Installation Guide

Supported Platforms

The NVIDIA Container Toolkit is available on a variety of Linux distributions and supports different container engines.

Note

As of NVIDIA Container Toolkit `1.7.0` (`nvidia-docker2 >= 2.8.0`) support for Jetson plaforms is included for Ubuntu 18.04, Ubuntu 20.04, and Ubuntu 22.04 distributions. This means that the installation instructions provided for these distributions are expected to work on Jetson devices.

Linux Distributions

Supported Linux distributions are listed below:

OS Name / Version	Identifier	amd64 / x86_64	ppc64le	arm64 / aarch64
Amazon Linux 2	amzn2	X		X
Amazon Linux 2017.09	amzn2017.09	X		
Amazon Linux 2018.03	amzn2018.03	X		
Open Suse/SLES 15.0	sles15.0	X		
Open Suse/SLES 15.x (*)	sles15.x	X		
Debian Linux 9	debian9	X		
Debian Linux 10	debian10	X		
Debian Linux 11 (#)	debian11	X		
Centos 7	centos7	X	X	
Centos 8	centos8	X	X	X
RHEL 7.x (&)	rhel7.x	X	X	
RHEL 8.x (@)	rhel8.x	X	X	X

OS Name / Version	Identifier	amd64 / x86_64	ppc64le	arm64 / aarch64
RHEL 9.x (@)	rhel9.x	X	X	X
Ubuntu 16.04	ubuntu16.04	X	X	
Ubuntu 18.04	ubuntu18.04	X	X	X
Ubuntu 20.04 (%)	ubuntu20.04	X	X	X
Ubuntu 22.04 (%)	ubuntu22.04	X	X	X

(*) Minor releases of Open Suse/SLES 15.x are symlinked (redirected) to `sles15.1`. (#) Debian 11 packages are symlinked (redirected) to `debian10`. (&) RHEL 7 packages are symlinked (redirected) to `centos7` (&) RHEL 8 and RHEL 9 packages are symlinked (redirected) to `centos8` (%) Ubuntu 20.04 and Ubuntu 22.04 packages are symlinked (redirected) to `ubuntu18.04`

Container Runtimes

Supported container runtimes are listed below:

OS Name / Version	amd64 / x86_64	ppc64le	arm64 / aarch64
Docker 18.09	X	X	X
Docker 19.03	X	X	X
Docker 20.10	X	X	X
RHEL/CentOS 8 podman	X		
CentOS 8 Docker	X		
RHEL/CentOS 7 Docker	X		

Note

On Red Hat Enterprise Linux (RHEL) 8, Docker is no longer a supported container runtime. See Building, Running and Managing Containers for more information on the container tools available on the distribution.

Pre-Requisites

NVIDIA Drivers

Before you get started, make sure you have installed the NVIDIA driver for your Linux distribution. The recommended way to install drivers is to use the package manager for your distribution but other installer mechanisms are also available (e.g. by downloading `.run` installers from NVIDIA Driver Downloads).

For instructions on using your package manager to install drivers from the official CUDA network repository, follow the steps in this guide.

Platform Requirements

The list of prerequisites for running NVIDIA Container Toolkit is described below:

1. GNU/Linux x86_64 with kernel version > 3.10
2. Docker >= 19.03 (recommended, but some distributions may include older versions of Docker. The minimum supported version is 1.12)
3. NVIDIA GPU with Architecture >= Kepler (or compute capability 3.0)
4. NVIDIA Linux drivers >= 418.81.07 (Note that older driver releases or branches are unsupported.)

Note

Your driver version might limit your CUDA capabilities. Newer NVIDIA drivers are backwards-compatible with CUDA Toolkit versions, but each new version of CUDA requires a minimum driver version. Running a CUDA container requires a machine with at least one CUDA-capable GPU and a driver compatible with the CUDA toolkit version you are using. The machine running the CUDA container only requires the NVIDIA driver, the CUDA toolkit doesn't have to be installed. The CUDA release notes includes a table of the minimum driver and CUDA Toolkit versions.

Docker

Getting Started

For installing Docker CE, follow the official instructions for your supported Linux distribution. For convenience, the documentation below includes instructions on installing Docker for various Linux distributions.

Warning

If you are migrating fron `nvidia-docker` 1.0, then follow the instructions in the Migration from nvidia-docker 1.0 guide.

Installing on Ubuntu and Debian

The following steps can be used to setup NVIDIA Container Toolkit on Ubuntu LTS (18.04, 20.04, and 22.04) and Debian (Stretch, Buster) distributions.

Setting up Docker

Docker-CE on Ubuntu can be setup using Docker's official convenience script:

```
$ curl https://get.docker.com | sh \
  && sudo systemctl --now enable docker
```

See also

Follow the official instructions for more details and post-install actions.

Setting up NVIDIA Container Toolkit

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-
container.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
```

Note

To get access to `experimental` features and access to release candidates, you may want to add the `experimental` branch to the repository listing:

```
$  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/experimental/$distribution/libnvidia-container.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
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used instead of the `libnvidia-container` repositories above.

Note

Note that in some cases the downloaded list file may contain URLs that do not seem to match the expected value of `distribution` which is expected as packages may be used for all compatible distributions. As an examples:

- For `distribution` values of `ubuntu20.04` or `ubuntu22.04` the file will contain `ubuntu18.04` URLs
- For a `distribution` value of `debian11` the file will contain `debian10` URLs

Note

If running `apt update` after configuring repositories raises an error regarding a conflict in the Signed-By option, see the relevant troubleshooting section.

Install the `nvidia-docker2` package (and dependencies) after updating the package listing:

```
$ sudo apt-get update
```

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

Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

At this point, a working setup can be tested by running a base CUDA container:

```
$ 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                                                 |
+-----------------------------------------------------------------------------+
```

Installing on CentOS 7/8

The following steps can be used to setup the NVIDIA Container Toolkit on CentOS 7/8.

Setting up Docker on CentOS 7/8

Note

If you're on a cloud instance such as EC2, then the official CentOS images may not include tools such as `iptables` which are required for a successful Docker installation. Try this command to get a more functional VM, before proceeding with the remaining steps outlined in this document.

```
$ sudo dnf install -y tar bzip2 make automake gcc gcc-c++ vim pciutils elfutils-
libelf-devel libglvnd-devel iptables
```

Setup the official Docker CE repository:

CentOS 8 CentOS 7

```
$ sudo dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-
ce.repo
```

Now you can observe the packages available from the *docker-ce* repo:

CentOS 8 CentOS 7

```
$ sudo dnf repolist -v
```

Since CentOS does not support specific versions of `containerd.io` packages that are required for newer versions of Docker-CE, one option is to manually install the `containerd.io` package and then proceed to install the `docker-ce` packages.

Install the `containerd.io` package:

CentOS 8 CentOS 7

```
$ sudo dnf install -y
https://download.docker.com/linux/centos/7/x86_64/stable/Packages/containerd.io-
1.4.3-3.1.el7.x86_64.rpm
```

And now install the latest `docker-ce` package:

CentOS 8 CentOS 7

```
$ sudo dnf install docker-ce -y
```

Ensure the Docker service is running with the following command:

```
$ sudo systemctl --now enable docker
```

And finally, test your Docker installation by running the `hello-world` container:

```
$ sudo docker run --rm hello-world
```

This should result in a console output shown below:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Setting up NVIDIA Container Toolkit

Setup the repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.repo | sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
```

Note

To get access to `experimental` features and access to release candidates, you may want to add the `experimental` branch to the repository listing:

```
$ yum-config-manager --enable libnvidia-container-experimental
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used instead of the `libnvidia-container` repositories above.

Install the `nvidia-docker2` package (and dependencies) after updating the package listing:

CentOS 8 | CentOS 7

```
$ sudo dnf clean expire-cache --refresh
```

CentOS 8 | CentOS 7

```
$ sudo dnf install -y nvidia-docker2
```

Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

At this point, a working setup can be tested by running a base CUDA container:

```
$ 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                                                 |
+-----------------------------------------------------------------------------+
```

Installing on RHEL 7

The following steps can be used to setup the NVIDIA Container Toolkit on RHEL 7.

Setting up Docker on RHEL 7

RHEL includes Docker in the `Extras` repository. To install Docker on RHEL 7, first enable this repository:

```
$ sudo subscription-manager repos --enable rhel-7-server-extras-rpms
```

Docker can then be installed using `yum`

```
$ sudo yum install docker -y
```

See also

More information is available in the KB article.

Ensure the Docker service is running with the following command:

```
$ sudo systemctl --now enable docker
```

And finally, test your Docker installation. We can query the version info:

```
$ sudo docker -v
```

You should see an output like below:

```
Docker version 1.13.1, build 64e9980/1.13.1
```

And run the `hello-world` container:

```
$ sudo docker run --rm hello-world
```

Giving you the following result:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Setting up NVIDIA Container Toolkit

Setup the repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.repo | sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
```

Note

To get access to `experimental` features and access to release candidates, you may want to add the `experimental` branch to the repository listing:

```
$ yum-config-manager --enable libnvidia-container-experimental
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used instead of the `libnvidia-container` repositories above.

On RHEL 7, install the `nvidia-container-toolkit` package (and dependencies) after updating the package listing:

```
$ sudo yum clean expire-cache
```

```
$ sudo yum install nvidia-container-toolkit -y
```

Note

On POWER (`ppc64le`) platforms, the following package should be used: `nvidia-container-hook` instead of `nvidia-container-toolkit`

Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

At this point, a working setup can be tested by running a base CUDA container:

```
$ sudo docker run --rm -e NVIDIA_VISIBLE_DEVICES=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            Off  | 00000000:00:1E.0 Off |                    0 |
| N/A   43C    P0    20W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

Note

Depending on how your RHEL 7 system is configured with SELinux, you may have to use `--security-opt=label=disable` on the Docker command line to share parts of the host OS that can not be relabeled. Without this option, you may observe this error when running GPU containers: `Failed to initialize NVML: Insufficient Permissions`. However, using this option disables SELinux separation in the container and the container is executed in an unconfined type. Review the SELinux policies on your system.

Installing on SUSE 15

The following steps can be used to setup the NVIDIA Container Toolkit on SUSE SLES 15 and OpenSUSE Leap 15.

Setting up Docker on SUSE 15

To install the latest Docker 19.03 CE release on SUSE 15 (OpenSUSE Leap or SLES), you can use the `Virtualization::containers` project.

First, set up the repository:

```
$ sudo zypper addrepo
https://download.opensuse.org/repositories/Virtualization:containers/openSUSE_Leap_15.2/Virt
 \
   && sudo zypper refresh
```

Install the `docker` package:

```
$ sudo zypper install docker
```

Ensure the Docker service is running with the following command:

```
$ sudo systemctl --now enable docker
```

And finally, test your Docker installation by running the `hello-world` container:

```
$ sudo docker run --rm hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Setting up NVIDIA Container Toolkit

Note

You may have to set `$distribution` variable to `opensuse-leap15.1` explicitly when adding the repositories

Setup the repository and refresh the package listings

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && sudo zypper ar https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.repo
```

Note

To get access to `experimental` features and access to release candidates, you may want to add the `experimental` branch to the repository listing:

```
$ zypper modifyrepo --enable libnvidia-container-experimental
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used instead of the `libnvidia-container` repositories above.

Install the `nvidia-docker2` package (and dependencies) after updating the package listing:

```
$ sudo zypper refresh
```

```
$ sudo zypper install -y nvidia-docker2
```

Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

At this point, a working setup can be tested by running a base CUDA container:

```
$ 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                                                 |
+-----------------------------------------------------------------------------+
```

Installing on Amazon Linux

The following steps can be used to setup the NVIDIA Container Toolkit on Amazon Linux 1 and Amazon Linux 2.

Setting up Docker on Amazon Linux

Amazon Linux is available on Amazon EC2 instances. For full install instructions, see Docker basics for Amazon ECS.

After launching the official Amazon Linux EC2 image, update the installed packages and install the most recent Docker CE packages:

```
$ sudo yum update -y
```

Install the `docker` package:

```
$ sudo amazon-linux-extras install docker
```

Ensure the Docker service is running with the following command:

```
$ sudo systemctl --now enable docker
```

And finally, test your Docker installation by running the `hello-world` container:

```
$ sudo docker run --rm hello-world
```

This should result in a console output shown below:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:7f0a9f93b4aa3022c3a4c147a449bf11e0941a1fd0bf4a8e6c9408b2600777c5
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Setting up NVIDIA Container Toolkit

Setup the repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.repo | sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
```

Note

To get access to `experimental` features and access to release candidates, you may want to add the `experimental` branch to the repository listing:

```
$ yum-config-manager --enable libnvidia-container-experimental
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used instead of the `libnvidia-container` repositories above.

Install the `nvidia-docker2` package (and dependencies) after updating the package listing:

```
$ sudo yum clean expire-cache
```

Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

At this point, a working setup can be tested by running a base CUDA container:

```
$ 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                                                 |
+-----------------------------------------------------------------------------+
```

containerd

Getting Started

For installing containerd, follow the official instructions for your supported Linux distribution. For convenience, the documentation below includes instructions on installing containerd for various Linux distributions supported by NVIDIA.

Step 0: Pre-Requisites

To install `containerd` as the container engine on the system, install some pre-requisite modules:

```
$ sudo modprobe overlay \
    && sudo modprobe br_netfilter
```

You can also ensure these are persistent:

```
$ cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
```

Note

If you're going to use `containerd` as a CRI runtime with Kubernetes, configure the `sysctl` parameters:

```
$ cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

And then apply the params:

```
$ sudo sysctl --system
```

Step 1: Install containerd

After the pre-requisities, we can proceed with installing *containerd* for your Linux distribution.

Setup the Docker repository as described here:

Ubuntu LTS

1. Install packages to allow `apt` to use a repository over HTTPS:

```
$ sudo apt-get update
```

```
$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

2. Add the repository GPG key and the repo:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

Now, install the `containerd` package:

Ubuntu LTS

```
$ sudo apt-get update \
    && sudo apt-get install -y containerd.io
```

Configure `containerd` with a default `config.toml` configuration file:

```
$ sudo mkdir -p /etc/containerd \
    && sudo containerd config default | sudo tee /etc/containerd/config.toml
```

To make use of the NVIDIA Container Runtime, additional configuration is required. The following options should be added to configure `nvidia` as a runtime and use `systemd` as the cgroup driver. A patch is provided below:

```
$ cat <<EOF > containerd-config.patch
--- config.toml.orig    2020-12-18 18:21:41.884984894 +0000
+++ /etc/containerd/config.toml 2020-12-18 18:23:38.137796223 +0000
@@ -94,6 +94,15 @@
        privileged_without_host_devices = false
        base_runtime_spec = ""
        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
+            SystemdCgroup = true
+       [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]
+          privileged_without_host_devices = false
+          runtime_engine = ""
+          runtime_root = ""
+          runtime_type = "io.containerd.runc.v1"
+          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]
+            BinaryName = "/usr/bin/nvidia-container-runtime"
+            SystemdCgroup = true
    [plugins."io.containerd.grpc.v1.cri".cni]
    bin_dir = "/opt/cni/bin"
    conf_dir = "/etc/cni/net.d"
EOF
```

After apply the configuration patch, restart `containerd`:

```
$ sudo systemctl restart containerd
```

You can test the installation by using the Docker `hello-world` container with the `ctr` tool:

```
$ sudo ctr image pull docker.io/library/hello-world:latest \
    && sudo ctr run --rm -t docker.io/library/hello-world:latest hello-world
```

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Step 2: Install NVIDIA Container Toolkit

After installing containerd, we can proceed to install the NVIDIA Container Toolkit. For `containerd`, we need to use the `nvidia-container-toolkit` package. See the architecture overview for more details on the package hierarchy.

First, setup the package repository and GPG key:

Ubuntu LTS CentOS / RHEL

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
    && curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-key
add - \
    && curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.list | sudo tee
/etc/apt/sources.list.d/nvidia-container-toolkit.list
```

Now, install the NVIDIA Container Toolkit:

Ubuntu LTS CentOS / RHEL

```
$ sudo apt-get update \
    && sudo apt-get install -y nvidia-container-toolkit
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used and the `nvidia-container-runtime` package should be installed instead. This means that the package repositories should be set up as follows:

Ubuntu LTS | CentOS / RHEL

```
$ 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-container-toolkit.list
```

The installed packages can be confirmed by running:

Ubuntu LTS

```
$ sudo apt list --installed *nvidia*
```

Step 3: Testing the Installation

Then, we can test a GPU container:

```
$ sudo ctr image pull docker.io/nvidia/cuda:11.0.3-base-ubuntu20.04
$ sudo ctr run --rm -t \
    --runc-binary=/usr/bin/nvidia-container-runtime \
    --env NVIDIA_VISIBLE_DEVICES=all \
    docker.io/nvidia/cuda:11.0.3-base-ubuntu20.04 \
    cuda-11.0.3-base-ubuntu20.04 nvidia-smi
```

You should see an output similar to the one shown below:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    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                                                 |
+-----------------------------------------------------------------------------+
```

podman

Getting Started

For installing podman, follow the official instructions for your supported Linux distribution. For convenience, the documentation below includes instructions on installing podman on RHEL 8.

Step 1: Install podman

On RHEL 8, check if the `container-tools` module is available:

```
$ sudo dnf module list | grep container-tools
```

This should return an output as shown below:

```
container-tools      rhel8 [d]         common [d]                               Most
recent (rolling) versions of podman, buildah, skopeo, runc, conmon, runc, conmon, CRIU,
Udica, etc as well as dependencies such as container-selinux built and tested together,
and updated as frequently as every 12 weeks.
container-tools      1.0               common [d]                               Stable
versions of podman 1.0, buildah 1.5, skopeo 0.1, runc, conmon, CRIU, Udica, etc as well as
dependencies such as container-selinux built and tested together, and supported for 24
months.
container-tools      2.0               common [d]                               Stable
versions of podman 1.6, buildah 1.11, skopeo 0.1, runc, conmon, etc as well as
dependencies such as container-selinux built and tested together, and supported as
documented on the Application Stream lifecycle page.
container-tools      rhel8 [d]         common [d]                               Most
recent (rolling) versions of podman, buildah, skopeo, runc, conmon, runc, conmon, CRIU,
Udica, etc as well as dependencies such as container-selinux built and tested together,
and updated as frequently as every 12 weeks.
container-tools      1.0               common [d]                               Stable
versions of podman 1.0, buildah 1.5, skopeo 0.1, runc, conmon, CRIU, Udica, etc as well as
dependencies such as container-selinux built and tested together, and supported for 24
months.
container-tools      2.0               common [d]                               Stable
versions of podman 1.6, buildah 1.11, skopeo 0.1, runc, conmon, etc as well as
dependencies such as container-selinux built and tested together, and supported as
documented on the Application Stream lifecycle page.
```

Now, proceed to install the `container-tools` module, which will install `podman`:

```
$ sudo dnf module install -y container-tools
```

Once, `podman` is installed, check the version:

```
$ podman version
Version:      2.2.1
API Version:  2
Go Version:   go1.14.7
Built:        Mon Feb  8 21:19:06 2021
OS/Arch:      linux/amd64
```

Step 2: Install NVIDIA Container Toolkit

After installing `podman`, we can proceed to install the NVIDIA Container Toolkit. For `podman`, we need to use the `nvidia-container-toolkit` package. See the architecture overview for more details on the package hierarchy.

First, setup the package repository and GPG key:

Ubuntu LTS | CentOS / RHEL

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
    && curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-key
add - \
    && curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.list | sudo tee
/etc/apt/sources.list.d/nvidia-container-toolkit.list
```

Now, install the NVIDIA Container Toolkit:

Ubuntu LTS | CentOS / RHEL

```
$ sudo apt-get update \
    && sudo apt-get install -y nvidia-container-toolkit
```

Note

For version of the NVIDIA Container Toolkit prior to `1.6.0`, the `nvidia-docker` repository should be used and the `nvidia-container-runtime` package should be installed instead. This means that the package repositories should be set up as follows:

Ubuntu LTS | CentOS / RHEL

```
$ 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-container-toolkit.list
```

The installed packages can be confirmed by running:

Ubuntu LTS

```
$ sudo apt list --installed *nvidia*
```

Step 2.1. Check the installation

Once the package installation is complete, ensure that the `hook` has been added:

```
$ cat  /usr/share/containers/oci/hooks.d/oci-nvidia-hook.json
```

```
{
    "version": "1.0.0",
    "hook": {
        "path": "/usr/bin/nvidia-container-toolkit",
        "args": ["nvidia-container-toolkit", "prestart"],
        "env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
        ]
    },
    "when": {
        "always": true,
        "commands": [".*"]
    },
    "stages": ["prestart"]
}
```

Step 3: Rootless Containers Setup

To be able to run rootless containers with `podman`, we need the following configuration change to the NVIDIA runtime:

```
$ sudo sed -i 's/^#no-cgroups = false/no-cgroups = true/;' /etc/nvidia-container-
runtime/config.toml
```

Note

If the user running the containers is a privileged user (e.g. `root`) this change should not be made and will cause containers using the NVIDIA Container Toolkit to fail.

Step 4: Running Sample Workloads

We can now run some sample GPU containers to test the setup.

1. Run `nvidia-smi`

   ```
   $ podman run --rm --security-opt=label=disable \
        --hooks-dir=/usr/share/containers/oci/hooks.d/ \
        nvidia/cuda:11.0.3-base-ubuntu20.04 nvidia-smi
   ```

   which should produce the following output:

   ```
   +-----------------------------------------------------------------------------+
   | NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
   |-------------------------------+----------------------+----------------------+
   | 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            Off  | 00000000:00:1E.0 Off |                    0 |
   | N/A   46C    P0    27W /  70W |      0MiB / 15109MiB |      0%      Default |
   |                               |                      |                  N/A |
   +-------------------------------+----------------------+----------------------+

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

2. Run an FP16 GEMM workload on the GPU that can leverage the Tensor Cores when available:

```
$ podman run --rm --security-opt=label=disable \
     --hooks-dir=/usr/share/containers/oci/hooks.d/ \
     --cap-add SYS_ADMIN nvidia/samples:dcgmproftester-2.0.10-cuda11.0-ubuntu18.04
\
     --no-dcgm-validation -t 1004 -d 30
```

You should be able to see an output as shown below:

```
Skipping CreateDcgmGroups() since DCGM validation is disabled
CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_MULTIPROCESSOR: 1024
CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT: 40
CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_MULTIPROCESSOR: 65536
CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MAJOR: 7
CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MINOR: 5
CU_DEVICE_ATTRIBUTE_GLOBAL_MEMORY_BUS_WIDTH: 256
CU_DEVICE_ATTRIBUTE_MEMORY_CLOCK_RATE: 5001000
Max Memory bandwidth: 320064000000 bytes (320.06 GiB)
CudaInit completed successfully.

Skipping WatchFields() since DCGM validation is disabled
TensorEngineActive: generated ???, dcgm 0.000 (27334.5 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27795.5 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27846.0 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27865.9 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27837.6 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27709.7 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27615.3 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27620.3 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27530.7 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27477.4 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27461.1 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27454.6 gflops)
TensorEngineActive: generated ???, dcgm 0.000 (27381.2 gflops)
```