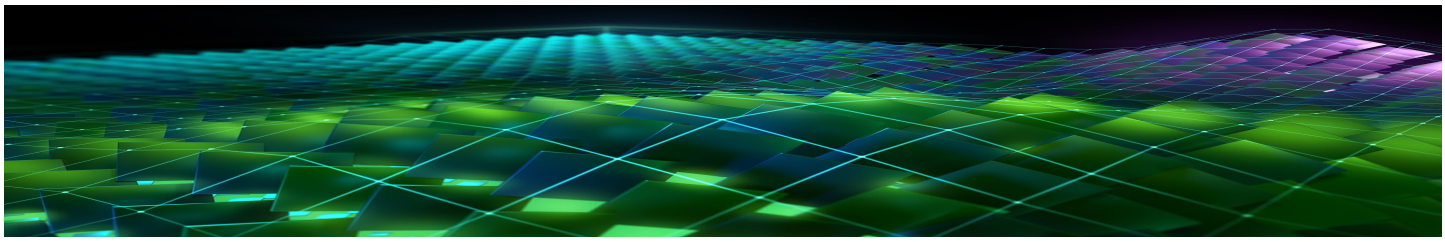


Getting Started with NVIDIA's AI Platform on GeForce RTX PCs

Whether you are a new student curious about artificial intelligence, or already far along in a degree program, this is a great place to start to set up your own PC to learn and develop for AI. In this section we will describe the high-level concepts associated with setting up your system, provide step-by-step instructions for the basic configuration, and provide links to tutorials and materials that build off of this with detailed information about topics such as deep learning, data analytics, machine learning, graph analytics, and many others.

Minimum System Requirements

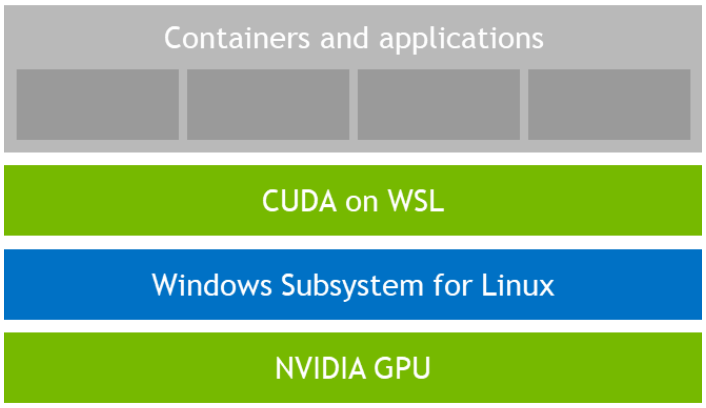
First, please review the minimum requirements:

- Minimum operating system version: Windows 11 SV2
- Graphics processor: Any NVIDIA RTX GPU (e.g. GeForce RTX 3060, RTX A5000). Note that many deep learning workloads have high GPU memory requirements. We recommend a GPU with at least 6 GB of GPU memory.

It may be possible to use these resources with systems that do not meet these requirements, but it is not recommended and is outside the scope of this document.

High-level Concepts

Before we get into the steps to get started, we need to introduce a few important concepts. A high level understanding of the components that make NVIDIA's AI platform will make the instructions easier to follow. Please review the diagram below that illustrates the software stack we will be setting up.



Windows Subsystem for Linux 2 (WSL2): Microsoft's WSL2 makes it possible to run software and applications that were written for the Linux operating system on Microsoft Windows, including NVIDIA's AI stack.

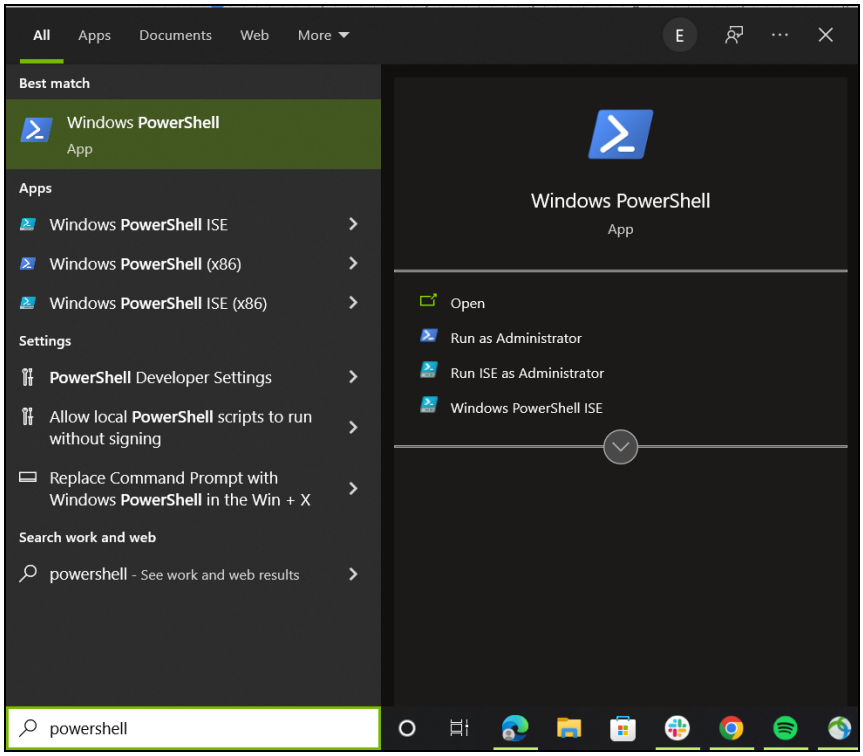
CUDA on WSL: NVIDIA CUDA is the preferred software platform for developing accelerated programs for the GPU and includes the runtime, drivers, and libraries. AI and ML are very computationally intensive, and relies on GPU acceleration to run in a reasonable amount of time. CUDA on WSL is a version of CUDA that runs in WSL, and allows programs running in WSL to leverage the GPU.

Docker Containers: Containers are a way of managing system configurations. Getting all of the right pieces of software setup correctly to run a specific AI application can be challenging. A container is essentially a self-contained environment that has everything set up correctly for a specific use case and makes it significantly easier to get going. There are containers for using the deep learning frameworks, like TensorFlow, and there are containers for using NVIDIA RAPIDS for machine learning, data analytics, and graphic analytics, and many others.

Instructions to install WSL2

The first step is to install WSL2. WSL2 is the software layer that allows you to run programs developed for Linux directly in Windows.

- For the rest of this guide, we will use Windows PowerShell to run commands inside WSL. Let's start by starting PowerShell using the *Run as Administrator* option in Windows Start menu:



- Type the following command inside the PowerShell window:

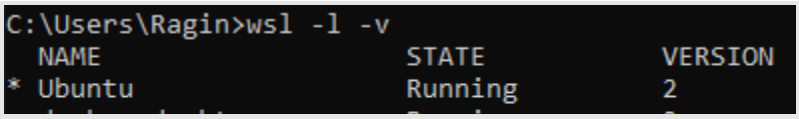
```
wsl --install
```

```
PS C:\Windows\system32> wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Downloading: WSL Kernel
Installing: WSL Kernel
WSL Kernel has been installed.
Downloading: Ubuntu
The requested operation is successful. Changes will not be effective until the system is rebooted.
```

- Next, reboot your computer. After logging back into Windows, you should see an Ubuntu command-line window asking to set up user credentials for your WSL Ubuntu. In this window, enter your desired username at ‘Enter new UNIX username’ and then the password. Remember these credentials as you will need them while using WSL.

- Start PowerShell as administrator and verify WSL Installation by typing the following at the prompt:

```
wsl -l -v
```



If the version says 2, then you're good to go. Otherwise follow the instructions here to upgrade to WSL2.

Installing Docker and NVIDIA Container Toolkit

- Start a WSL2 session inside the Powershell window using:

```
wsl
```

Note: If you get an error stating the referenced object type does not support the attempted operation, run this command in the Windows PowerShell and restart WSL.

```
netsh  winsock reset
```

- Inside this Powershell window, first install Docker using this command:

```
curl https://get.docker.com | sh && sudo service docker start
```

- Next, setup the package repository and the GPG key:

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
      && 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/$distribution/libnvidia-contain
er.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
```

- Finally, install nvidia-docker2 using the following commands:

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

- Restart docker for the changes to take effect:

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

Note: Often, after restarting your computer, you might encounter the following error running Docker commands inside WSL:

```
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the
docker daemon running?
```

To resolve this, run the following command to start the Docker service in WSL:

```
sudo service docker start
```

Note: If you get an error stating /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link, complete the steps below.

Inside WSL

```
exit
```

Outside WSL and in the Windows Command Prompt (Run as Admin)

```
cd C:\Windows\System32\lxss\lib
rm libcuda.so
rm libcuda.so.1
wsl -e /bin/bash
```

Inside WSL

```
ln -s libcuda.so.1.1 libcuda.so.1 && ln -s libcuda.so.1.1 libcuda.so
```

Restart docker for the changes to take effect:

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

- Validate the installation of *nvidia-docker2* using the CUDA 11.0 base docker image:

```
sudo docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04
nvidia-smi
```

This should result in a console output shown below:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0 |
| N/A   34C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

- Next, sign up for an account on NVIDIA GPU Cloud (NGC). Follow the instructions here to generate an API key. Store this API key in a secure location because you will need it to fetch the containers.

- Login to NGC's Docker container repository:

```
sudo docker login nvcr.io
```

When prompted enter the following credentials:

```
username - $oauthtoken
password - your_NGC_API_key
```

Pro-tip: By default, Docker commands need to be run as root user (sudo docker ..). You can also use Docker without sudo by performing the following commands:

1.) Create a docker group and add your current user to it

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

2) Verify you can run docker without sudo:

```
docker run hello-world
```

Getting Started with GPU-accelerated TensorFlow

Next we're going to install the TensorFlow container from NGC. NGC is a catalog of downloadable containers, models, and samples related to AI and high-performance computing. TensorFlow is a software framework for building, training, and running deep learning models.

- Download NGC Containers for Tensorflow from NGC using this command:

```
sudo docker pull nvcr.io/nvidia/tensorflow:22.10-tf2-py3
```

- Start the container using the following command:

```
sudo docker run --gpus all -it --rm nvcr.io/nvidia/tensorflow:22.10-tf2-py3
```

- Test your Tensorflow container setup.
  In the following steps, we validate that our Tensorflow Docker environment is ready for development:

  - First, check that you can see the GPUs inside of Tensorflow. In the Docker command prompt, start python and run the following commands to verify TensorFlow can access the GPU:

```
$ python
>>> import tensorflow as tf

# If tf1
>>> print(tf.test.is_gpu_available())
True

# If tf2
>>> tf.config.list_physical_devices("GPU").__len__() > 0
True

>>> exit
```

- Check that you have network access inside the container.
  Run the following command (ping www.nvidia.com) inside your Docker container. The ping command is commonly used to verify that your computer can reach addresses on the Internet. Ideally, the ping command should start printing success messages that look like this:

```
$ ping www.nvidia.com
PING e33907.a.akamaiedge.net (23.50.233.61): 56 data bytes
64 bytes from 23.50.233.61: icmp_seq=0 ttl=51 time=64.809 ms
64 bytes from 23.50.233.61: icmp_seq=1 ttl=51 time=30.998 ms
```

Instead of the above, if you receive error messages that look like the following, proceed to the Troubleshooting network issues section below.

```
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
```

- To start the TensorFlow Docker container with a JupyterLab server, start the container with the following command:

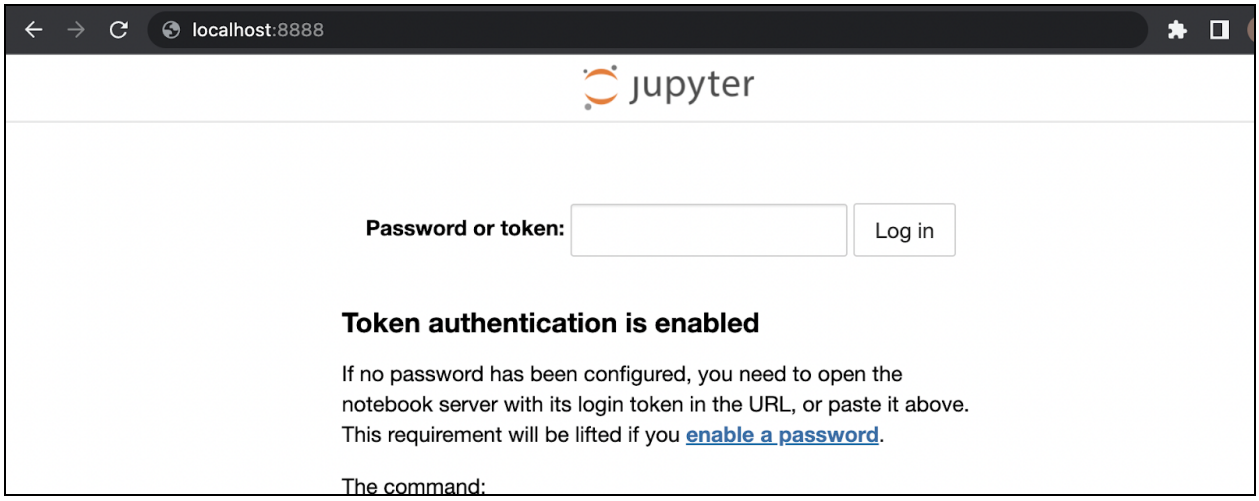
```
sudo docker run --gpus all -it --rm -p 8888:8888
nvcr.io/nvidia/tensorflow:22.10-tf2-py3
```

- Once inside the container, start a JupyterLab server using:

```
jupyter-lab
```

The above command starts a JupyterLab web server on the computer that can be accessed using a web browser by navigating to http://localhost:8888 . The server is protected by a token that can be found in the jupyter-lab logs inside the TensorFlow container. The console logs below show the location of the token. Copy the token from your container into the Jupyter web page, and click on 'Log In' to access the JupyterLab Server.

```
[I 22:29:25.799 LabApp] Jupyter Notebook 6.4.10 is running at:
[I 22:29:25.799 LabApp] http://hostname:8888/?token=<your_jupyter_token>
```



Troubleshooting network issues inside WSL Docker containers

If you are unable to access the Internet from inside WSL Docker containers, it is likely that you are facing nameserver issues. To resolve this, we recommend changing your WSL settings to use the Google DNS instead of your default (ISP) DNS. To do this, follow these steps:

- If you are inside a Docker container, make sure to exit it using the exit command:

```
$ exit
```

- Inside your WSL terminal, open the /etc/resolv.conf file using your favorite text editor. Here, we use nano. For example:

```
$ sudo nano /etc/resolv.conf
```

Here, update the nameserver IP address from your default to 8.8.8.8. After the update your /etc/resolv.conf file should look like this:

```
nameserver 8.8.8.8
```

Save the updated file by pressing Ctrl+X keys and following the on-screen prompt to overwrite the file.

- Next, we restart WSL. To do this, start a new PowerShell window as administrator and enter the following command at the prompt:

```
$ wsl --shutdown
```

This should shut down the WSL instance. Start a new one typing the following command inside the same terminal:

```
$ wsl
```

Validate that you can now access the Internet from inside Docker containers. Start Docker and launch the Tensorflow container using the following commands:

```
$ sudo service docker start

$ sudo docker run --gpus all -it --rm
nvcr.io/nvidia/tensorflow:22.07-tf2-py

$ ping www.nvidia.com
```

The ping command should now be able to get responses from nvidia.com.

Next Steps

In this article, we showed you how to set up your own Windows 11 PC powered by RTX to learn and develop GPU-accelerated AI. Below are additional resources to continue your journey.

- Learning Deep Learning - Part of the NVIDIA Deep Learning Institute (DLI), Magnus Ekman's foundational book takes you from basic techniques to applications of deep learning, one of the most exciting techniques in modern AI,. The book is available in hard copy or electronic format.
- Learning RAPIDS - This page provides additional information on using RAPIDS in WSL, and links to additional resources and tools.
- CUDA on WSL Detailed Documentation - Want to delve deeper into the details of CUDA on WSL? These documents provide additional technical information as well as instructions for alternative configurations.

You can also get access to learning resources on a wide range of GPU-related topics at NVIDIA On-Demand.