

# Tensorflow GPU: Windows

## Introducción

Este documento proporciona una guía paso a paso para configurar un entorno de desarrollo en Windows que permita utilizar TensorFlow con soporte para GPU. A través del uso de Docker, WSL (Windows Subsystem for Linux), y contenedores Devcontainers en Visual Studio Code, se detalla cómo instalar y verificar correctamente los componentes necesarios, incluyendo controladores de GPU NVIDIA, Docker Desktop y la integración con WSL. Al finalizar esta guía, el usuario podrá ejecutar scripts de Python y Jupyter Notebooks aprovechando la aceleración por hardware de la GPU, facilitando así el desarrollo de proyectos de aprendizaje profundo y otras aplicaciones intensivas en cómputo.

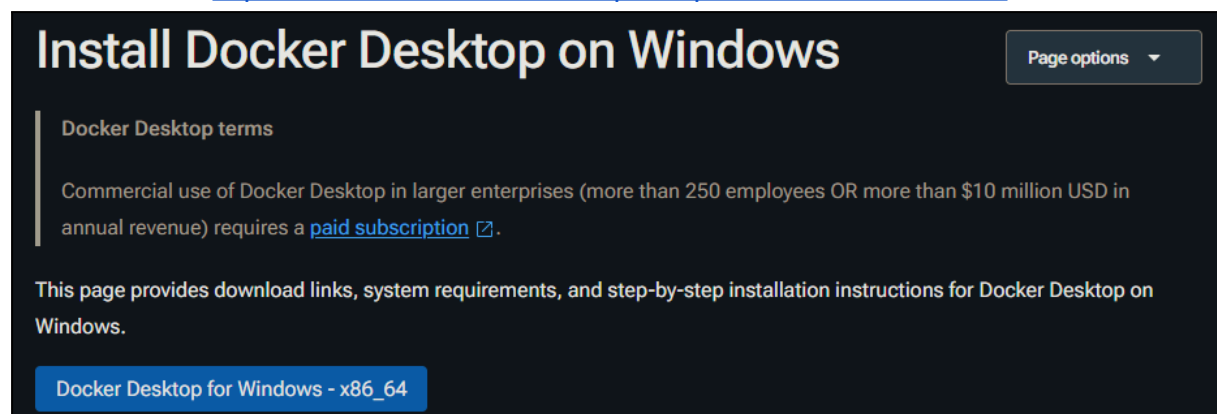
## Guía paso a paso

(Opcional) Descargamos e instalamos la última versión de los drivers de la placa de video en caso de no tenerlos ya instalados:

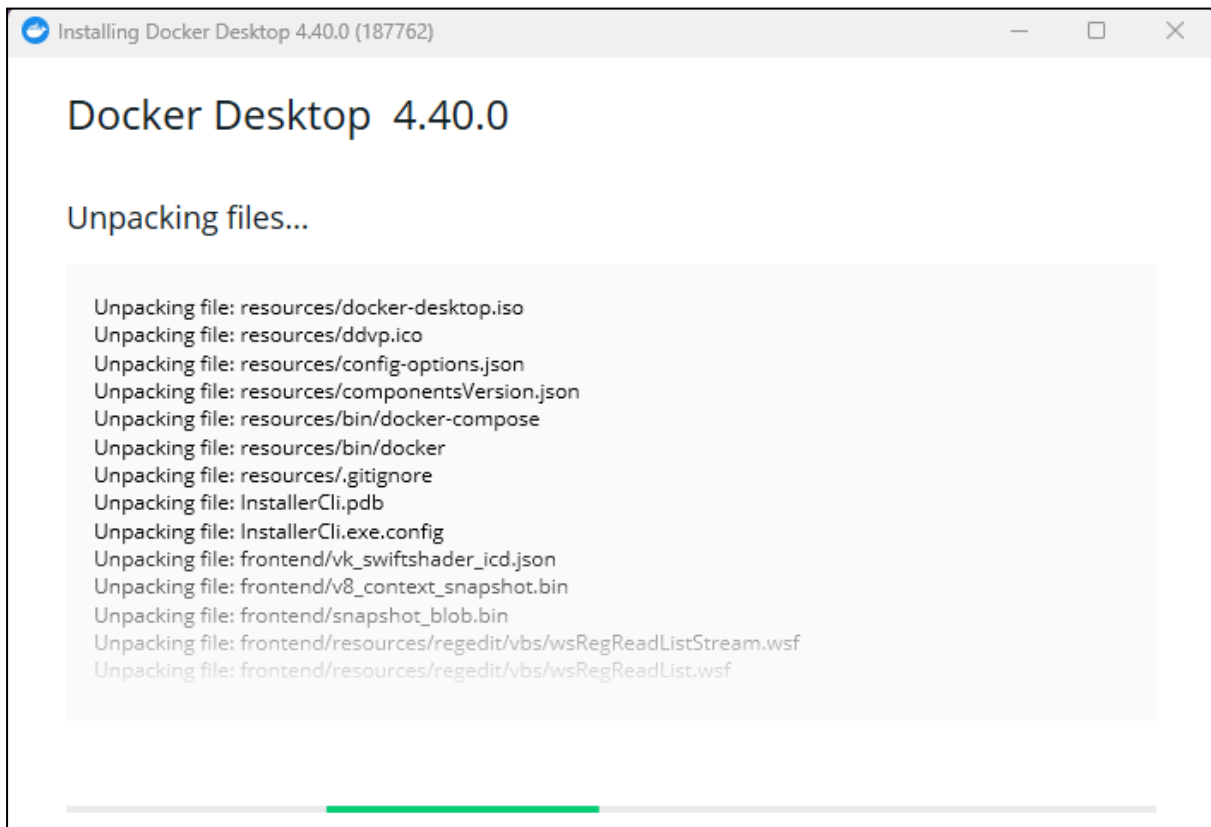
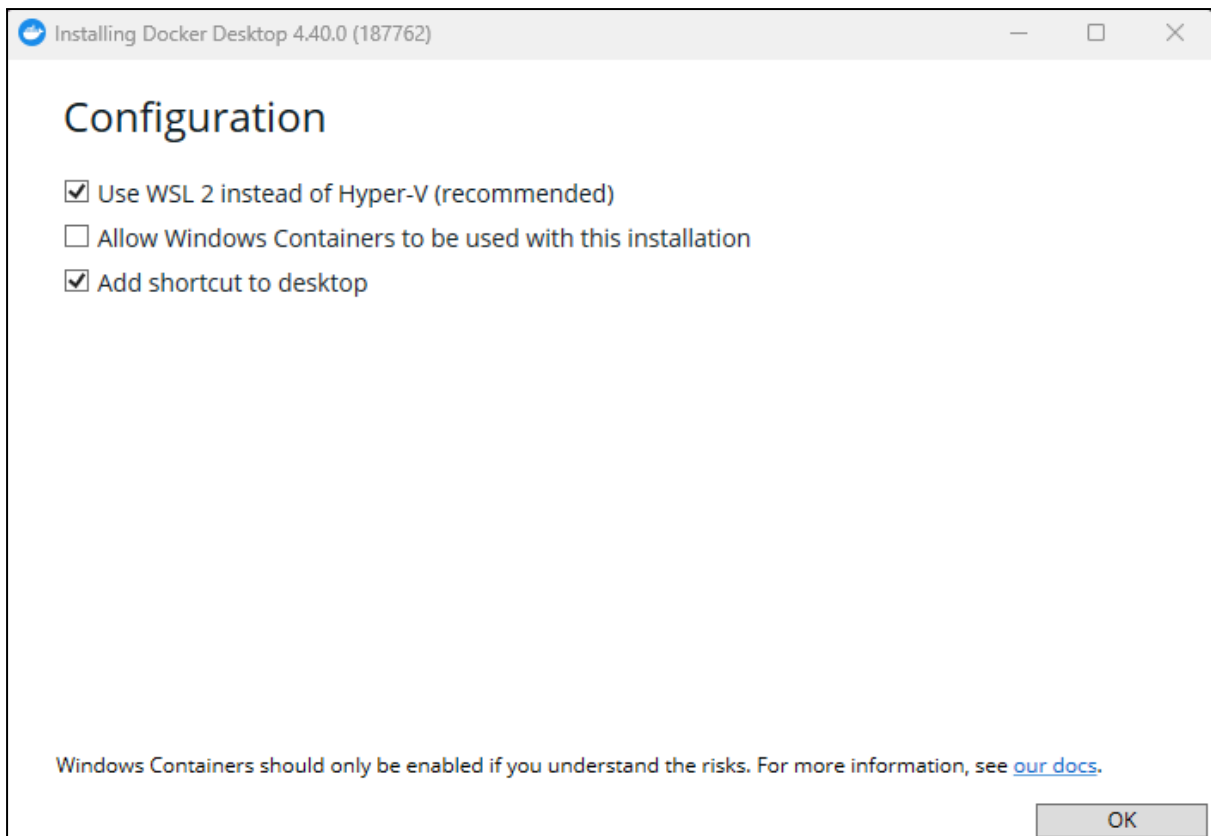
<https://www.nvidia.com/en-us/drivers/>

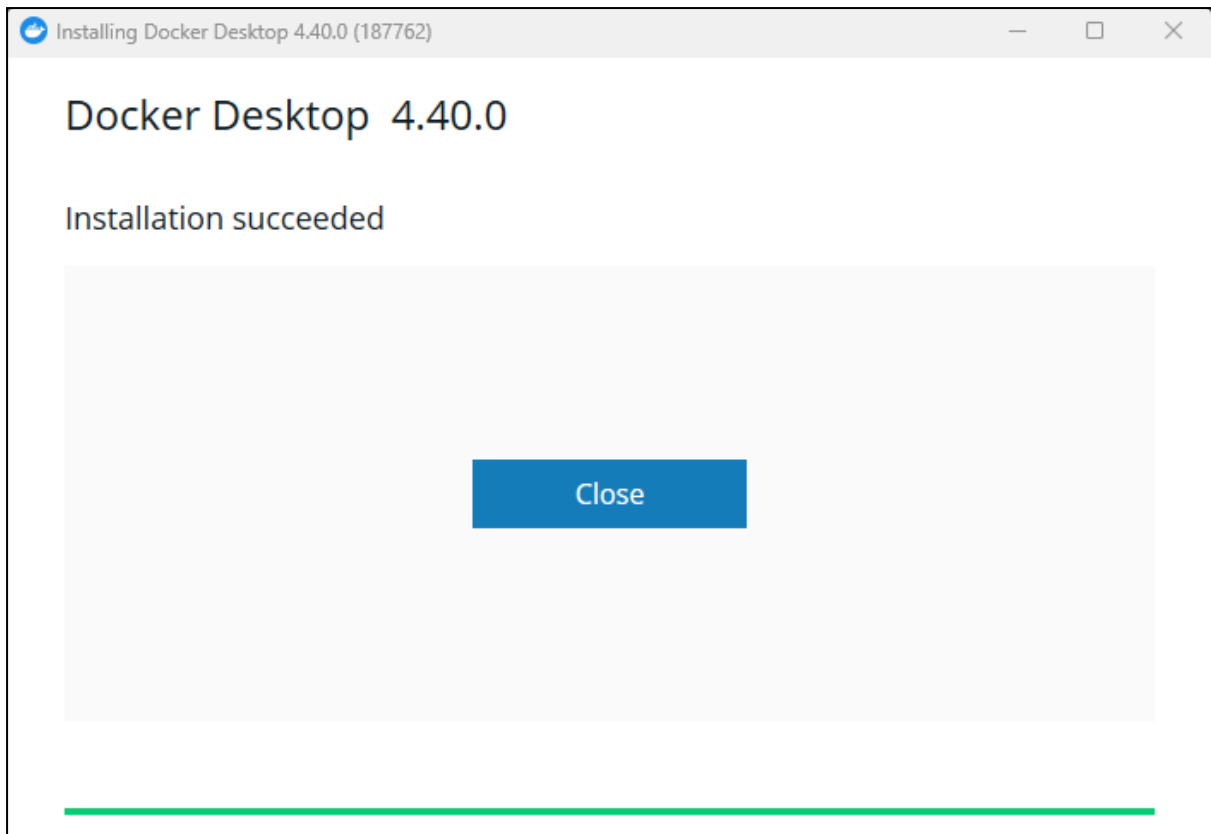
Descargamos Docker Desktop Installer del siguiente link:

<https://docs.docker.com/desktop/setup/install/windows-install/>



Abrimos el archivo .exe descargado y seguimos los pasos:



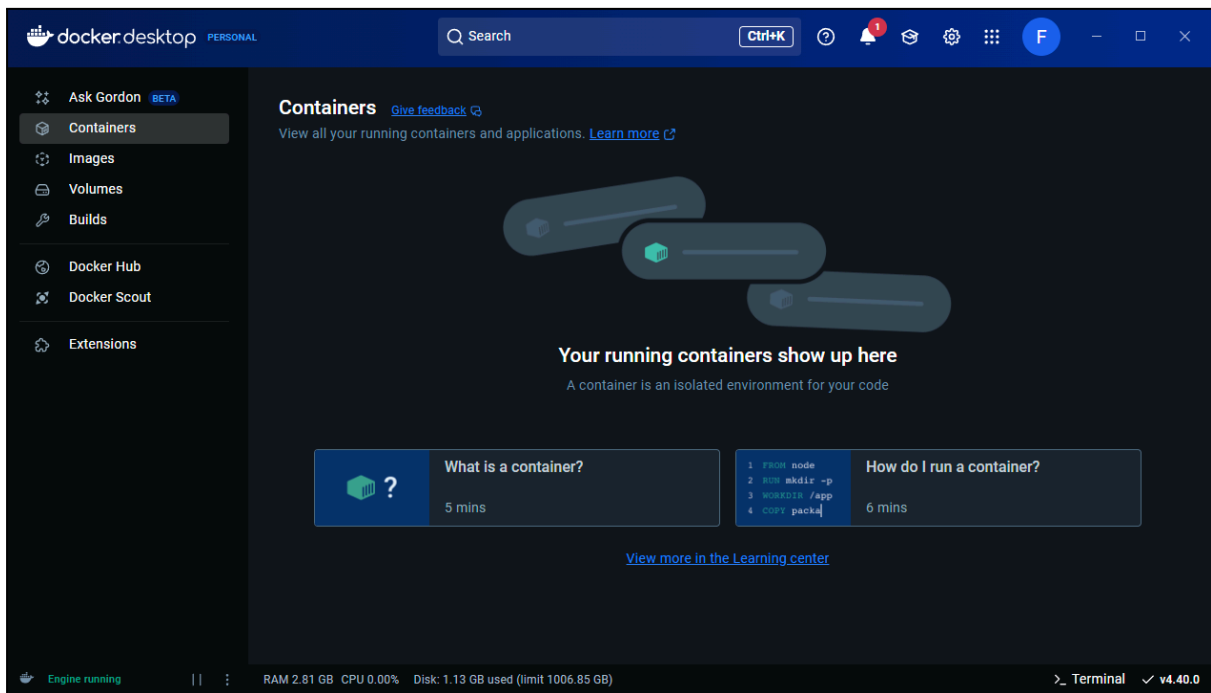


Una vez instalado Docker Desktop:

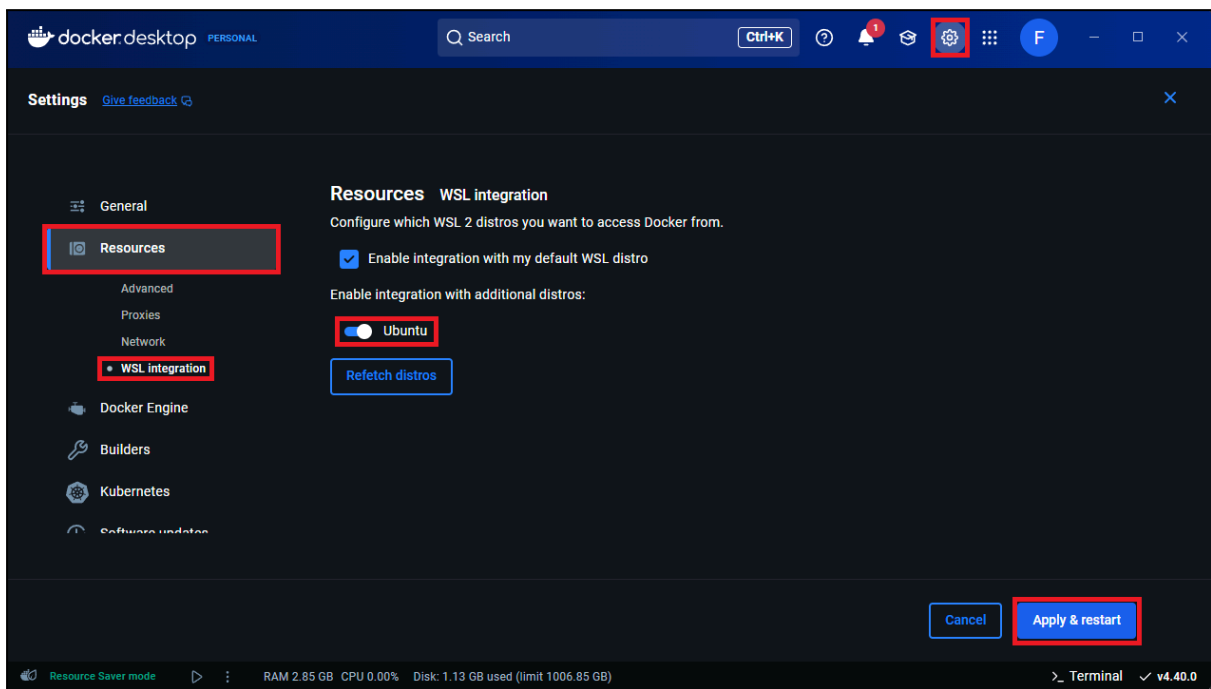
- Abrimos una terminal de Windows Powershell (Win + X, i).
- Instalamos la distro de ubuntu para wsl: `wsl --install -d Ubuntu`
- Configuramos Ubuntu como la distro por defecto: `wsl --set-default Ubuntu`
- Verificamos que se instaló correctamente: `wsl --list --verbose`

```
Windows PowerShell
PS C:\Users\Facu> wsl --install -d Ubuntu
Downloading: Ubuntu
Installing: Ubuntu
Distribution successfully installed. It can be launched via 'wsl.exe -d Ubuntu'
PS C:\Users\Facu> wsl --set-default Ubuntu
The operation completed successfully.
PS C:\Users\Facu> wsl --list --verbose
  NAME      STATE      VERSION
* Ubuntu    Stopped    2
PS C:\Users\Facu>
```

Abrimos Docker Desktop:



Habilitamos la integración con Ubuntu:



En una terminal de Windows Powershell levantamos un contenedor de docker de pruebas para corroborar que se instalo todo correctamente y la GPU está disponible:

```
docker run --env NVIDIA_DISABLE_REQUIRE=1 --gpus all  
nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
```

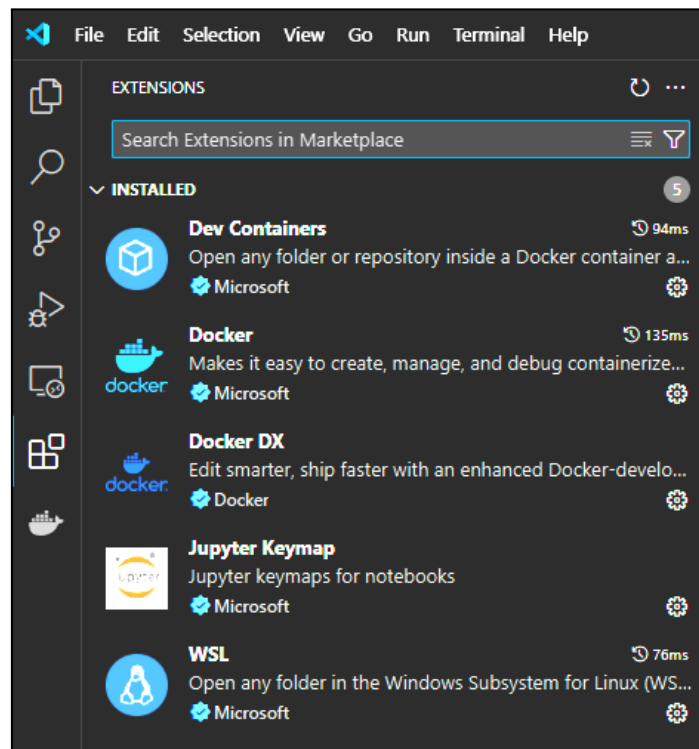
```
PS C:\Users\Facu> docker run --env NVIDIA_DISABLE_REQUIRE=1 --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
Unable to find image 'nvcr.io/nvidia/k8s/cuda-sample:nbody' locally
nbody: Pulling from nvidia/k8s/cuda-sample
9026fb14bf88: Pull complete
a424d45fd86f: Pull complete
207b64ab7ce6: Pull complete
e9bfff09d04df: Pull complete
1f37f461c076: Pull complete
f65423f1b49b: Pull complete
2b60900a3ea5: Pull complete
22c5ef60a68e: Pull complete
1939e4248814: Pull complete
548afb82c856: Pull complete
edc14edf1b04: Pull complete
Digest: sha256:59261e419d6d48a772aad5bb213f9f1588fcd042b115ceb7166c89a51f03363
Status: Downloaded newer image for nvcr.io/nvidia/k8s/cuda-sample:nbody
Run "nbody -benchmark [-numbodies=<numBodies>]" to measure performance.
    -fullscreen      (run n-body simulation in fullscreen mode)
    -fp64             (use double precision floating point values for simulation)
    -hostmem          (stores simulation data in host memory)
    -benchmark        (run benchmark to measure performance)
    -numbodies=<N>    (number of bodies (>= 1) to run in simulation)
    -device=<d>        (where d=0,1,2,... for the CUDA device to use)
    -numdevices=<i>    (where i=(number of CUDA devices > 0) to use for simulation)
    -compare          (compares simulation results running once on the default GPU and once on the CPU)
    -cpu              (run n-body simulation on the CPU)
    -tipsy=<file.bin> (load a tipsy model file for simulation)

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.

> Windowed mode
> Simulation data stored in video memory
> Single precision floating point simulation
> 1 Devices used for simulation
GPU Device 0: "Ampere" with compute capability 8.6

> Compute 8.6 CUDA device: [NVIDIA GeForce RTX 3090]
83968 bodies, total time for 10 iterations: 74.298 ms
= 948.961 billion interactions per second
= 18979.218 single-precision GFLOP/s at 20 flops per interaction
```

Abrimos VS Code e instalamos las siguientes extensiones:

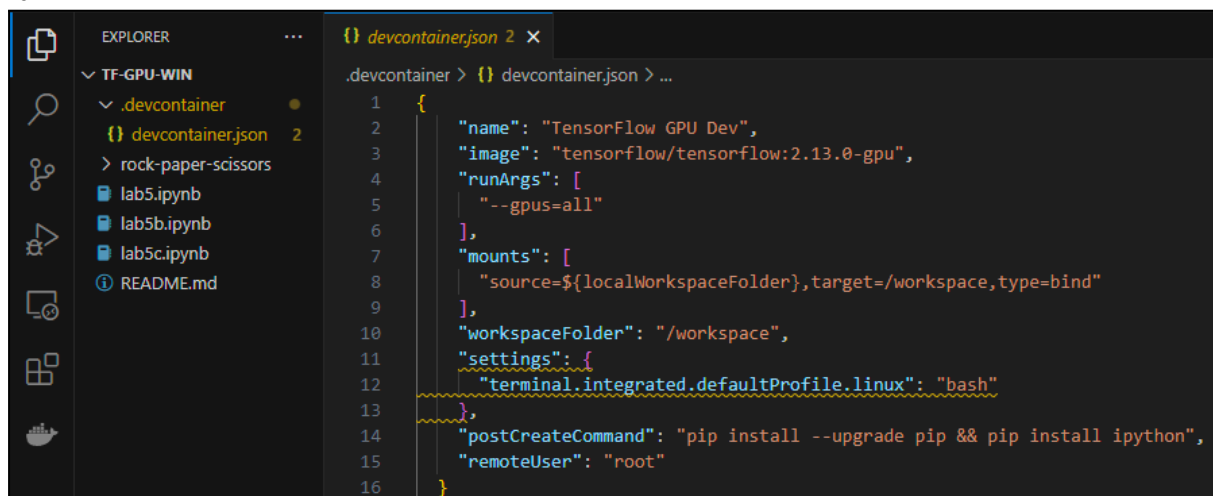


Abrimos algún proyecto o repositorio en el que queramos usar tensorflow con GPU y configuramos devcontainers:

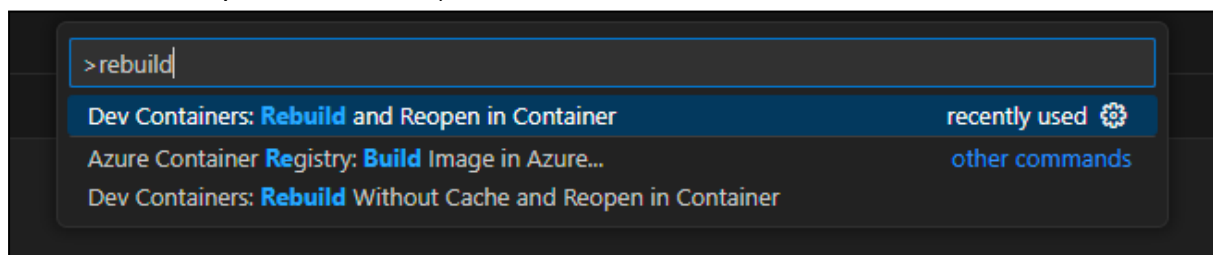
- Creamos una carpeta .devcontainer en la raíz del repositorio.
- Dentro de .devcontainers creamos el archivo devcontainer.json
- Volcamos el siguiente contenido y guardamos:

```
{
  "name": "TensorFlow GPU Dev",
  "image": "tensorflow/tensorflow:2.13.0-gpu",
  "runArgs": [
    "--gpus=all"
  ],
  "mounts": [
    "source=${localWorkspaceFolder},target=/workspace,type=bind"
  ],
  "workspaceFolder": "/workspace",
  "settings": {
    "terminal.integrated.defaultProfile.linux": "bash"
  },
  "postCreateCommand": "pip install --upgrade pip && pip install ipython",
  "remoteUser": "root"
}
```

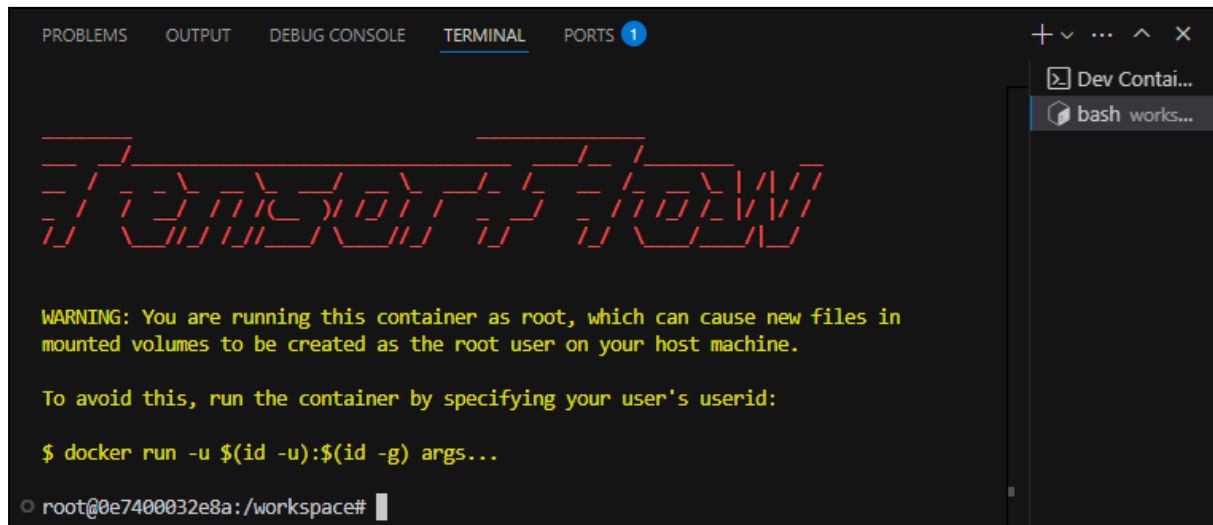
Ejemplo:



Una vez configurado abrimos el contenedor de docker con devcontainers (ctrl + shift + P, Rebuild and Reopen in Container)



Luego de crearse el contenedor si abrimos una nueva consola deberíamos ver:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1
+ Dev Contai...
+ bash works...

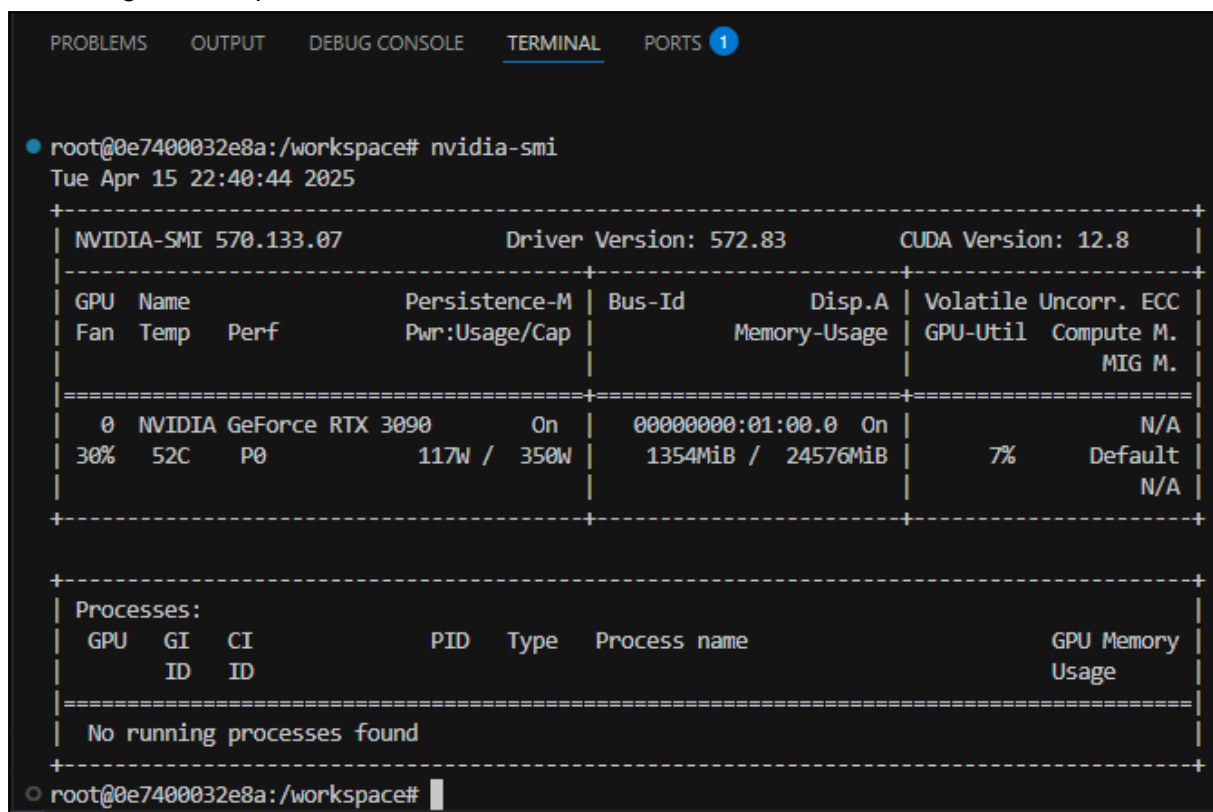
WARNING: You are running this container as root, which can cause new files in
mounted volumes to be created as the root user on your host machine.

To avoid this, run the container by specifying your user's userid:

$ docker run -u $(id -u):$(id -g) args...

root@0e7400032e8a:/workspace#
```

Para asegurarnos que la GPU fue levantada con éxito corremos nvidia-smi:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

root@0e7400032e8a:/workspace# nvidia-smi
Tue Apr 15 22:40:44 2025

+-----+
| NVIDIA-SMI 570.133.07              Driver Version: 572.83          CUDA Version: 12.8          |
+-----+-----+-----+-----+-----+-----+
| GPU   Name                               Persistence-M   Bus-Id        Disp.A         Volatile Uncorr. ECC      |
| Fan  Temp  Perf    Pwr:Usage/Cap       Pwr:Usage/Cap    Memory-Usage   GPU-Util    Compute M.             |
|=====-=+=====+=====+=====+=====+=====+=====+=====+
|  0  NVIDIA GeForce RTX 3090             On               00000000:01:00.0 On              7%            N/A             |
| 30%   52C   P0             117W / 350W      1354MiB / 24576MiB             Default             |
|                                         N/A             |
+-----+-----+-----+-----+-----+-----+

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

root@0e7400032e8a:/workspace#
```

Si seguimos todos los pasos correctamente deberíamos poder correr tanto scripts de python que usen tensorflow con GPU así como también Jupyter Notebooks. En caso de que los Jupyter no corran, asegúrese de instalar las librerías (pip3 install) y de seleccionar el Kernel para correr los Jupyter Notebooks.

