



CUDA on WSL User Guide

User Guide

Table of Contents

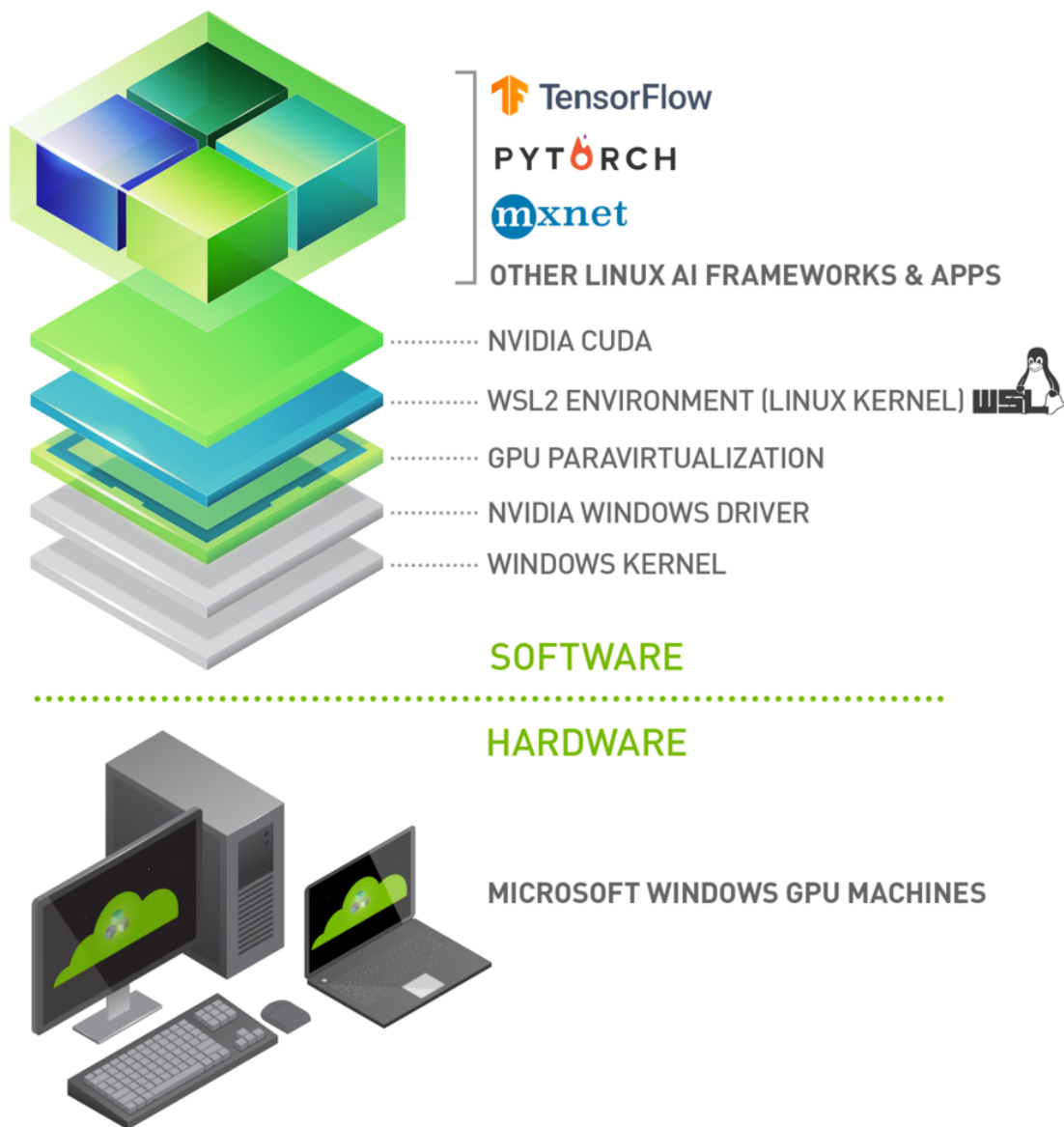
Chapter 1. Introduction.....	1
Chapter 2. Getting Started.....	3
2.1. WSL 2 System Requirements.....	3
2.2. Installing Microsoft Windows Insider Preview OS Builds (only for Windows 10).....	3
2.3. Installing NVIDIA Windows Drivers for CUDA, DirectX, and DirectML Support.....	4
2.4. Installing WSL 2 and Setting up a Linux Development Environment.....	4
2.5. Running CUDA Applications.....	5
2.6. Setting up CUDA Toolkit on WSL 2.....	5
2.7. Building CUDA Samples.....	6
2.8. Running CUDA Containers.....	6
2.8.1. Setup.....	6
2.8.2. Install Docker.....	6
2.8.3. Install NVIDIA Container Toolkit.....	7
2.8.4. Running Simple CUDA Containers.....	7
2.8.5. Jupyter Notebooks.....	8
2.8.6. Deep Learning Framework Containers.....	10
Chapter 3. NVIDIA Compute Software Support Matrix for WSL 2.....	11
Chapter 4. Known Limitations for Linux CUDA Applications.....	12
Chapter 5. Features Not Yet Supported.....	13
Chapter 6. Troubleshooting.....	14
6.1. Container Runtime Initialization Errors.....	14
6.2. Checking WSL Kernel Version.....	15
Chapter 7. Release Notes.....	16
7.1. 510.06.....	16
7.1.1. Changelog.....	16
7.1.2. New Features.....	16
7.1.3. Resolved Issues.....	17
7.1.4. Known Limitations.....	17
7.1.5. Known Issues.....	17
7.2. 471.21.....	17
7.3. 470.76.....	18
7.4. 470.14.....	19
7.5. 465.42.....	19
7.6. 465.21.....	19

7.7. 465.12..... 20
7.8. 460.20..... 20
7.9. 460.15..... 20
7.10. 455.41..... 20
7.11. 455.38..... 20

Chapter 1. Introduction

Windows Subsystem for Linux (WSL) is a Windows 10 feature that enables users to run native Linux command-line tools directly on Windows. WSL is a containerized environment within which users can run Linux native applications from the command line of the Windows 10 shell without requiring the complexity of a dual boot environment. Internally, WSL is tightly integrated with the Microsoft Windows operating system, which allows it to run Linux applications alongside traditional Windows desktop and modern store apps.

Figure 1. CUDA on WSL Overview



With WSL 2 and GPU paravirtualization technology, Microsoft enables developers to run GPU accelerated applications on Windows. NVIDIA GPU acceleration will be available with official support on Pascal and later GeForce and Quadro GPUs in Windows Device Driver Model. However, we recommend using Turing or newer architectures for better performance.

The following document describes a workflow for getting started with running CUDA applications or containers in a WSL 2 environment.

Chapter 2. Getting Started

Getting started with running CUDA on WSL requires you to complete these steps in order:

1. [WSL 2 System Requirements](#)
2. (Windows 10 only) [Installing Microsoft Windows Insider Preview OS Builds \(only for Windows 10\)](#)
3. [Installing NVIDIA Windows Drivers for CUDA, DirectX, and DirectML Support](#)
4. [Installing WSL 2 and Setting up a Linux Development Environment](#)

2.1. WSL 2 System Requirements

- ▶ WSL 2 GPU acceleration will be available on Pascal and later GPU architecture on both GeForce and Quadro product SKUs in WDDM mode. It will not be available on Quadro GPUs in TCC mode or Tesla GPUs yet.
- ▶ Ensure you are on the latest WSL Kernel or at least 4.19.121+. Once again we recommend 5.10.16.3 or later for better performance and functional fixes.
- ▶ If you are on Windows 11, you no longer need to be on Windows Insider Program to use WSL. Refer to Windows11 [system requirements](#).
- ▶ If you are continuing to use Windows 10, see [Installing Microsoft Windows Insider Preview OS Builds \(only for Windows 10\)](#).

2.2. Installing Microsoft Windows Insider Preview OS Builds (only for Windows 10)

- ▶ If you are on Windows 11 please skip this section.
- ▶ WSL 2 is available on Windows10 for Windows Insider Program members. If you are going to be on Windows 10 and would like to install WSL 2, register for the [Windows Insider Program](#) and choose Release Preview Channel as your [flighting channel](#) (previously fast rings) and get the latest build. You can check your build version number by running `winner` via the Run command.

- ▶ Learn more on [Releasing Windows 10 Build 19043.1263 \(21H1\) to Release Preview Channel](#).

2.3. Installing NVIDIA Windows Drivers for CUDA, DirectX, and DirectML Support

- ▶ Download the NVIDIA Windows Driver from the download section on the [CUDA on WSL](#) page. Choose the appropriate driver depending on the type of NVIDIA GPU in your system - GeForce and Quadro.
- ▶ Install the driver using the executable on the Windows machine. The Windows Display Driver will install both the regular driver components for native Windows and for WSL support. **This is the only driver you need to install.**

CUDA support enables GPU accelerated computing for data science, machine learning and inference solutions. DirectX support enables graphics on WSL 2 by supporting DX12 APIs. TensorFlow with DirectML support on WSL 2 will get NV GPU hardware acceleration for training and inference workloads.

For some helpful examples, see <https://docs.microsoft.com/en-us/windows/win32/direct3d12/gpu-tensorflow-wsl>.



Note: This is the only driver you need to install. Do not install any Linux display driver in WSL. CUDA, DirectML and DirectX support will be available with this driver. No additional drivers are needed for WSL 2.

2.4. Installing WSL 2 and Setting up a Linux Development Environment

1. Install WSL 2 by following the instructions in the Microsoft documentation available [here](#).
2. Install your preferred Linux distro from the Microsoft Driver Store.
3. For basic WSL 2 commands, review the documentation at <https://docs.microsoft.com/en-us/windows/wsl/basic-commands>.
4. Ensure you have the latest WSL kernel by running `wsl --update`, then launch WSL 2 from your preferred Windows terminal:

```
C:\> wsl.exe --update
C:\> wsl.exe
```


2.5. Running CUDA Applications

Once you are on your favorite Linux distro through WSL 2, and after the NVIDIA Windows driver is installed there is nothing more to do to run existing CUDA applications that were built on Linux. Just run your CUDA app as you would run it under Linux!

CUDA support on WSL 2 allows you to run existing GPU accelerated Linux applications or containers almost seamlessly (see limitations here). But in order to build new applications CUDA Toolkit is needed.

2.6. Setting up CUDA Toolkit on WSL 2

In order to compile a CUDA application on WSL 2, you will have to install only the CUDA toolkit. We recommend either of the following two options for installing the CUDA Toolkit.

Installation of CUDA Toolkit using WSL-Ubuntu Package

It is recommended to use the Linux package manager to install the CUDA for the Linux distributions supported under WSL 2. CUDA Toolkit available for Linux distributions can be used for WSL 2 as well, but these toolkits come packaged with the NVIDIA GPU Linux driver which must not be installed. So care must be taken not to override the WSL 2 native drivers already installed in the system.

In order to ease the installation and not to accidentally override the Windows native WSL 2 driver already installed in the system, we have created a separate package under the category WSL-Ubuntu.

Follow the sample instructions below to install the CUDA Toolkit from the WSL-Ubuntu package found [here](#) on Ubuntu.

```
$ 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/11.4.0/local_installers/cuda-repo-wsl-ubuntu-11-4-local_11.4.0-1_amd64.deb
$ sudo dpkg -i cuda-repo-wsl-ubuntu-11-4-local_11.4.0-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-wsl-ubuntu-11-4-local/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

Note that we are installing CUDA because the WSL-ubuntu package only has the CUDA toolkit.

Installation of CUDA Toolkit using Meta Package

If you installed the toolkit using the WSL-Ubuntu package, please skip this section. In case you prefer to install CUDA toolkit from our regular package, note that for WSL 2, you should use the `cuda-toolkit-<version>` meta-package to avoid installing the NVIDIA driver that is typically bundled with the toolkit.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
$ sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

```
$ wget https://developer.download.nvidia.com/compute/cuda/11.4.0/local_installers/
cuda-repo-ubuntu2004-11-4-local_11.4.0-470.42.01-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu2004-11-4-local_11.4.0-470.42.01-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
$ sudo apt-get update
```

Do not choose the “cuda”, “cuda-11-0”, or “cuda-drivers” meta-packages under WSL 2 if you are installing the regular CUDA toolkit as these packages will result in an attempt to install the Linux NVIDIA driver under WSL 2.

```
$ apt-get install -y cuda-toolkit-11-4
```

You can also install other components of the toolkit by choosing the right [meta-package](#).

2.7. Building CUDA Samples

Build the CUDA samples available obtained from github or the ones under `/usr/local/cuda/samples` from your installation of the CUDA Toolkit in the previous section. The BlackScholes application is located under `/usr/local/cuda/samples/4_Finance/BlackScholes`. Alternatively, as mentioned earlier you can transfer a binary built on Linux to WSL 2 and run directly.

```
C:\> wsl
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
$ cd /usr/local/cuda-11.4/samples/4_Finance/BlackScholes
$ make BlackScholes
$ ./BlackScholes

Initializing data...
...allocating CPU memory for options.
...allocating GPU memory for options.
...generating input data in CPU mem.
...copying input data to GPU mem.
Data init done.

Executing Black-Scholes GPU kernel (131072 iterations)...
Options count          : 8000000
BlackScholesGPU() time : 0.207633 msec
Effective memory bandwidth: 385.295561 GB/s
Gigaoptions per second : 38.529556
```

2.8. Running CUDA Containers

2.8.1. Setup

This chapter describes the workflow for setting up the NVIDIA Container Toolkit in preparation for running GPU accelerated containers.

2.8.2. Install Docker

Use the Docker installation script to install Docker for your choice of WSL 2 Linux distribution.

Note that NVIDIA Container Toolkit has not yet been validated with [Docker Desktop WSL 2](#) backend.



Note: For this release, install the standard Docker-CE for Linux distributions.

```
$ curl https://get.docker.com | sh
```

2.8.3. Install NVIDIA Container Toolkit

Install the NVIDIA Container Toolkit (previously known as `nvidia-docker2`). Official WSL 2 support is available starting with `nvidia-docker2` v2.6.0 and the underlying runtime library (`libnvidia-container` \geq 1.5.1).

For brevity, the installation instructions provided here are for Ubuntu.

Set up the `stable` repositories and the GPG key. The changes to the runtime to support WSL 2 are available in the `stable` repositories.

```
$ 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
```

Install the NVIDIA runtime packages (and their dependencies) after updating the package listing.

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

Open a separate WSL 2 window and start the Docker daemon again using the following commands to complete the installation.

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

In this section, we will walk through some examples of running containers in a WSL 2 environment.

2.8.4. Running Simple CUDA Containers

In this example, let's run an N-body simulation CUDA sample. This example has already been containerized and available from NGC.

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

From the console, you should see output as shown below.

```
$ docker run --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
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)
    -nembodies=<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: "Turing" with compute capability 7.5

> Compute 7.5 CUDA device: [NVIDIA GeForce RTX 2080]
47104 bodies, total time for 10 iterations: 68.445 ms
= 324.169 billion interactions per second
= 6483.371 single-precision GFLOP/s at 20 flops per interaction

```

2.8.5. Jupyter Notebooks

In this example, let's run a Jupyter notebook.

```
$ docker run -it --gpus all -p 8888:8888 tensorflow/tensorflow:latest-gpu-py3-jupyter
```

After the container starts, you should see that everything is ready:



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...
```

```

[I 04:00:11.167 NotebookApp] Writing notebook server cookie secret to /root/.local/
share/jupyter/runtime/notebook_cookie_secret
jupyter_http_over_ws extension initialized. Listening on /http_over_websocket
[I 04:00:11.447 NotebookApp] Serving notebooks from local directory: /tf
[I 04:00:11.447 NotebookApp] The Jupyter Notebook is running at:
[I 04:00:11.447 NotebookApp] http://72b6a6dfac02:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
[I 04:00:11.447 NotebookApp] or http://127.0.0.1:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
[I 04:00:11.447 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 04:00:11.451 NotebookApp]

```

```

To access the notebook, open this file in a browser:
file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
Or copy and paste one of these URLs:

```

```

http://72b6a6dfac02:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
or http://127.0.0.1:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b

```

After the URL is available from the console output, input the URL into your browser to start developing with the Jupyter notebook. Ensure that you replace `127.0.0.1` with `localhost` in the URL when connecting to the Jupyter notebook from the browser. You should be able to view the TensorFlow tutorial in your browser. Choose any of the tutorials for this example.

Navigate to the **Cell** menu and select the **Run All** item, then check the log within the Jupyter notebook WSL 2 container to see the work accelerated by the GPU of your Windows PC.

```

...
[I 16:00:00.150 NotebookApp] 302 GET /?
token=1fc498fd08ea697cd1a01b5061bcc1b381254eeb4f768d5d (172.17.0.1) 0.51ms
[I 16:00:07.509 NotebookApp] Writing notebook-signing key to /root/.local/share/
jupyter/notebook_secret
[W 16:00:07.511 NotebookApp] Notebook tensorflow-tutorials/classification.ipynb is
not trusted

[I 16:00:08.281 NotebookApp] Kernel started: 8fcadc98-b40e-41ee-8ce2-003afd444678
2021-07-07 16:00:14.343853: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libnvinfer.so.6
2021-07-07 16:00:14.357695: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libnvinfer_plugin.so.6
2021-07-07 16:00:25.573058: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2021-07-07 16:00:25.736913: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.

...
2021-07-07 16:00:26.149385: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
2021-07-07 16:00:27.309024: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096]
Device interconnect StreamExecutor with strength 1 edge matrix:
2021-07-07 16:00:27.309079: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102]
0
2021-07-07 16:00:27.309101: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115]
0: N
2021-07-07 16:00:27.310983: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.
2021-07-07 16:00:27.311180: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1324]
Could not identify NUMA node of platform GPU id 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2021-07-07 16:00:27.311599: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.
2021-07-07 16:00:27.311963: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6706
MB memory) -> physical GPU (device: 0, name: NVIDIA GeForce RTX 2080, pci bus id:
0000:65:00.0, compute capability: 7.5)
2021-07-07 16:00:30.450162: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
Allocation of 376320000 exceeds 10% of system memory.
2021-07-07 16:00:31.962248: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2021-07-07 16:01:15.050709: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
Allocation of 62720000 exceeds 10% of system memory.
2021-07-07 16:01:16.025014: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
Allocation of 62720000 exceeds 10% of system memory.

```

```
[I 16:02:09.048 NotebookApp] Saving file at /tensorflow-tutorials/
classification.ipynb
```

2.8.6. Deep Learning Framework Containers

Get started quickly with AI training using pre-trained models available from NVIDIA and the [NGC catalog](#). Follow the instructions in [this post](#) for more details.

As an example, let's run a TensorFlow container to do a ResNet-50 training run using GPUs using the 20.03 container from NGC. This is done by launching the container and then running the training script from the `nvidia-examples` directory.

```
$ docker run --gpus all -it --shm-size=1g --ulimit memlock=-1 --ulimit
stack=67108864 nvcr.io/nvidia/tensorflow:20.03-tf2-py3

=====
== TensorFlow ==
=====

NVIDIA Release 20.03-tf2 (build 11026100)
TensorFlow Version 2.1.0

Container image Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved.
Copyright 2017-2019 The TensorFlow Authors. All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying
project or file.

NOTE: MOFED driver for multi-node communication was not detected.
Multi-node communication performance may be reduced.

root@c64bb1f70737:/workspace# cd nvidia-examples/
root@c64bb1f70737:/workspace/nvidia-examples# ls
big_lstm build imagenet_data cnn tensorrt
root@c64bb1f70737:/workspace/nvidia-examples# python cnn/resnet.py
...
WARNING:tensorflow:Expected a shuffled dataset but input dataset `x` is not
shuffled. Please invoke `shuffle()` on input dataset.
2020-06-15 00:01:49.476393: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2020-06-15 00:01:49.701149: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
global_step: 10 images_per_sec: 93.2
global_step: 20 images_per_sec: 276.8
global_step: 30 images_per_sec: 276.4
```

Chapter 3. NVIDIA Compute Software Support Matrix for WSL 2

Package	Suggested Versions
NVIDIA Windows Driver x86 - CUDA, Video Nvidia-smi (Limited Feature Set)	Latest driver on WIP or devzone Pascal and later GPUs
NVIDIA Container Toolkit	Minimum versions - v2.6.0 with libnvidia-container - 1.5.1+
CUDA toolkit	Latest CUDA toolkit from 11.x releases can be used. Developer tools: Debuggers and Profilers are not supported yet
NCCL	2.11.4+

Chapter 4. Known Limitations for Linux CUDA Applications

The following table lists the known limitations on WSL 2 that may affect CUDA applications that use some of these features that are fully supported on Linux.

Limitations	Impact
Maxwell GPU is not supported.	Maxwell GPUs are not officially supported in WSL 2, but it may still work. Pascal and later GPU is recommended.
Unified Memory - Full Managed Memory Support is not available on Windows native and therefore WSL 2 will not support it for the foreseeable future.	UVM full features will not be available and therefore applications relying on UVM full features may not work. If your application is using Managed Memory, your application could see reduced performance and high system memory usage. Concurrent CPU/GPU access is not supported. CUDA queries will say whether it is supported or not and applications are expected to check this.
Pinned system memory (example: System memory that an application makes resident for GPU accesses) availability for applications is limited.	For example, some deep learning training workloads, depending on the framework, model and dataset size used, can exceed this limit and may not work.
Root user on bare metal (not containers) will not find nvidia-smi at the expected location.	Use <code>/usr/lib/wsl/lib/nvidia-smi</code> or manually add <code>/usr/lib/wsl/lib/</code> to the PATH).
With the NVIDIA Container Toolkit for Docker 19.03, only <code>--gpus all</code> is supported.	On multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.

Chapter 5. Features Not Yet Supported

The following table lists the set of features that are currently not supported.

Limitations	Impact
Legacy CUDA IPC APIs are not yet supported. Newer style cumemmap IPC with fd is supported.	Typically multi-process / multi-gpu applications will be impacted.
CUDA Developer tools - Debuggers and Profilers	Developers who require debugging support are encouraged to find alternatives in the meanwhile.
NVML (nvidia-smi) does not support all the queries yet.	GPU utilization, active compute process are some queries that are not yet supported. Modifiable state features (ECC, Compute mode, Persistence mode) will not be supported.
OpenGL-CUDA Interop is not yet supported.	Applications relying on OpenGL will not work.

Chapter 6. Troubleshooting

6.1. Container Runtime Initialization Errors

In some cases, when running a Docker container, you may encounter `nvidia-container-cli : initialization error`:

```
$ sudo docker run --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -
benchmark
docker: Error response from daemon: OCI runtime create failed:
container_linux.go:349: starting container process caused "process_linux.go:449:
container_init caused \"process_linux.go:432: running prestart hook 0 caused
\\\"error running hook: exit status 1, stdout: , stderr: nvidia-container-cli:
initialization error: driver error: failed to process request\\\\\\n\\\\\\\"\": unknown.
ERRO[0000] error waiting for container: context canceled
```

This usually indicates that the right Windows OS build or Microsoft Windows Insider Preview Builds (Windows 10 only), WSL 2, NVIDIA drivers and NVIDIA Container Toolkit may not be installed correctly. Review the known issues and changelog sections to ensure the right versions of the driver and container toolkit are installed.

Ensure you have followed through the steps listed under Setup under Running CUDA containers; especially ensure that the docker daemon is still running.

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

Or start the daemon directly and see if that resolves the issue:

```
$ sudo dockerd
```

If you are still running into this issue, use the dxdiag tools from the Run dialog and provide the diagnostic logs to NVIDIA by posting in the [Developer Forums](#) or by filing a [report](#).

You can also use the CUDA on WSL 2 [Developer Forums](#) to get in touch with NVIDIA product and engineering teams for help.

6.2. Checking WSL Kernel Version

1. Ensure you have the latest kernel by running the following command in PowerShell:

```
$ wsl cat /proc/version  
  
Linux version 5.10.16.3-microsoft-standard-WSL2  
(x86_64-msft-linux-gcc (GCC) 9.3.0, GNU ld (GNU Binutils) 2.34.0.20200220) #1 SMP  
Fri Apr 2 22:23:49 UTC 2021
```

2. If you don't have the last WSL kernel updated, you will see the following blocking warning upon trying to launch a Linux distribution within the WSL 2 container:

Chapter 7. Release Notes

7.1. 510.06

7.1.1. Changelog

- ▶ 8/31/2021:
 - ▶ NVIDIA Driver for Windows 10 and Windows 11
 - ▶ WIP Build: 22000.160 on devchannel, 19044.1200 on release preview channel
 - ▶ WSL Linux Kernel 5.10.43

7.1.2. New Features

The following new features are included in this release:

- ▶ CUDA on WSL2 support for Windows 10 21H2 that is available on the release preview channel. <https://blogs.windows.com/windowsexperience/2021/07/15/introducing-the-next-feature-update-to-windows-10-21h2/>
- ▶ Added the following support with containers:
 - ▶ NVML/nvidia-smi
 - ▶ Windows 10 21H2 in release preview channel

Users are recommended to install `nvidia-docker2` and its dependencies by enabling the `experimental` branch of the repos. After installation, verify that the `libnvidia-container` package is at least 1.5.0-rc.2 either using `apt list | grep -in libnvidia-container` or `nvidia-container-cli --version`

See <https://docs.nvidia.com/cuda/wsl-user-guide/index.html#ch05-running-containers>
- ▶ Users can also now query the GPU architecture in plain text using `nvidia-smi -q`

```
=====NVSMI LOG=====
Driver Version           : 4XXX
CUDA Version             : 11.X
```

```

Attached GPUs                : 1
GPU 00000000:XX:00.0
Product Name                 : NVIDIA RTX XXXXX
Product Brand                : NVIDIA RTX
Product Architecture        : Ampere
<...>

```

7.1.3. Resolved Issues

With the above latest nvidia-docker2 from the experimental branch, the following issue is resolved:

- ▶ When running the NGC Deep Learning (DL) Framework GPU containers in WSL 2, you will no longer encounter the below message:

```
The NVIDIA Driver was not detected. GPU functionality will not be
available.
```

7.1.4. Known Limitations

The following features are not supported in this release:

1. Note that NVIDIA Container Toolkit has not yet been validated with [Docker Desktop WSL 2](#) backend. Use Docker-CE for Linux instead inside your WSL 2 Linux distribution.
2. CUDA debugging or profiling tools are not supported in WSL 2. This capability will be added in a future release.
3. cumemmap IPC with fd is now supported. Other Legacy IPC APIs are not yet supported.
4. Unified Memory is limited to the same feature set as on native Windows systems.
5. With the NVIDIA Container Toolkit for Docker 19.03, only `--gpus all` is supported. This means that on multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.

7.1.5. Known Issues

The following are known issues in this release:

- ▶ `"/usr/lib/wsl/lib"` is not included in the WSL PATH environment for OS Win10 21H2 (19044.1200).

To use nvidia-smi on Win 10 21H1(19044.1200), manually add `"/usr/lib/wsl/lib"` to the PATH or invoke the nvidia-smi tool within WSL specifying the full path as `/usr/lib/wsl/lib/nvidia-smi`.

7.2. 471.21

Changelog

- ▶ 7/14/2021:

- ▶ NVIDIA Driver for Windows 10 and later: 471.21
- ▶ Windows 11 officially supported
- ▶ WIP build: 22000, WSL Linux Kernel 5.10.43

New Features

The following new features are included in this release:

- ▶ None.

Known Limitations

The following features are not supported in this release:

- ▶ Note that NVIDIA Container Toolkit has not yet been validated with [Docker Desktop WSL 2](#) backend. Use Docker-CE for Linux instead inside your WSL 2 Linux distribution.
- ▶ CUDA debugging or profiling tools are not supported in WSL 2. This capability will be added in a future release.
- ▶ `cumemmap` IPC with `fd` is now supported. Other Legacy IPC APIs are not yet supported.
- ▶ Unified Memory is limited to the same feature set as on native Windows systems.
- ▶ With the NVIDIA Container Toolkit for Docker 19.03, only `--gpus all` is supported. This means that on multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.
- ▶ When running the NGC Deep Learning (DL) Framework GPU containers in WSL 2, you may encounter a message:

```
The NVIDIA Driver was not detected. GPU functionality will not be available.
```

Note that this message is an incorrect warning for WSL 2 and will be fixed in future releases of the DL Framework containers to correctly detect the NVIDIA GPUs. The DL Framework containers will still continue to be accelerated using CUDA on WSL 2.

Known Issues

The following are known issues in this release:

- ▶ In-game Vsync fails to cap the frames to the refresh rate of the display only when Gsync is enabled. This will impact DX in-game vsync scenarios on native Windows with Gsync.

7.3. 470.76

- ▶ 6/3/2021:
 - ▶ `nvidia-smi` is enabled with limited feature support

- ▶ Fixes in the NVML libraries: Symbols not exported correctly in the NVML Lib preventing the library to be loaded in some cases
- ▶ Some performance optimization in the CUDA driver related to memory allocation
- ▶ Bug fixes in the CUDA driver for WSL
- ▶ NVIDIA Driver for Windows 10: 470.76
- ▶ WIP build: 21390, WSL Linux Kernel: 5.10.16

7.4. 470.14

- ▶ 3/23/2021: Resolved issues with nvidia-smi crashing on some systems.
 - ▶ NVIDIA Driver for Windows 10: 470.14
 - ▶ WIP build: 21332, WSL Linux kernel 5.4.91
- ▶ 3/9/2021
 - ▶ WIP OS 21313+ and Linux kernel 5.4.91
 - ▶ Do not use WIP OS 21327

7.5. 465.42

- ▶ 1/28/2021: CUDA Toolkit 11.2 support, nvidia-smi, NVML support and critical performance improvements.
The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 465.42
 - ▶ Recommended WIP build: 21292

7.6. 465.21

- ▶ 12/16/2020: Support for 3060Ti. Fix for installation problems observed in notebooks
The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 465.21

7.7. 465.12

- ▶ 11/16/2020: The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 465.12

7.8. 460.20

- ▶ 9/23/2020: The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 460.20

7.9. 460.15

- ▶ 9/2/2020:
The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 460.15

7.10. 455.41

- ▶ 6/19/2020: Updated driver release to address cache coherency issues on some CPU systems, including AMD Ryzen.
The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 455.41

7.11. 455.38

- ▶ 6/17/2020: Initial Version.
The following software versions are supported with this preview release for WSL 2:
 - ▶ NVIDIA Driver for Windows 10: 455.38
 - ▶ NVIDIA Container Toolkit: nvidia-docker2 (2.3) and libnvidia-container (>= 1.2.0-rc.1)

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2021 NVIDIA Corporation & affiliates. All rights reserved.