**NVIDIA FLARE with Docker**

**GPU configuration in Windows System**

The supported operation system for NVIDIA FLARE is Linux or OSX (Note: some optional dependencies are not compatible, such as tenseal and openmined.psi). There is no guide available on how to setup GPU configuration for containerized deployment of NVIDIA FLARE with Docker in Windows system.

**Install NVIDIA Driver for GPU Support**

Install NVIDIA GeForce Game Ready or NVIDIA RTX Quadro Windows 11 display driver on your system with a compatible GeForce or NVIDIA RTX/Quadro card from <https://www.nvidia.com/Download/index.aspx>.

A screenshot of a computer downloading

Description automatically generated

A screenshot of a computer

Description automatically generated

**This is the only driver you need to install. Do not install any Linux display driver in WSL.**

**Install WSL 2**

With Windows Subsystem for Linux (WSL), developers can access the power of both Windows and Linux at the same time on a Windows machine. WSL 2 is a key enabler in making GPU acceleration to be seamlessly shared between Windows and Linux applications on the same system a reality.

## Prerequisites

You must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11 to use the commands below.

Open PowerShell or Windows Command Prompt in **administrator** mode by right-clicking and selecting "Run as administrator", and then install WSL. You will see a list of available distros.

wsl.exe --install

Ensure you have the latest WSL kernel:

wsl.exe --update

You may need to download the latest package from linkage below. Install downloaded file, wsl\_update\_x64.msi.

[WSL2 Linux kernel update package for x64 machines](https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi)

Run below to install Ubuntu-22.04 (NVIDIA FLARE requires Python 3.8+, Ubuntu-22.04 comes with Python 3.10. )

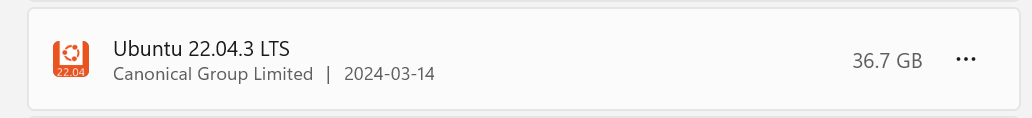
Correction: Python 3.8+ means Python 3.8 or higher which means Python 3.8 is compatible. If running NVIDIA FLARE in a virtual environment, you need to create a virtual environment with venv. Note that in newer versions of Ubuntu (after Ubuntu 20.04), you may need to make sure you are using Python 3.8 and not a newer version.

wsl --install -d Ubuntu-22.04

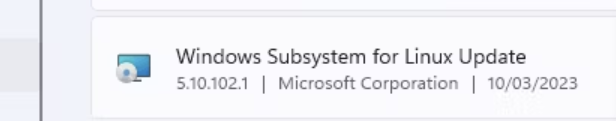
A computer screen shot of a computer

Description automatically generated

After that, you should be able to see Ubuntu-22.04 available on your pc. Run it and set up the username and password the first time you run Ubuntu-22.04.



When you see an error of “WSL 2 requires an update to its kernel component” when running Ubuntu-22.04, check if below is in place in your installed apps. If not, make sure [WSL2 Linux kernel update package for x64 machines](https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi) is downloaded and installed.



## Reference

<https://learn.microsoft.com/en-us/windows/wsl/install-manual> (Step 4 - Download the Linux kernel update package)

**Setup CUDA Support for WSL 2**

Once a Windows NVIDIA GPU driver is installed on the system, CUDA becomes available within WSL 2. The CUDA driver installed on Windows host will be stubbed inside the WSL 2 as libcuda.so, therefore **users must not install any NVIDIA GPU Linux driver within WSL 2.** WSL-Ubuntu CUDA toolkit installer needs to be installed without overwriting the NVIDIA driver that was already mapped into the WSL 2 environment.

First, remove the old GPG key:

sudo apt-key del 7fa2af80

Then, Installation of Linux x86 CUDA Toolkit using WSL-Ubuntu Package

Installation Instructions ([download page for WSL-Ubuntu](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=WSL-Ubuntu&target_version=2.0&target_type=deb_local)):

wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86\_64/cuda-wsl-ubuntu.pin

sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600

wget https://developer.download.nvidia.com/compute/cuda/12.4.0/local\_installers/cuda-repo-wsl-ubuntu-12-4-local\_12.4.0-1\_amd64.deb

sudo dpkg -i cuda-repo-wsl-ubuntu-12-4-local\_12.4.0-1\_amd64.deb

sudo cp /var/cuda-repo-wsl-ubuntu-12-4-local/cuda-\*-keyring.gpg /usr/share/keyrings/

sudo apt-get update

sudo apt-get -y install cuda-toolkit-12-4

## Reference

<https://docs.nvidia.com/cuda/wsl-user-guide/index.html#nvidia-compute-software-support-on-wsl-2>

**Install Docker: Install using the apt repository**

Set up Docker's apt repository.

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

Install the Docker packages.

To install a specific version of Docker Engine, start by listing the available versions in the repository:

# List the available versions:

apt-cache madison docker-ce | awk '{ print $3 }'

5:24.0.0-1~ubuntu.22.04~jammy

5:23.0.6-1~ubuntu.22.04~jammy

...

Select the desired version and install:

VERSION\_STRING=5:24.0.0-1~ubuntu.22.04~jammy

sudo apt-get install docker-ce=$VERSION\_STRING docker-ce-cli=$VERSION\_STRING containerd.io docker-buildx-plugin docker-compose-plugin

Verify that the Docker Engine installation is successful by running the hello-world image.

sudo docker run hello-world

## Reference

<https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>

[**Install the NVIDIA Container Toolkit**](https://docs.nvidia.com/ai-enterprise/deployment-guide-vmware/0.1.0/docker.html#installing-the-nvidia-container-toolkit)**: Installing with Apt**

Configure the production repository:

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

Optionally, configure the repository to use experimental packages:

Sudo sed -i -e '/experimental/ s/^#//g' /etc/apt/sources.list.d/nvidia-container-toolkit.list

Update the packages list from the repository:

sudo apt-get update

Install the NVIDIA Container Toolkit packages:

sudo apt-get install -y nvidia-container-toolkit

**Configuration with Docker**

Configure the container runtime by using the nvidia-ctk command:

sudo nvidia-ctk runtime configure --runtime=docker

Restart the Docker daemon:

sudo systemctl restart docker

You may see error below:

A screen shot of a computer screen

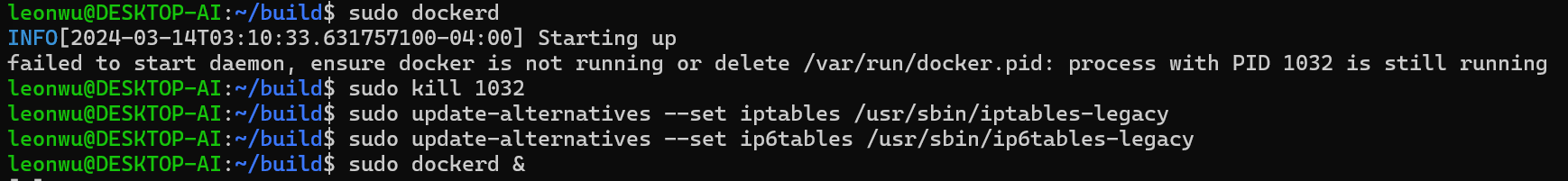
Description automatically generated

Or:

[Cannot connect to the Docker daemon at unix:/var/run/docker.sock. Is the docker daemon running?](https://stackoverflow.com/questions/44678725/cannot-connect-to-the-docker-daemon-at-unix-var-run-docker-sock-is-the-docker)

Solution:

Run: systemctl start docker. If not working, Run: sudo dockerd



Expected behavior:

dockerd should end up at the state API listen on /var/run/docker.sock

Actual behavior: dockerd throws the following error:

failed to start daemon: Error initializing network controller: error obtaining controller instance: unable to add return rule in DOCKER-ISOLATION-STAGE-1 chain: (iptables failed: iptables --wait -A DOCKER-ISOLATION-STAGE-1 -j RETURN: iptables v1.8.7 (nf\_tables): RULE\_APPEND failed (No such file or directory): rule in chain DOCKER-ISOLATION-STAGE-1

(exit status 4))

For wsl2 ubuntu 22.04, solution is run below:

sudo update-alternatives --set iptables /usr/sbin/iptables-legacy

sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy

sudo dockerd &

## Reference

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#configuration>

<https://github.com/microsoft/WSL/issues/6655>

### **Running a Sample Workload with Docker**

After you install and configure the toolkit and install an NVIDIA GPU Driver, you can verify your installation by running a sample workload (similar as nvidia-smi).

sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi

Your output should resemble the following output:

A screenshot of a computer

Description automatically generated

## Reference

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/sample-workload.html#running-a-sample-workload>

<https://docs.nvidia.com/ai-enterprise/deployment-guide-vmware/0.1.0/docker.html#testing-docker-and-nvidia-container-runtime>

At this point, GPU configuration for containerized deployment of NVIDIA FLARE with Docker in Windows system is completed. You can run a quick-start sample on NVFLARE at below linkage using GPU.

<https://nvflare.readthedocs.io/en/main/getting_started.html#containerized-deployment-with-docker>

### **Fix the nvflare dashboard –start issue**

Updated on April 28, 2024 to fix the nvflare dashboard –start issue:

1. **Uninstall Docker CLI**

sudo apt-get remove --purge docker docker-engine docker.io containerd runc

sudo apt-get purge docker-ce docker-ce-cli containerd.io docker-compose-plugin

sudo apt-get autoremove

1. **Remove Docker Files**

sudo rm -rf /var/lib/docker

sudo rm -rf /var/lib/containerd

sudo rm -rf /usr/bin/docker

sudo rm -rf /usr/libexec/docker

which docker

which dockerd

sudo rm $(which docker)

sudo rm $(which dockerd)

sudo systemctl disable docker

sudo rm /etc/systemd/system/docker.service

sudo rm /etc/init.d/docker

sudo systemctl daemon-reload

docker --version

dockerd –version

echo $PATH

env | grep -i docker

remove any Docker-related entries if found.

1. **Reinstall Docker (**Ubuntu):

sudo apt-get update

sudo apt-get install docker.io

start the docker:

sudo dockerd (if error exits, run step 4 otherwise skip step 4)

1. **Create a systemd service file for Docker (for running** sudo systemctl start docker. Not necessary as if sudo dockerd works without error**)**

(1) You can manually create a systemd service file for Docker. Open a text editor with sudo to create the **docker.service** file under **/etc/systemd/system/**.

sudo nano /etc/systemd/system/docker.service

(2) Add the following configuration to the file:

[Unit]

Description=Docker Application Container Engine

Documentation=https://docs.docker.com

After=network.target firewalld.service

Wants=network.target

[Service]

Type=notify

# the default is not to use systemd for cgroups because the delegate issues still

# exists and systemd currently does not support the cgroup feature set required

# for containers run by docker

ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

ExecReload=/bin/kill -s HUP $MAINPID

TimeoutSec=0

RestartSec=2

Restart=always

# Note that StartLimit\* options were moved from "Service" to "Unit" in systemd 229.

# Both the old, and new location are accepted by systemd 230 and up, so using the old location

# to make them work for either version of systemd.

StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.

# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make

# this work for either version of systemd.

StartLimitInterval=60s

# Having non-zero Limit\*s causes performance problems due to accounting overhead

# in the kernel. We recommend using cgroups to do container-local accounting.

LimitNOFILE=infinity

LimitNPROC=infinity

LimitCORE=infinity

# Uncomment TasksMax if your systemd version supports it.

# Only systemd 226 and above support this version.

#TasksMax=infinity

[Install]

WantedBy=multi-user.target

Save and close the file (in nano, press Ctrl+O, Enter, and then Ctrl+X to exit).

(3) Reload the systemd manager configuration

sudo systemctl daemon-reload

(4) Start and enable Docker service

sudo systemctl start docker

sudo systemctl enable docker

(5) Check the status of the Docker service

sudo systemctl status docker

docker run hello-world

1. **Ensure Docker Socket is Accessible**

docker run -v /var/run/docker.sock:/var/run/docker.sock ...

1. **Run the Container with Appropriate Permissions**

docker run --privileged ...

combine. Example:

sudo docker run --privileged --rm -it --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -v "$(pwd -P)/my-workspace:/workspace/my-workspace" -v /var/run/docker.sock:/var/run/docker.sock -w /workspace/my-workspace nvflare-pt:latest