Using PLEXIL with WSL 2 Install Instructions

Optional - Install “Windows Terminal” from the Microsoft Store. It’s a useful terminal window that is able to use Windows Powershell, all your Linux distros, Windows CMD, etc., all at the same time.

Installing WSL 2:

1. Press the Windows key, search “Turn Windows Features on or off,” and select the option.

2. Check the boxes next to “Virtual Machine Platform” and “Windows Subsystem for Linux.” If you plan on using docker, select the “Windows Hypervisor Platform” box as well. Press “Ok” and follow instructions. You likely need to reboot.

3. You need to be using Windows version 2004, build 19041 or higher. This will most likely require you to check for updates at the very least and update your OS.
   NOTE: It is possible you will need to join the Windows Insider Program to access all of the required updates. Join the “Release Preview” ring.
   If you have an Nvidia GPU: If you have an Nvidia GPU, it’s possible for WSL 2 to have GPU acceleration while using docker. You need to join the Windows Insider Program and select the “Fast” ring.
   After Joining Insider Program: Go to settings > Update & Security > Windows Update. Check for Updates.
   Download and install updates.
   Reboot.

4. Download and install the Linux kernel update package. (Only needs to be done the first time. Kernel updates will come through Windows updates.) Download and follow installer instructions.
   https://docs.microsoft.com/en-us/windows/wsl/wsl2-kernel

5. Open Powershell and run this command to set your default WSL version to 2.

   ```
   wsl --set-default-version 2
   ```

6. Open the Microsoft Store and install “Ubuntu 18.04 LTS.”

7. Open Ubuntu once it’s installed for your first time setup. Set your username and password.

8. Open Windows Powershell and run

```
wsl --list --verbose
```

9. If your distribution says it's using WSL version 1, run this command to convert it to version 2.

   ```
   wsl --set-version <distribution name> <versionNumber>
   ```

10. If you have any issues, refer to https://docs.microsoft.com/en-us/windows/wsl/install-win10

Setting up a GUI and desktop environment on WSL 2:
(Optional but recommended for some features)

1. Open Windows Powershell and run

   ```
   ipconfig
   ```

   Look under "Ethernet adapter vEthernet (WSL):" and copy the "IPv4 Address."

   Open your ".bashrc and set your "DISPLAY" environment variable.

   ```
   cd ~
   ```

   ```
   nano .bashrc
   ```

   (Place at the bottom of your .bashrc file, save and exit)
   ```
   export DISPLAY=<COPIED IPv4 ADDRESS>:0
   (looks something like DISPLAY=123.45.678.9:0)
   ```
   NOTE: This will need to be done every time you restart your computer before opening the GUI.

   Now source your .bashrc file.

   ```
   source ~/.bashrc
   ```

2. Next download and install VcXsrv. https://sourceforge.net/projects/vcxsrv/
   Follow the steps in the installer.
3. When prompted, allow VcXsrv on both private and public networks.
   - If you weren't prompted, you will need to open your Start menu and search "Allow an

app through Windows Firewall." Open it up and scroll down to "VcXsrv windows xserver" and check the boxes under "Private" and "Public." Press OK and exit.

4. Now install your chosen desktop. Xfce and KDE (Kubuntu) have been tested to work successfully.

   Xfce:

   ```
   sudo apt-get update && sudo apt-get upgrade -y

   sudo apt-get install xfce4
   ```

   KDE:

   ```
   sudo apt-get update && sudo apt-get upgrade -y

   sudo add-apt-repository ppa:kubuntu-ppa/backports

   sudo apt-get install kubuntu-desktop
   ```

   If you'd like to try another desktop environment, here's a reference.
   https://www.linuxtrainingacademy.com/install-desktop-on-ubuntu-server/

5. Open your start menu and select XLaunch.
   In the first window select "One Large Window." Click Next.

   In the second window select "Start no client." Click Next.

   In the third window check boxes "Clipboard," "Primary Selection," Native opengl," and "Disable access control." (This should be all options) Click Next.

   You can now save the configuration to launch quickly in the future or click "Finish" to open the VcXsrv window.

6. Reopen your Ubuntu shell and run the command to open your desktop environment.

   Xfce:

   ```
   startxfce4
   ```

   KDE:

   ```
   startkde
   ```

NOTE: If this failed try,

```
sudo apt update && sudo apt upgrade
```

and try to start your desktop again.

7. In Xfce you can install Firefox by running "`sudo apt-get install firefox`" since the default browser doesn't work. KDE already has Firefox installed.

   Reference:
   https://autoize.com/xfce4-desktop-environment-and-x-server-for-ubuntu-on-wsl-2/

(Optional) Setting up Docker with WSL 2:

1. Download Docker Desktop Stable 2.3.0.2 or later.
   https://hub.docker.com/editions/community/docker-ce-desktop-windows/
2. Follow installation instructions. If prompted, enable WSL 2 during installation (This might be selected by default for you.)
3. Start Docker Desktop from the Start menu.
   Go to Settings > General.
   Check the box next to "Use the WSL 2 based engine."
   Click "Apply & Restart"
   Go to Settings > Resources > WSL INTEGRATION
   Check the box next to "Enable integration with my default WSL distro."
   Switch on the "Ubuntu-18.04" option.
   Click "Apply & Restart"
4. You can now use Docker with both Windows and WSL.

   Reference:
   https://docs.docker.com/docker-for-windows/wsl/

If you want to use WSL 2 with Visual Studio Code and Docker:
https://code.visualstudio.com/blogs/2020/03/02/docker-in-wsl2
NOTE: Visual Studio Code might not open in a remote window when using the command "`code .`". It should say something like "WSL: Ubuntu" in the bottom left corner of the window. If it doesn't, you can click the bottom left corner (or press F1) and select "Remote-WSL: Reopen Folder in WSL." Then select Ubuntu and it will open correctly.

(Optional) Setting up WSL 2 with an Nvidia GPU:

If you have an Nvidia GPU, WSL 2 is able to have GPU acceleration when running docker containers with Ubuntu.

1. Make sure you have selected the “Fast” ring in the Windows Insider Program. Check for Windows Updates and Install/Reboot as necessary.

2. Open an Ubuntu terminal and run

   ```
   uname -r
   ```

   Your kernel should be at least 4.19.121
   If it is below that version, you can try to run this in Powershell

   ```
   wsl.exe --update
   ```

   Check your kernel version again. If that didn’t work, go to Settings > Update & Security > Windows Update < Advanced options. Turn on the option “Receive updates for other Microsoft products when you update Windows.” Check for updates again, install, and reboot.

3. Join the Nvidia Developer Program to get early access to the Drivers.
   https://developer.nvidia.com/developer-program

   Download and install the appropriate Nvidia Drivers.
   https://developer.nvidia.com/cuda/wsl/download

   If you want to read more about CUDA on WSL
   https://developer.nvidia.com/blog/announcing-cuda-on-windows-subsystem-for-linux-2/

4. **NOTE:** This step might not be necessary if you have Docker Desktop for Windows. However, Docker Desktop failed much of the time for me, and following this step fixed my issues.

   First quit Docker Desktop for Windows.

   Install Docker in WSL

   ```
   sudo apt -y install docker.io
   ```

5. Install Nvidia Container Toolkit
   “Set the distribution variable, import the Nvidia repository GPG key, and then add the Nvidia repositories to the Ubuntu apt package manager:”

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo
apt-key add -

curl -s -L https://nvidia.github.io/nvidia-
docker/$distribution/nvidia-docker.list | sudo tee
/etc/apt/sources.list.d/nvidia-docker.list

curl -s -L https://nvidia.github.io/libnvidia-
container/experimental/$distribution/libnvidia-container-
experimental.list | sudo tee /etc/apt/sources.list.d/libnvidia-
container-experimental.list
```

6. Refresh the Ubuntu apt repositories and install the Nvidia runtime.

```
sudo apt update && sudo apt install -y nvidia-docker2
```

7. Close all Ubuntu Terminals and open Powershell. Manually shutdown Ubuntu.

```
wsl.exe --shutdown Ubuntu
```

8. Open Ubuntu and start Docker

```
sudo service docker stop
sudo service docker start
```

9. Run this command to test GPU Compute

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

If everything is configured correctly, you should have an output somewhat like this.

```
Unable to find image 'nvcr.io/nvidia/k8s/cuda-sample:nbody' locally
nbody: Pulling from nvidia/k8s/cuda-sample
22dc81ace0ea: Pull complete
1a8b3c87dba3: Pull complete
91390a1c435a: Pull complete
07844b14977e: Pull complete
b78396653dae: Pull complete
95e837069dfa: Pull complete
fef4aadda783: Pull complete
```

```
343234bd5cf3: Pull complete
d1e57bfda6f0: Pull complete
c67b413dfc79: Pull complete
529d6d22ae9f: Pull complete
d3a7632db2b3: Pull complete
4a28a573fcc2: Pull complete
71a88f11fc6a: Pull complete
11019d591d86: Pull complete
10f906646436: Pull complete
9b617b771963: Pull complete
6515364916d7: Pull complete
Digest:
sha256:aaca690913e7c35073df08519f437fa32d4df59a89ef1e012360fbec46524ec8
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
MapSMtoCores for SM 7.5 is undefined.  Default to use 64 Cores/SM
GPU Device 0: "GeForce RTX 2070 with Max-Q Design" with compute capability 7.5

> Compute 7.5 CUDA device: [GeForce RTX 2070 with Max-Q Design]
36864 bodies, total time for 10 iterations: 75.215 ms
= 180.675 billion interactions per second
= 3613.504 single-precision GFLOP/s at 20 flops per interaction
```

NOTE: If these steps do not work try this.

1. Stop Docker Desktop.

2. Go back and perform step 4.

3. Exit all Ubuntu Terminals and shutdown wsl with Powershell for good measure.

```
wsl.exe --shutdown
```

4. Reopen an Ubuntu Terminal.

5. Then start the docker daemon in WSL.

```
sudo dockerd
```

6. Open another Ubuntu terminal.

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

7. Check to see if output looks like the example.

Resource:
https://ubuntu.com/blog/getting-started-with-cuda-on-ubuntu-on-wsl-2

You can now install PLEXIL like normal.

1. Install git if you don't already have it.

   ```
   sudo apt install git
   ```

2. Clone the PLEXIL repository.
   ```
   git clone https://git.code.sf.net/p/plexil/git plexil
   ```

3. Install your dependencies if not already installed.

   You can use

   ```
   sudo apt install <package-name>
   ```

   ```
   - make
   - autotools-dev
   ```

```
- autoconf
- libtool
- g++
- ant
- gperf
- openjdk-8-jdk
- freeglut3-dev
```

4. Add these to your .bashrc file. Make sure to change the first line to be the location of plexil.

```
export PLEXIL_HOME=/home/joe/plexil
export PATH=$PLEXIL_HOME/scripts:$PATH
source $PLEXIL_HOME/scripts/plexil-setup.sh
```

5. Source your .bashrc file.

```
source ~/.bashrc
```

6. Make and configure.

```
cd $PLEXIL_HOME

make src/configure

cd src
```

This is a good configuration for most uses.

```
./configure --prefix=$PLEXIL_HOME --enable-ipc --enable-module-tests --
enable-viewer --enable-sas --enable-test-exec --enable-udp
```

7. Build PLEXIL!

```
cd $PLEXIL_HOME

make
```