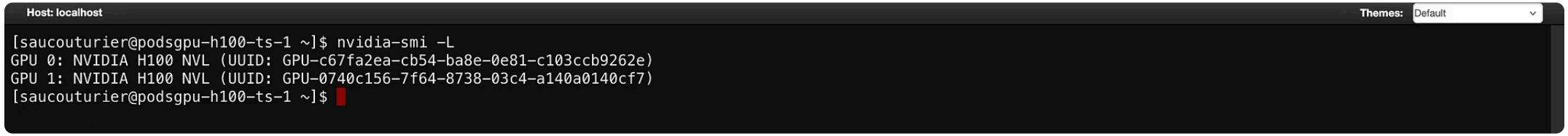
Le prompt prend alors le nom du nœud de calcul, nous indiquant que nous sommes relocalisé sur le nœud de calcul :



3) Exécuter la commande ci-dessous pour vérifier la présence des gpus :

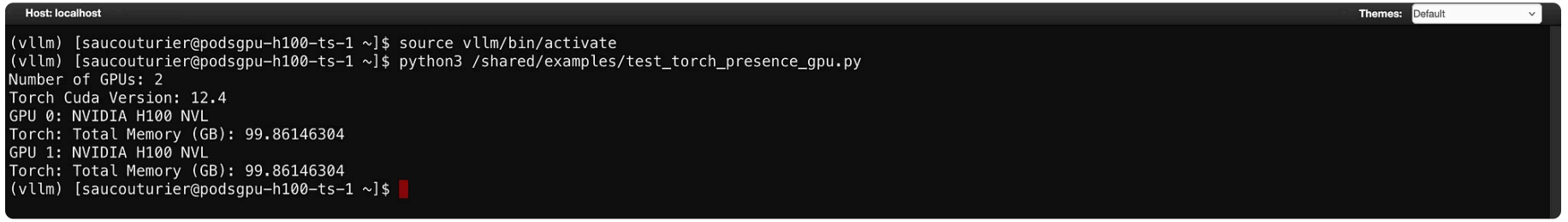
Nous pouvons vérifier la présence des 2 GPUs car nous avons réservé un nœud de la partition 2xH100-96 :

```
$> nvidia-smi -L
```



4) Exécuter la commande ci-dessous pour activer le venv et exécuter un test de présence gpu via python :

```
$> source vllm/bin/activate
$> python3 /shared/examples/test_torch_presence_gpu.py
```



5) Exécuter la commande « vllm » sur 2 GPUs :

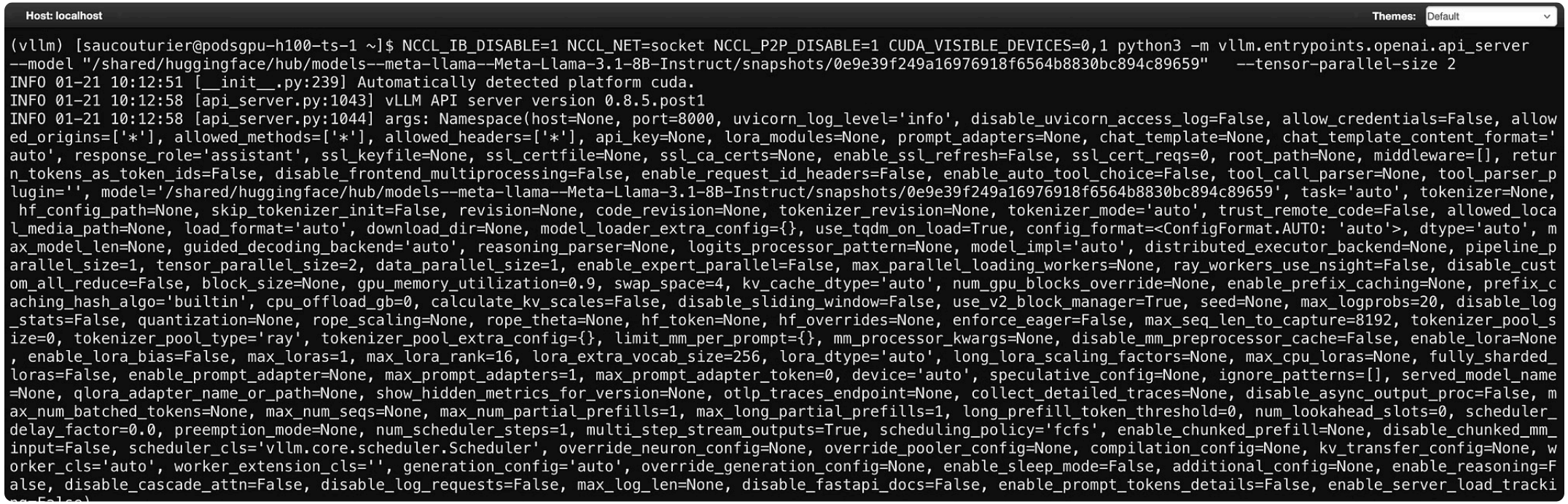
Attention, la configuration GPU nécessite de positionner les variables d'environnement suivantes :

- NCCL_IB_DISABLE=1
- NCCL_NET=socket
- NCCL_P2P_DISABLE=1

Dans le cas contraire, les GPUS crash et il faut rebooter physiquement le serveur GPUH100 (bug nvidia).

```
$> NCCL_IB_DISABLE=1 NCCL_NET=socket NCCL_P2P_DISABLE=1 CUDA_VISIBLE_DEVICES=0,1 python3 -m vllm.entrypoints.openai.api_server --model "/shared/huggingface/hub/models--meta-llama--Meta-Llama-3.1-8B-Instruct/snapshots/0e9e39f249a16976918f6564b8830bc894c89659" --tensor-parallel-size 2
```

Voici le résultat obtenu :



```
INFO 01-21 10:15:06 [kv_cache_utils.py:634] GPU KV cache size: 1,150,720 tokens
INFO 01-21 10:15:06 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.78x
INFO 01-21 10:15:06 [kv_cache_utils.py:634] GPU KV cache size: 1,150,720 tokens
INFO 01-21 10:15:06 [kv_cache_utils.py:637] Maximum concurrency for 131,072 tokens per request: 8.78x
(VllmWorker rank=1 pid=23053) INFO 01-21 10:15:26 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VllmWorker rank=0 pid=23052) INFO 01-21 10:15:26 [custom_all_reduce.py:195] Registering 4355 cuda graph addresses
(VllmWorker rank=1 pid=23053) INFO 01-21 10:15:26 [gpu_model_runner.py:1686] Graph capturing finished in 21 secs, took 0.60 GiB
(VllmWorker rank=0 pid=23052) INFO 01-21 10:15:26 [gpu_model_runner.py:1686] Graph capturing finished in 21 secs, took 0.60 GiB
INFO 01-21 10:15:26 [core.py:159] init engine (profile, create kv cache, warmup model) took 80.49 seconds
INFO 01-21 10:15:26 [core_client.py:439] Core engine process 0 ready.
WARNING 01-21 10:15:27 [config.py:1239] Default sampling parameters have been overridden by the model's Hugging Face generation config recommended from the model creator
. If this is not intended, please relaunch vLLM instance with `--generation-config vllm`.
INFO 01-21 10:15:27 [serving_chat.py:118] Using default chat sampling params from model: {'temperature': 0.6, 'top_p': 0.9}
INFO 01-21 10:15:27 [serving_completion.py:61] Using default completion sampling params from model: {'temperature': 0.6, 'top_p': 0.9}
INFO 01-21 10:15:27 [api_server.py:1090] Starting vLLM API server on http://0.0.0.0:8000
INFO 01-21 10:15:27 [launcher.py:28] Available routes are:
INFO 01-21 10:15:27 [launcher.py:36] Route: /openapi.json, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /docs, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /docs/oauth2-redirect, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /redoc, Methods: GET, HEAD
INFO 01-21 10:15:27 [launcher.py:36] Route: /health, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /load, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /ping, Methods: POST, GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /tokenize, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /detokenize, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/models, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /version, Methods: GET
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/chat/completions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/completions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/embeddings, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /pooling, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /score, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/score, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/audio/transcriptions, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v1/rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /v2/rerank, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /invocations, Methods: POST
INFO 01-21 10:15:27 [launcher.py:36] Route: /metrics, Methods: GET
INFO:     Started server process [22650]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

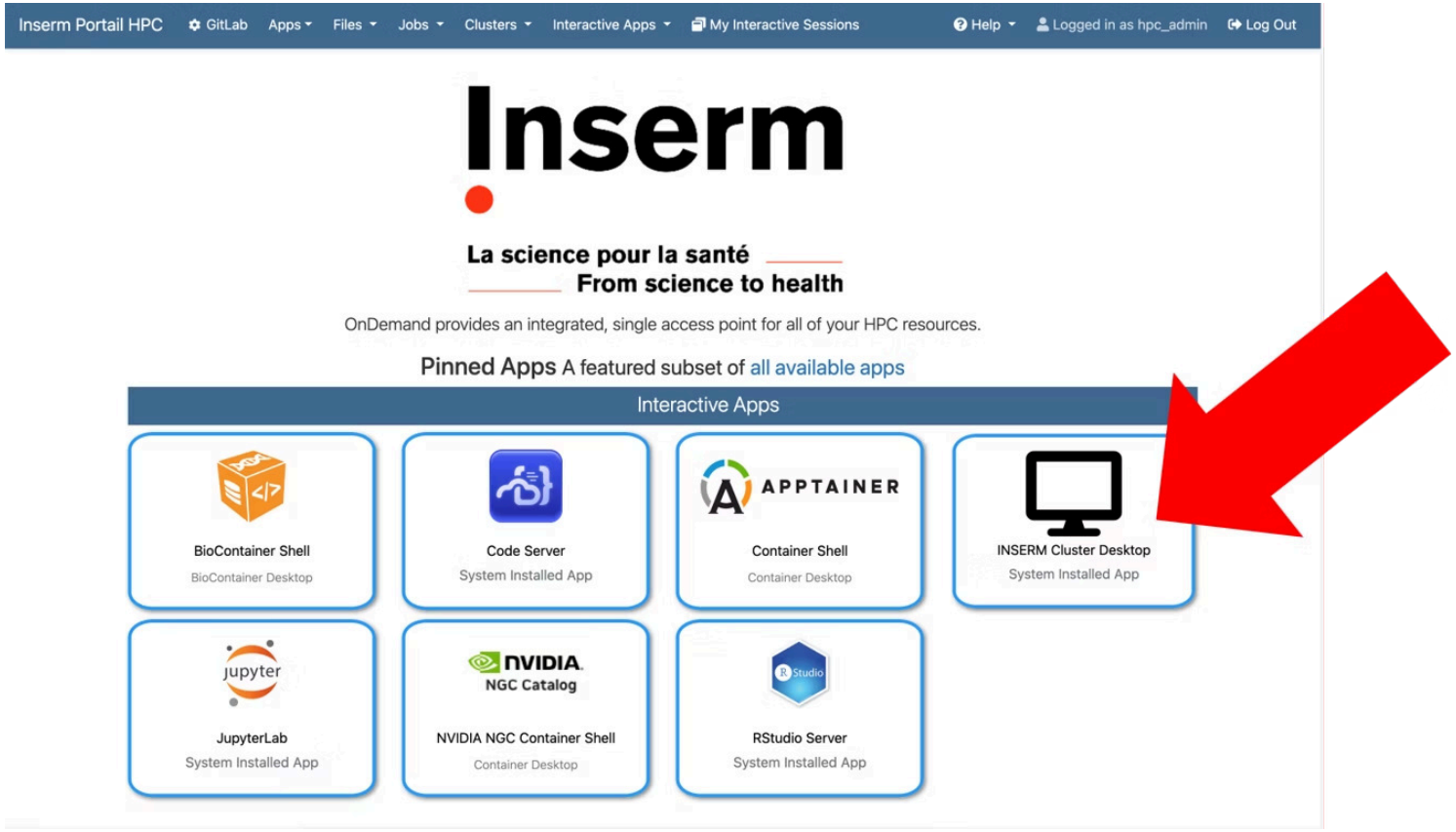
Le code a parfaitement fonctionné et a utilisé la parallélisation des calculs sur deux GPUS.

Utiliser "CTRL-C" pour arrêter le code.

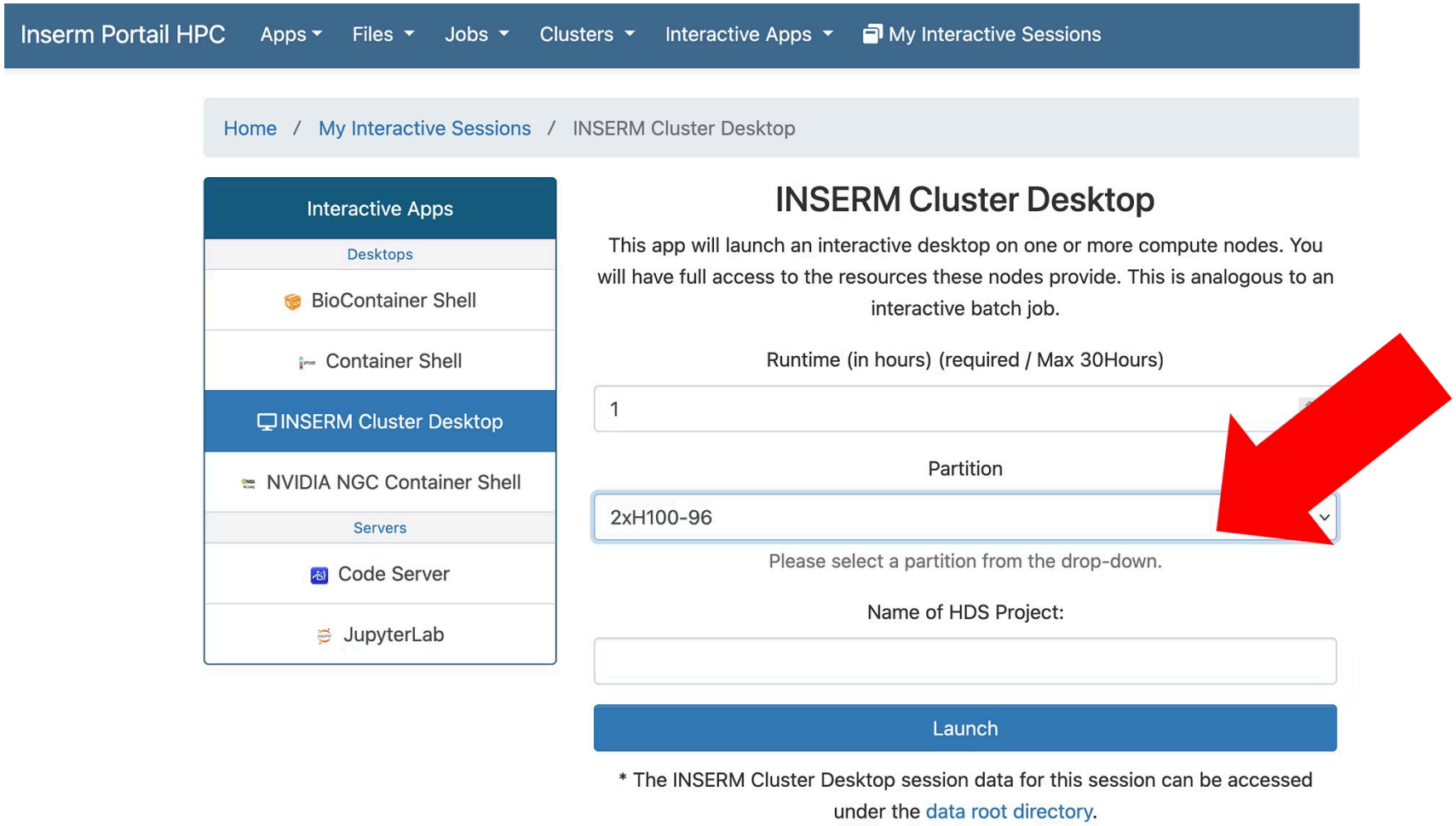
4 Mise en œuvre des calculs via une GUI Linux

Pour bénéficier d'une interface graphique, nous pouvons réserver un nœud de calcul via le widget Inserm Cluster Desktop et y accéder avec une session GUI.

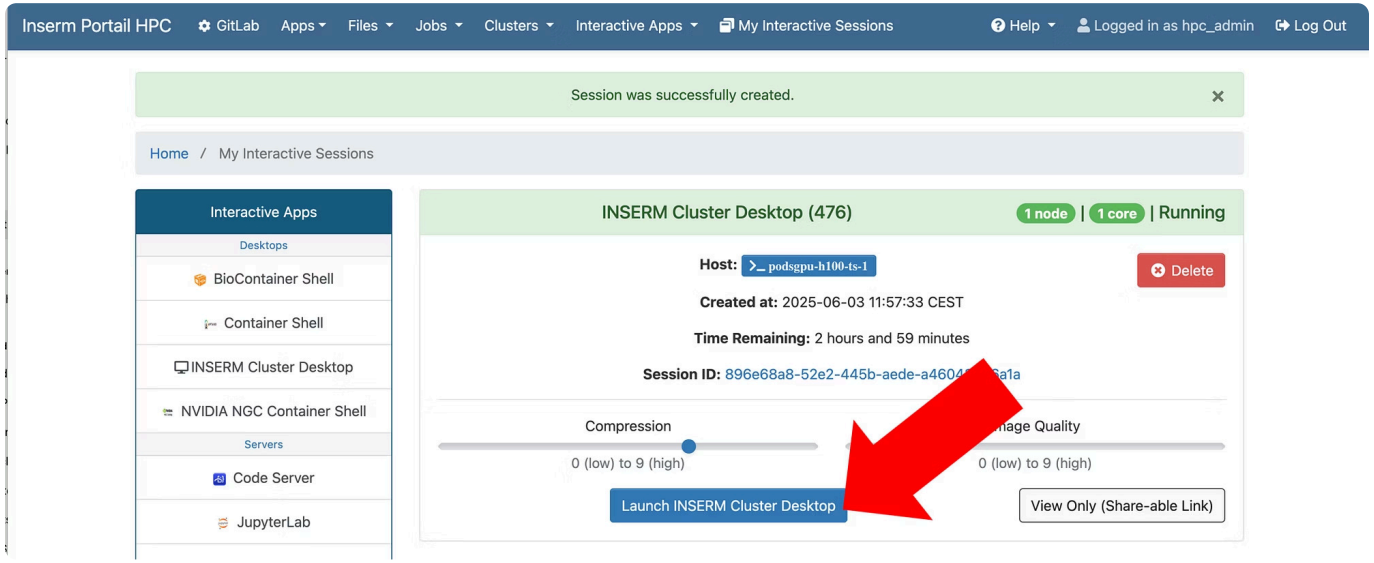
1) Choisir le widget Inserm Cluster Desktop



2) Sélectionner la partition GPU :



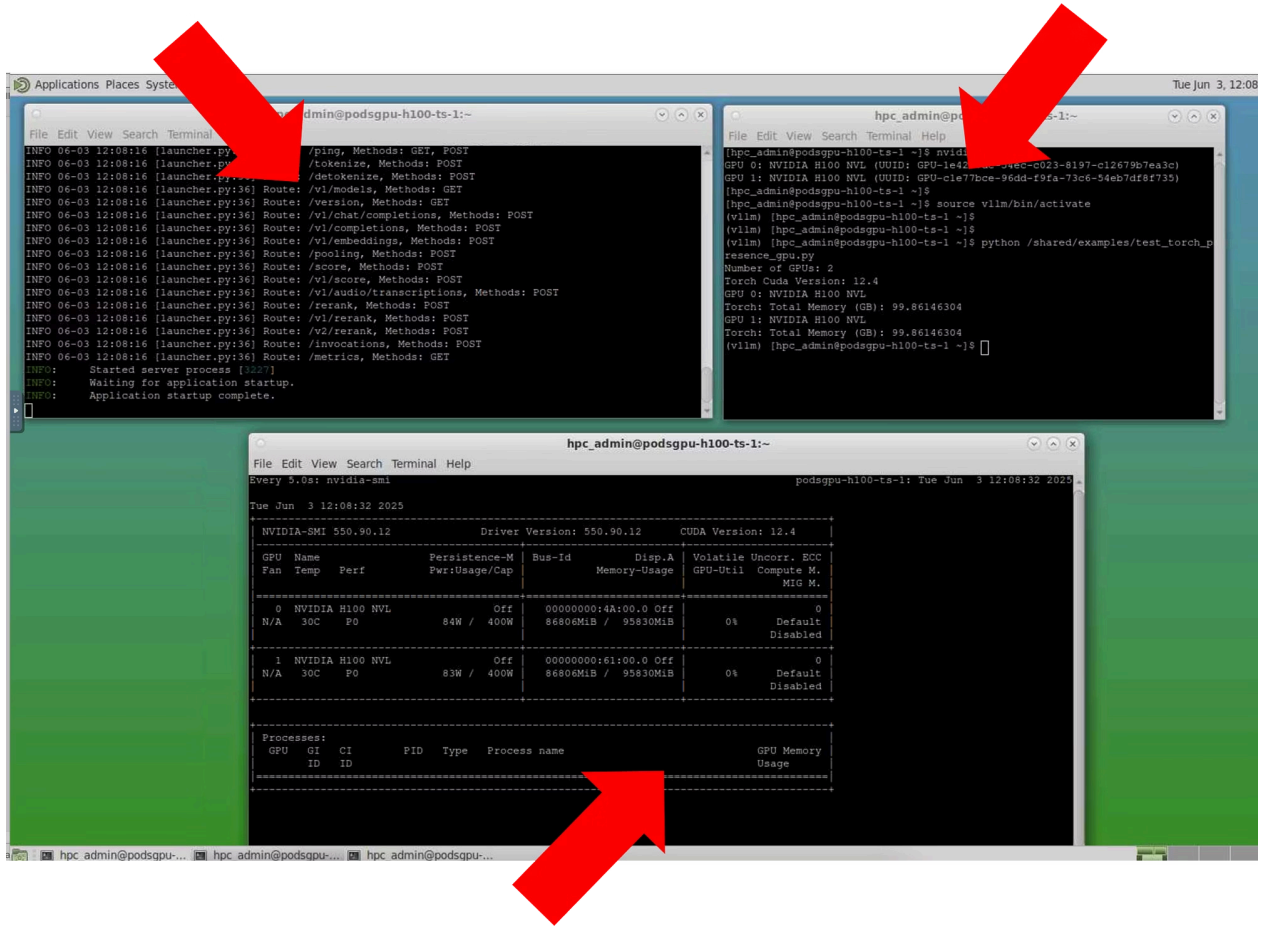
3) Cliquer sur Launch Inserm Cluster Desktop pour accéder à l'interface



4) Rejouer les commandes du chapitre 3 pour activer le venv, vérifier la présence GPU et lancer le vllm.

Test VLLM via le venv "vllm"

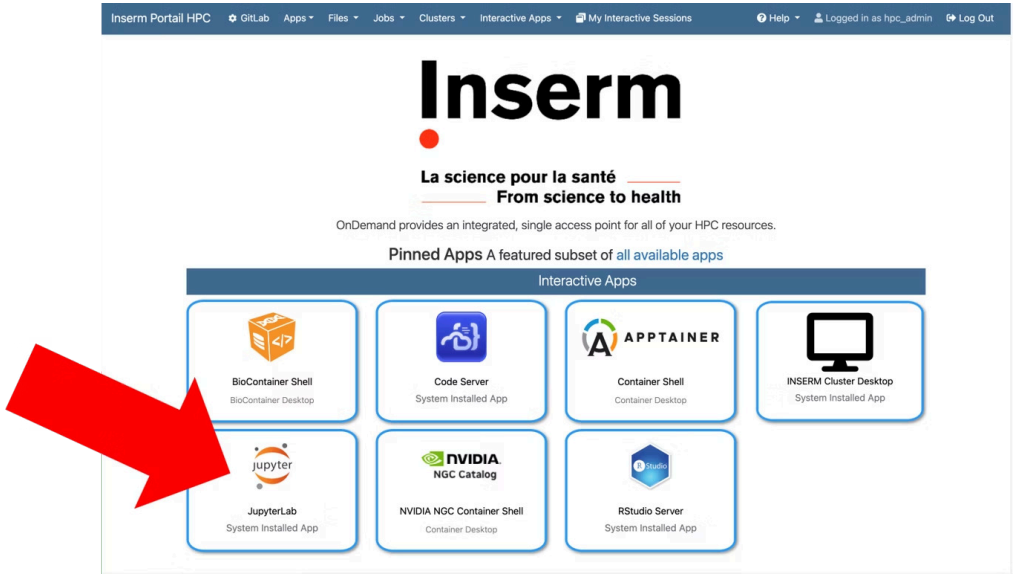
Test Presence GPU via le venv "vllm"



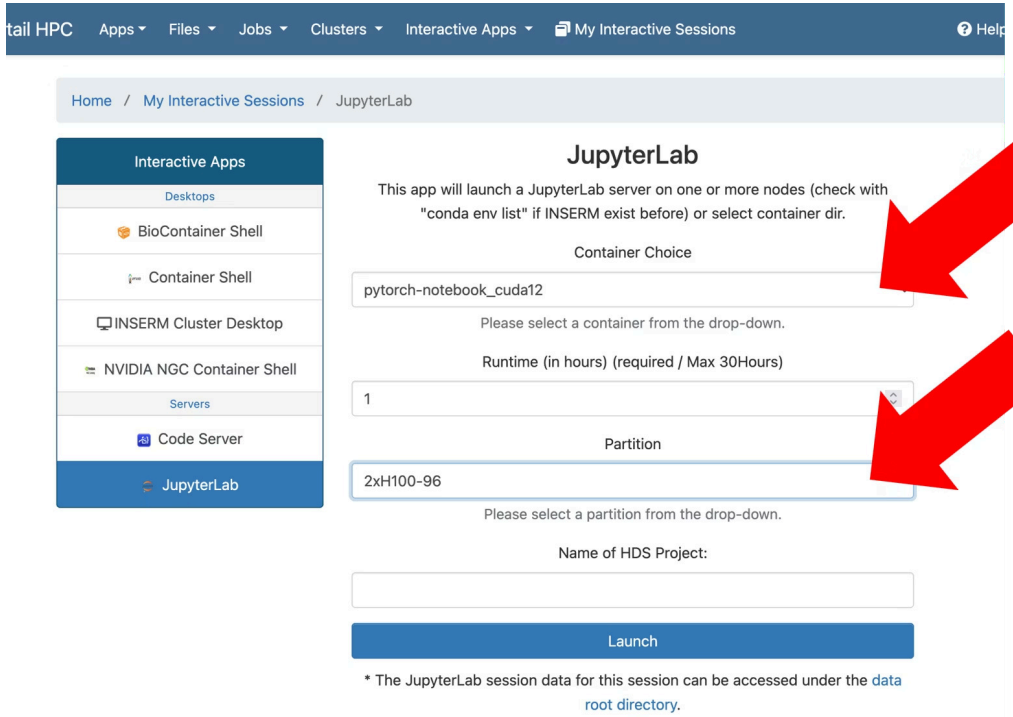
Surveillance GPUs via un "watch -n 5 nvidia-smi »

5 Mise en œuvre des calculs via jupyterhub

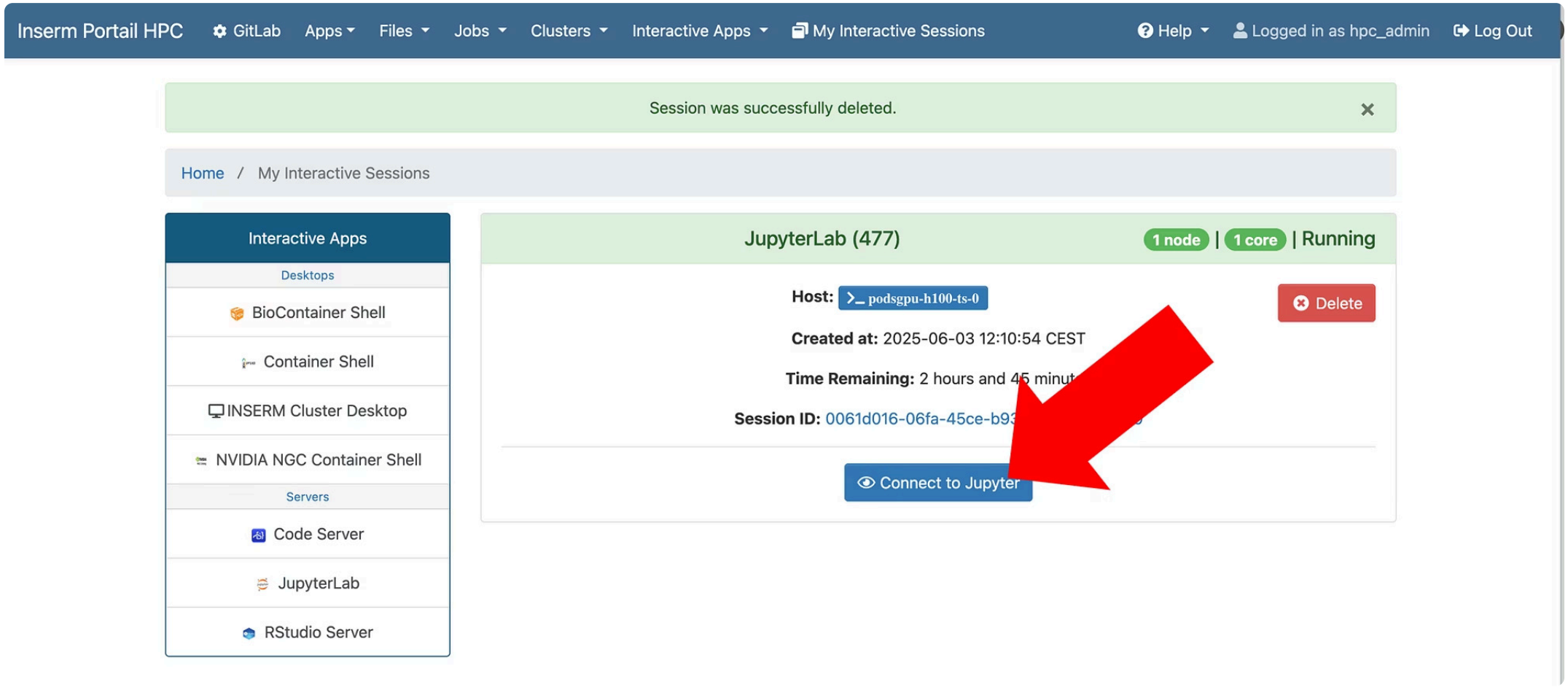
1) Choisir le widget JupyterLab



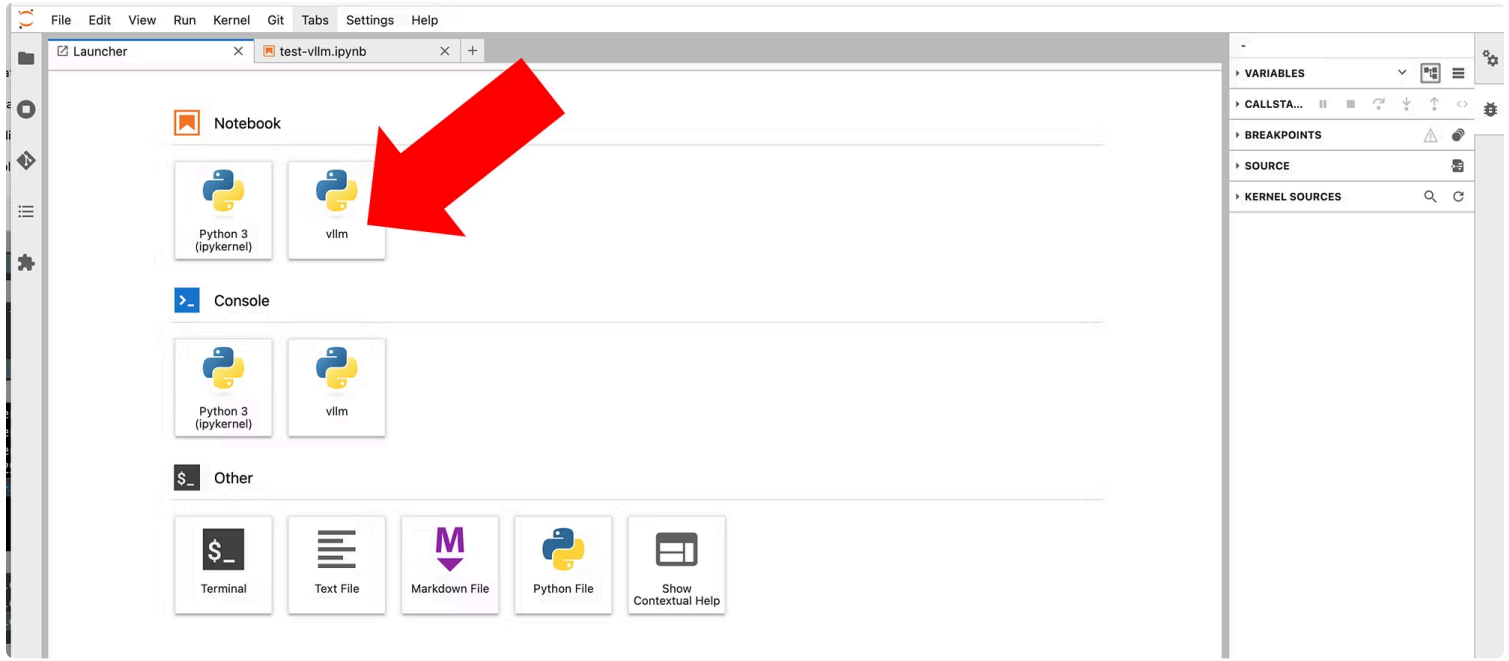
2) Sélectionner le container choice pytorch-notebook_cuda12 et la partition 2xH100-96



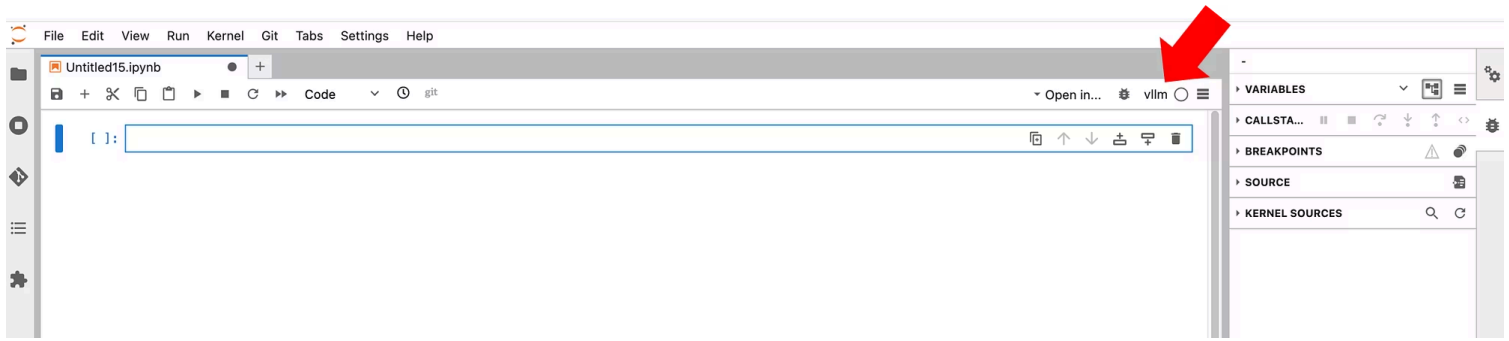
3) Cliquer sur Connect to Jupyter pour accéder à l'interface



4) Cliquer sur "vllm" disponible dans l'interface jupyterhub de notre compte grâce à l'enregistrement de l'environnement virtuel au chapitre 2.

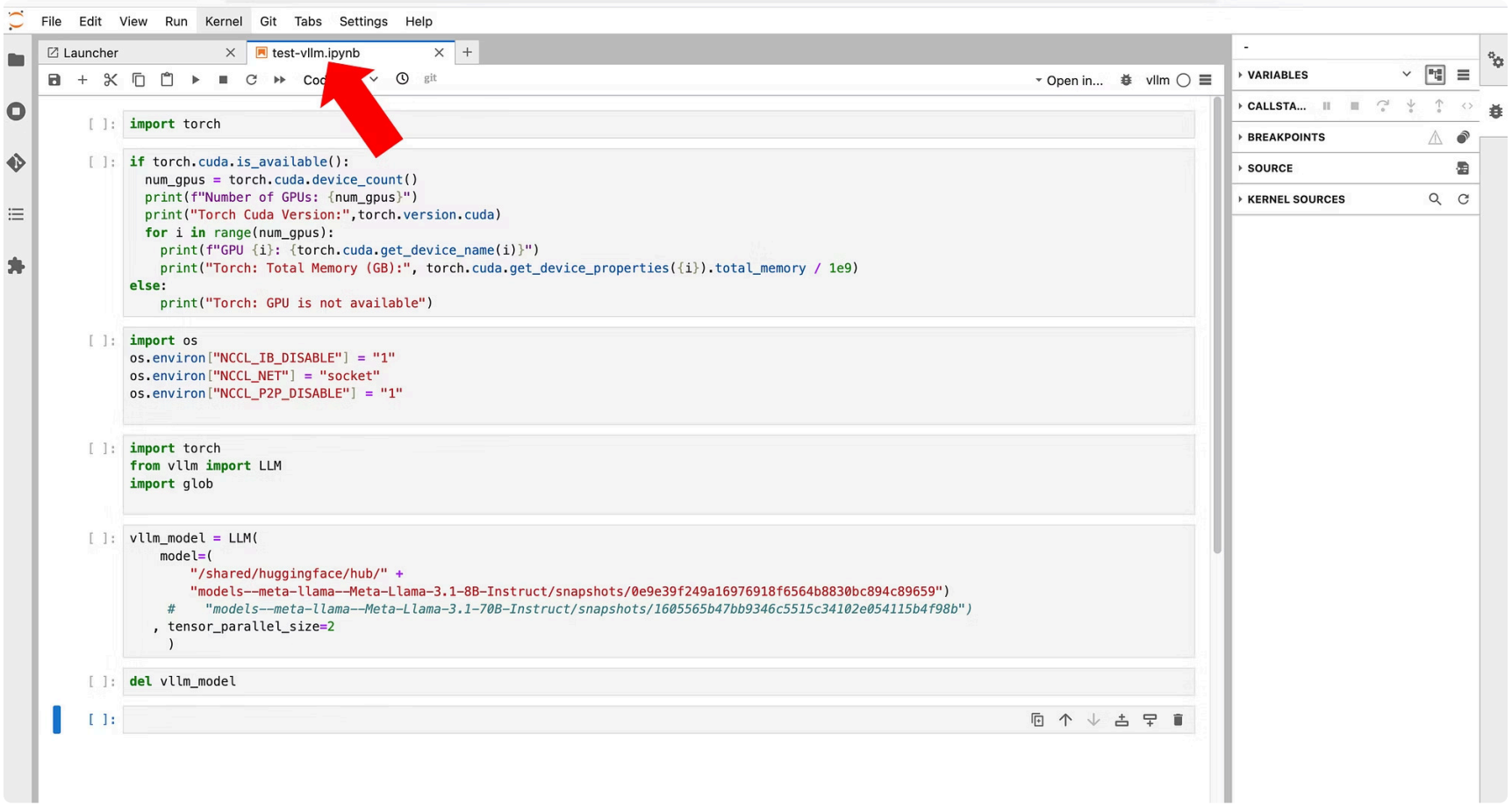


4 bis) Bien s'assurer d'être dans le venv vllm dans un notebook python vide (donc à python 3.11)



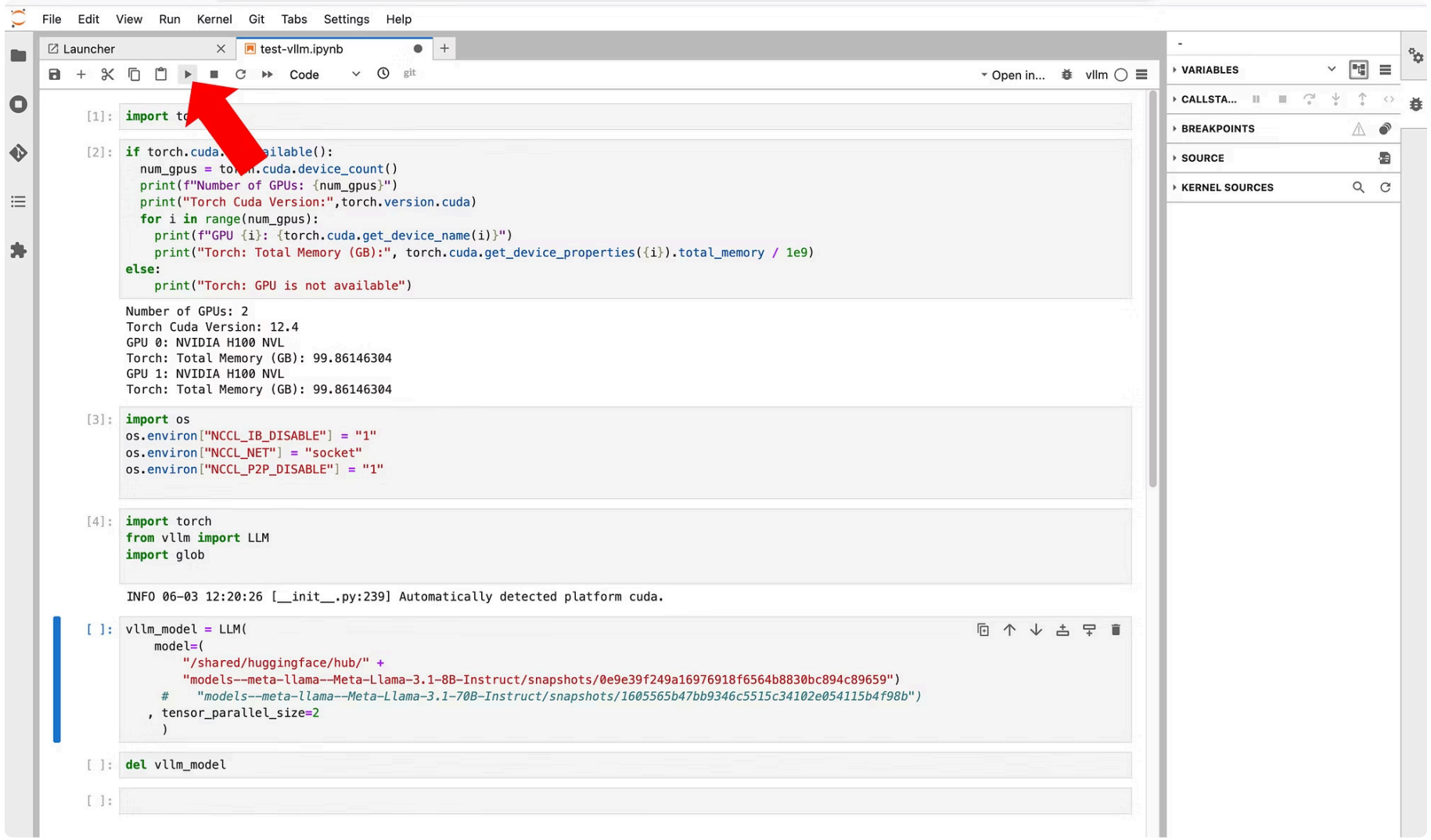
5) Cliquer sur l'onglet test vllm

Le code applicatif est recopié sous forme de bloc exécutif et le fichier est sauvegardé :

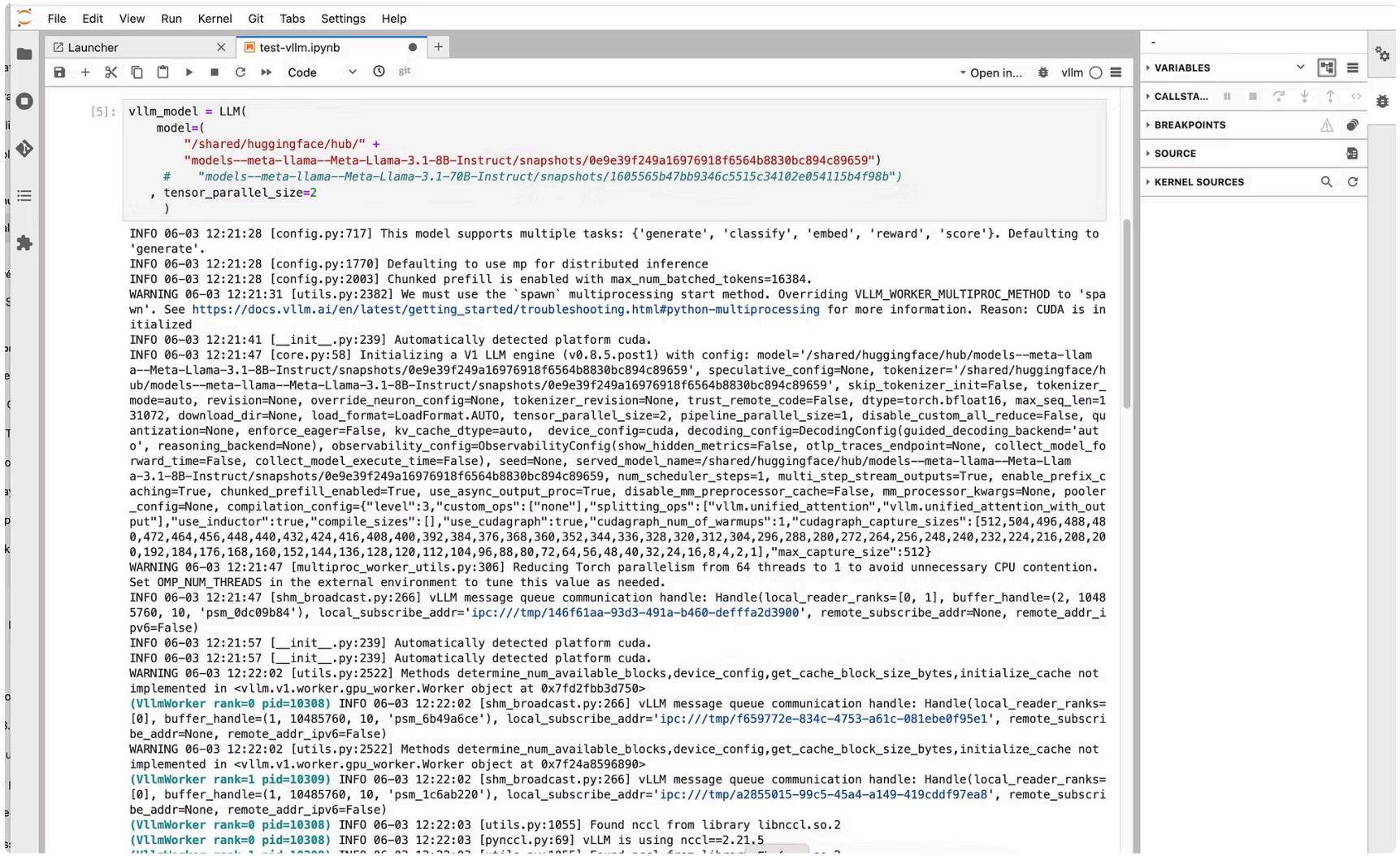


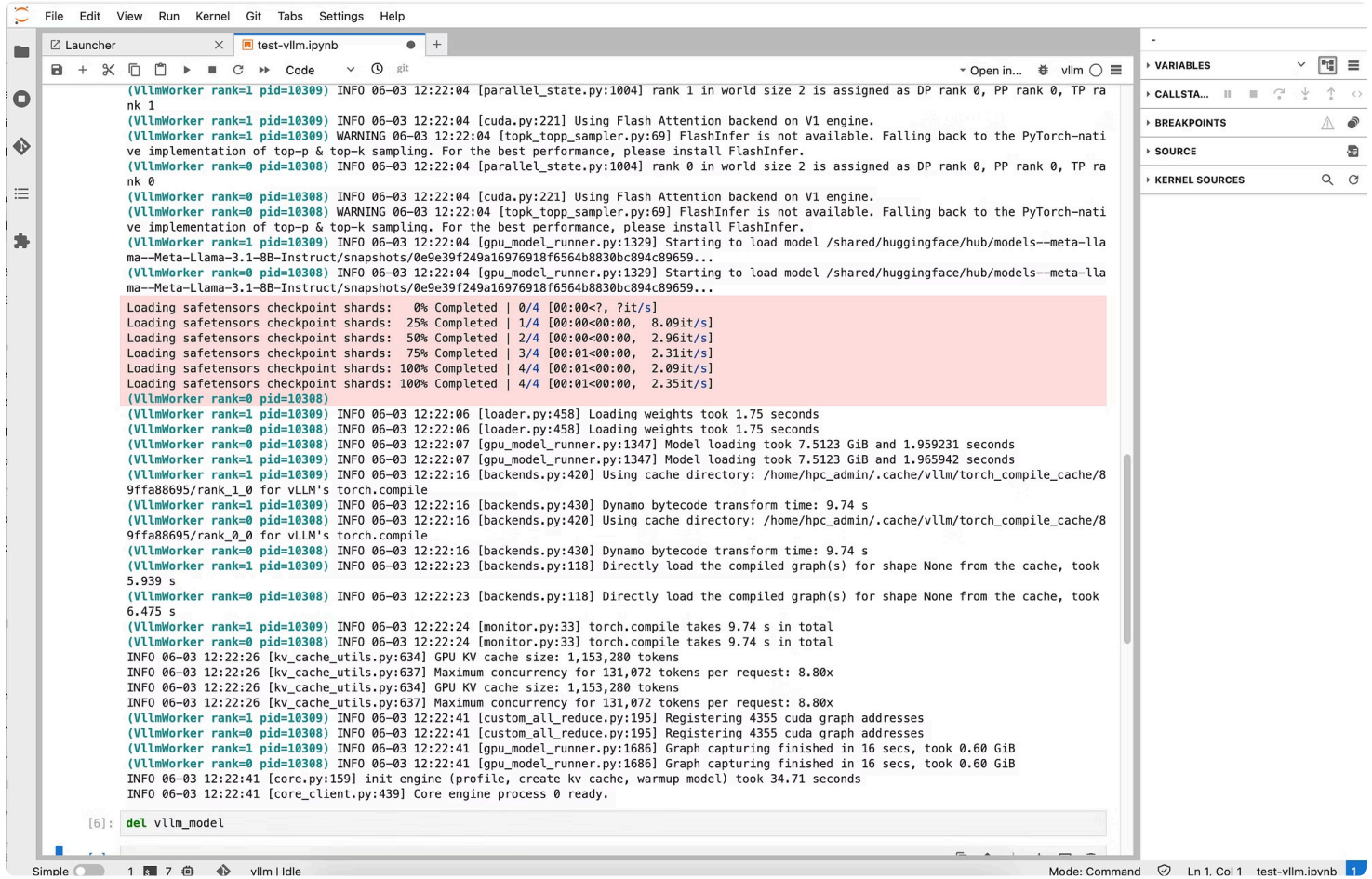
Point d'attention : Le bloc 3 du code configure les variables d'environnement essentielles pour prévenir tout crash des GPUs (voir détails au chapitre 3.4).

6) Cliquer sur le bouton play pour procéder au run, bloc après bloc



Voici le run du bloc test VLLM ci-dessous :





Succès : Le bloc test VLLM s'exécute parfaitement ! Les ressources GPU sont ensuite correctement libérées grâce à l'instruction « del vllm_model ».

Si vous avez des questions, contacter vos référents habituels sur le HPC.