Installation Guide for Linux

Release 12.9

NVIDIA Corporation

Apr 18, 2025

Contents

NVIDIA CUDA Installation Guide for Linux

The installation instructions for the CUDA Toolkit on Linux.

CUDA® is a parallel computing platform and programming model invented by NVIDIA®. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA was developed with several design goals in mind:

- Provide a small set of extensions to standard programming languages, like C, that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources.

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

This guide will show you how to install and check the correct operation of the CUDA development tools.

Note: Instructions for installing NVIDIA Drivers are now in https://docs.nvidia.com/datacenter/tesla/driver-installation-guide/index.html.

Chapter 1. System Requirements

To use NVIDIA CUDA on your system, you will need the following installed:

- CUDA-capable GPU
- A supported version of Linux with a gcc compiler and toolchain
- CUDA Toolkit (available at https://developer.nvidia.com/cuda-downloads)

The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for this CUDA Toolkit release.

The following table lists the supported Linux distributions. Please review the footnotes associated with the table.

Table 1: Native Linux Distribution Support in CUDA 12.9

Distribution	**Kernel**[1]	**Default GCC**	**GLIBC**
x86_64			
RHEL 9.y (y <= 5)	5.14.0-503	11.5.0	2.34
RHEL 8.y (y <= 10)	4.18.0-553	8.5.0	2.28
OpenSUSE Leap 15.y (y == 6)	6.4.0-150600.21	7.5.0	2.38
Rocky Linux 8.y (y <= 10)	4.18.0-553	8.5.0	2.28
Rocky Linux 9.y (y <=5)	5.14.0-503	11.5.0	2.34
SUSE SLES 15.y (y <= 6)	6.4.0-150600.21	7.5.0	2.31
Ubuntu 24.04.z (z <= 2) LTS	6.8.0-41	13.3.0	2.39
Ubuntu 22.04.z (z <= 5) LTS	6.5.0-45	12.3.0	2.35
Ubuntu 20.04.z (z <= 6) LTS	5.15.0-67	9.4.0	2.31
Debian 12.9	6.1.123-1	12.2.0	2.36
Fedora 41	6.11.4-301	14.2.1	2.40
KylinOS V10 SP3 2403	4.19.90-89.11.v2401	10.x	2.28
MSFT Azure Linux <= 3.0	6.6.64.2-9.azl3	13.2.0	2.38-8
Amazon Linux 2023	6.1.82-99.168	11.4.1	2.34

continues on next page

Table 1 – continued from previous page

Distribution	**Kernel**[1]	**Default GCC**	**GLIBC**
Oracle Linux 8	4.18.0-553	8.5.0	2.28
Oracle Linux 9	5.14.0-427	11.4.1	2.34
Arm64 sbsa (Generic)			
RHEL 9.y (y <= 5)	5.14.0-503	11.5.0	2.34
RHEL 8.y (y <= 10)	4.18.0-553	8.5.0	2.28
SUSE SLES 15.y (y == 6)	6.4.0-150600.21	7.5.0	2.38
Kylin V10 SP3 2403	4.19.90-89	10.x	2.28
Ubuntu 24.04.z (z <= 1) LTS	6.8.0-31	13.2.0	2.39
Ubuntu 22.04 LTS (z <= 5) LTS	6.5.0-1019	11.4.0	2.35
Ubuntu 20.04.z (z <= 5) LTS	5.4.0-174	11.4.0	2.31
GRACE Arm64 sbsa			
MSFT Azure Linux 3.0	6.6.64.2-9.azl3	13.2.0	2.38-8
Arm64 sbsa Jetson (dGPU)			
20.04.06 LTS Rel35 JP 5.x	5.10.65-tegra	9.4.0	2.31
22.04.4 LTS Rel36 - JP6.x	5.15.136-tegra	11.4.0	2.35
AArch64 Jetson (iGPU)			
L4T Ubuntu 22.04 Rel36 - JP6.x	6.1.80-tegra	11.4.0	2.35

(1) The following notes apply to the kernel versions supported by CUDA:

- For specific kernel versions supported on Red Hat Enterprise Linux (RHEL), visit https://access.redhat.com/articles/3078.
- A list of kernel versions including the release dates for SUSE Linux Enterprise Server (SLES) is available at https://www.suse.com/support/kb/doc/?id=000019587.

(2) Support for Debian 11.x is deprecated.

Chapter 2. OS Support Policy

- CUDA support for Ubuntu 20.04.x, Ubuntu 22.04.x, Ubuntu 24.04.x, RHEL 8.x, RHEL 9.x, Rocky Linux 8.x, Rocky Linux 9.x, SUSE SLES 15.x, OpenSUSE Leap 15.x, Amazon linux 2023, and Azure Linux 2.0 will be until the standard EOSS as defined for each OS. Please refer to the support lifecycle for these OSes to know their support timelines.
- CUDA supports the latest Fedora release version. The version supported might require a specific GCC compatibility package. For Fedora release timelines, visit https://docs.fedoraproject.org/en-US/releases/.
- CUDA supports a single KylinOS release version. For details, visit https://www.kylinos.cn/.

Refer to the support lifecycle for these supported OSes to know their support timelines and plan to move to newer releases accordingly.

Chapter 3. Host Compiler Support Policy

In order to compile the CPU "Host" code in the CUDA source, the CUDA compiler NVCC requires a compatible host compiler to be installed on the system. The version of the host compiler supported on Linux platforms is tabulated as below. NVCC performs a version check on the host compiler's major version and so newer minor versions of the compilers listed below will be supported, but major versions falling outside the range will not be supported.

Table 2: Supported Compilers

Distribution	GCC	Clang	NVHPC	XLC	ArmC/C++	ICC
x86_64	6.x - 14.x	7.x - 19.x	24.9	No	No	2021.7
Arm64 sbsa	6.x - 14.x	7.x - 19.x	24.9	No	24.04	No

For GCC and Clang, the preceding table indicates the minimum version and the latest version supported. If you are on a Linux distribution that may use an older version of GCC toolchain as default than what is listed above, it is recommended to upgrade to a newer toolchain CUDA 11.0 or later toolkit. Newer GCC toolchains are available with the Red Hat Developer Toolset for example. For platforms that ship a compiler version older than GCC 6 by default, linking to static or dynamic libraries that are shipped with the CUDA Toolkit is not supported. We only support libstdc++ (GCC's implementation) for all the supported host compilers for the platforms listed above.

3.1. Host Compiler Compatibility Packages

Really up to date distributions might ship with a newer compiler than what is covered by the Supported Compilers table above. Usually, those distribution also provide a GCC compatibility package that can be used instead of the default one.

Depending on the distribution, the package that needs to be installed is different, but the logic for configuring it is the same. If required, configuration steps are described in the relevant section for the specific Linux distribution, but they always end up with configuring the `NVCC_BIN` environment variable as described in the NVCC documentation.

3.2. Supported C++ Dialects

NVCC and NVRTC (CUDA Runtime Compiler) support the following C++ dialect: C++11, C++14, C++17, C++20 on supported host compilers. The default C++ dialect of NVCC is determined by the default dialect of the host compiler used for compilation. Refer to host compiler documentation and the *CUDA Programming Guide* for more details on language support.

C++20 is supported with the following flavors of host compiler in both host and device code.

GCC	Clang	NVHPC	Arm C/C++
>=10.x	>=11.x	>=22.x	>=22.x

Chapter 4. About This Document

This document is intended for readers familiar with the Linux environment and the compilation of C programs from the command line. You do not need previous experience with CUDA or experience with parallel computation. Note: This guide covers installation only on systems with X Windows installed.

Note: Many commands in this document might require *superuser* privileges. On most distributions of Linux, this will require you to log in as root. For systems that have enabled the sudo package, use the sudo prefix for all necessary commands.

Chapter 5. Pre-installation Actions

Some actions must be taken before the CUDA Toolkit can be installed on Linux:

- Verify the system has a CUDA-capable GPU.
- Verify the system is running a supported version of Linux.
- Verify the system has gcc installed.
- Download the NVIDIA CUDA Toolkit.
- Handle conflicting installation methods.

Note: You can override the install-time prerequisite checks by running the installer with the `-override` flag. Remember that the prerequisites will still be required to use the NVIDIA CUDA Toolkit.

5.1. Verify You Have a CUDA-Capable GPU

To verify that your GPU is CUDA-capable, go to your distribution's equivalent of System Properties, or, from the command line, enter:

```
lspci | grep -i nvidia
```

If you do not see any settings, update the PCI hardware database that Linux maintains by entering `update-pciids` (generally found in `/sbin`) at the command line and rerun the previous `lspci` command.

If your graphics card is from NVIDIA and it is listed in https://developer.nvidia.com/cuda-gpus, your GPU is CUDA-capable.

The Release Notes for the CUDA Toolkit also contain a list of supported products.

5.2. Verify You Have a Supported Version of Linux

The CUDA Development Tools are only supported on some specific distributions of Linux. These are listed in the CUDA Toolkit release notes.

To determine which distribution and release number you're running, type the following at the command line:

```
uname -m && cat /etc/*release
```

You should see output similar to the following, modified for your particular system:

```
x86_64
Red Hat Enterprise Linux Workstation release 6.0 (Santiago)
```

The `x86_64` line indicates you are running on a 64-bit system. The remainder gives information about your distribution.

5.3. Verify the System Has gcc Installed

The `gcc` compiler is required for development using the CUDA Toolkit. It is not required for running CUDA applications. It is generally installed as part of the Linux installation, and in most cases the version of gcc installed with a supported version of Linux will work correctly.

To verify the version of gcc installed on your system, type the following on the command line:

```
gcc --version
```

If an error message displays, you need to install the development tools from your Linux distribution or obtain a version of `gcc` and its accompanying toolchain from the Web.

5.4. Choose an Installation Method

The CUDA Toolkit can be installed using either of two different installation mechanisms: distribution-specific packages (RPM and Deb packages), or a distribution-independent package (runfile packages).

The distribution-independent package has the advantage of working across a wider set of Linux distributions, but does not update the distribution's native package management system. The distribution-specific packages interface with the distribution's native package management system. It is recommended to use the distribution-specific packages, where possible.

Note: For both native as well as cross development, the toolkit must be installed using the distribution-specific installer. See the *CUDA Cross-Platform Installation* section for more details.

5.5. Download the NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit is available at https://developer.nvidia.com/cuda-downloads.

Choose the platform you are using and download the NVIDIA CUDA Toolkit.

The CUDA Toolkit contains the tools needed to create, build and run a CUDA application as well as libraries, header files, and other resources.

The download can be verified by comparing the MD5 checksum posted at https://developer.download.nvidia.com/compute/cuda/12.6.3/docs/sidebar/md5sum.txt with that of the downloaded file. If either of the checksums differ, the downloaded file is corrupt and needs to be downloaded again.

5.6. Handle Conflicting Installation Methods

Before installing CUDA, any previous installations that could conflict should be uninstalled. This will not affect systems which have not had CUDA installed previously, or systems where the installation method has been preserved (RPM/Deb vs. Runfile). See the following charts for specifics.

Table 3: CUDA Toolkit Installation Compatibility Matrix

		Installed Toolkit Version == X.Y		Installed Toolkit Version != X.Y	
		RPM/Deb	run	RPM/Deb	run
Installing Toolkit Version X.Y	RPM/Deb	No Action	Uninstall Run	No Action	No Action
	run	Uninstall RPM/Deb	Uninstall Run	No Action	No Action

Use the following command to uninstall a Toolkit runfile installation:

```
sudo /usr/local/cuda-X.Y/bin/cuda-uninstaller
```

Use the following commands to uninstall an RPM/Deb installation:

```
sudo dnf remove <package_name>                      # RHEL 8 / Rocky Linux 8 / RHEL 9
↪/ Rocky Linux 9 / Fedora / KylinOS 10 / Amazon Linux 2023
```

```
sudo tdnf remove <package_name>                     # Azure Linux
```

```
sudo zypper remove <package_name>                   # OpenSUSE / SLES
```

```
sudo apt-get --purge remove <package_name>          # Debian / Ubuntu
```

Chapter 6. Package Manager Installation

Basic instructions can be found in the Quick Start Guide. Read on for more detailed instructions.

6.1. Overview

Installation using RPM or Debian packages interfaces with your system's package management system. When using RPM or Debian local repo installers, the downloaded package contains a repository snapshot stored on the local filesystem in /var/. Such a package only informs the package manager where to find the actual installation packages, but will not install them.

If the online network repository is enabled, RPM or Debian packages will be automatically downloaded at installation time using the package manager: apt-get, dnf, tdnf, or zypper.

Distribution-specific instructions detail how to install CUDA:

- *RHEL / Rocky*
- *KylinOS*
- *Fedora*
- *SLES*
- *OpenSUSE*
- *WSL*
- *Ubuntu*
- *Debian*
- *Amazon Linux*
- *Azure Linux CM2*

Finally, some helpful *package manager capabilities* are detailed.

These instructions are for native development only. For cross-platform development, see the *CUDA Cross-Platform Environment* section.

Note: Optional components such as `nvidia-fs`, `libnvidia_nscq`, and `fabricmanager` are not installed by default and will have to be installed separately as needed.

6.2. RHEL / Rocky

6.2.1. Prepare RHEL / Rocky

1. Perform the *Pre-installation Actions*.
2. **Satisfy third-party package dependency:**
   - **Enable optional repos:**

     On **RHEL 9 Linux** only, execute the following steps to enable optional repositories.

     - **On x86_64 systems:**

       ```
       subscription-manager repos --enable=rhel-9-for-x86_64-appstream-rpms
       subscription-manager repos --enable=rhel-9-for-x86_64-baseos-rpms
       subscription-manager repos --enable=codeready-builder-for-rhel-9-x86_64-
       ↪rpms
       ```

     On **RHEL 8 Linux** only, execute the following steps to enable optional repositories.

     - **On x86_64 systems:**

       ```
       subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms
       subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
       subscription-manager repos --enable=codeready-builder-for-rhel-8-x86_64-
       ↪rpms
       ```

3. **Remove Outdated Signing Key:**

   ```
   sudo rpm --erase gpg-pubkey-7fa2af80*
   ```

4. Choose an installation method: *Local Repo Installation for RHEL / Rocky* or *Network Repo Installation for RHEL / Rocky*.

6.2.2. Local Repo Installation for RHEL / Rocky

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.<arch>.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - `rhel8`
   - `rhel9`

   and `<arch>` should be replaced by one of the following:

   - `x86_64`
   - `aarch64`

6.2.3. Network Repo Installation for RHEL / Rocky

1. **Enable the network repo:**

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/<arch>/cuda-<distro>.repo
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `rhel8/sbsa`
   - `rhel8/x86_64`
   - `rhel9/sbsa`
   - `rhel9/x86_64`

2. **Install the new CUDA public GPG key:**

   The new GPG public key for the CUDA repository (RPM-based distros) is d42d0685.

   On a fresh installation of RHEL, the dnf package manager will prompt the user to accept new keys when installing packages the first time. Indicate you accept the change when prompted.

   For upgrades, you must also also fetch an updated .repo entry:

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/<arch>/cuda-<distro>.repo
   ```

3. **Clean DNF repository:**

   ```
   sudo dnf clean all
   ```

6.2.4. Common Instructions for RHEL / Rocky

These instructions apply to both local and network installation.

1. **Install CUDA SDK:**

   ```
   sudo dnf install cuda-toolkit
   ```

2. **Install GPUDirect Filesystem:**

   ```
   sudo dnf install nvidia-gds
   ```

3. **Add libcuda.so symbolic link, if necessary**

   The `libcuda.so` library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use `libcuda.so`, it may be useful to add a symbolic link from `libcuda.so` in the `/usr/lib{,64}` directory.

4. **Reboot the system:**

   ```
   sudo reboot
   ```

5. Perform the *post-installation actions*.

6.3. KylinOS

6.3.1. Prepare KylinOS

1. Perform the *pre-installation actions*.
2. Choose an installation method: *local repo* or *network repo*.

6.3.2. Local Repo Installation for KylinOS

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.<arch>.rpm
   ```

   where <distro> should be replaced by one of the following:

   - kylin10

   and <arch> should be replaced by one of the following:

   - x86_64
   - aarch64

6.3.3. Network Repo Installation for KylinOS

1. **Enable the network repo:**

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/<arch>/cuda-<distro>.repo
   ```

   where <distro>/<arch> should be replaced by one of the following:

   - kylin10/sbsa
   - kylin10/x86_64

2. **Install the new CUDA public GPG key:**

   The new GPG public key for the CUDA repository (RPM-based distros) is d42d0685.

   On a fresh installation of KylinOS, the dnf package manager will prompt the user to accept new keys when installing packages the first time. Indicate you accept the change when prompted.

3. **Clean DNF repository:**

   ```
   sudo dnf clean all
   ```

6.3.4. Common Instructions for KylinOS

These instructions apply to both local and network installation.

1. **Install CUDA SDK:**

   ```
   sudo dnf install cuda-toolkit
   ```

2. **Install GPUDirect Filesystem:**

   ```
   sudo dnf install nvidia-gds
   ```

3. **Add libcuda.so symbolic link, if necessary**

   The `libcuda.so` library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use `libcuda.so`, it may be useful to add a symbolic link from `libcuda.so` in the `/usr/lib{,64}` directory.

4. **Reboot the system:**

   ```
   sudo reboot
   ```

5. Perform the *post-installation actions*.

6.4. Fedora

6.4.1. Prepare Fedora

1. Perform the *pre-installation actions*.
2. **Remove Outdated Signing Key:**

   ```
   sudo rpm --erase gpg-pubkey-7fa2af80*
   ```

3. Choose an installation method: *local repo* or *network repo*.

6.4.2. Local Repo Installation for Fedora

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.x86_64.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - `fedora39`

6.4.3. Network Repo Installation for Fedora

1. **Enable the network repo:**

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/x86_64/cuda-<distro>.repo
   ```

   where `<distro>` should be replaced by one of the following:

   - `fedora39`

2. **Install the new CUDA public GPG key:**

   The new GPG public key for the CUDA repository (RPM-based distros) is d42d0685.

   On a fresh installation of Fedora, the dnf package manager will prompt the user to accept new keys when installing packages the first time. Indicate you accept the change when prompted.

   For upgrades, you must also fetch an updated `.repo` entry:

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/x86_64/cuda-<distro>.repo
   ```

3. **Clean DNF repository:**

   ```
   sudo dnf clean all
   ```

6.4.4. Common Installation Instructions for Fedora

These instructions apply to both local and network installation for Fedora.

1. **Install CUDA SDK:**

   ```
   sudo dnf install cuda-toolkit
   ```

2. **Reboot the system:**

   ```
   sudo reboot
   ```

3. **Add libcuda.so symbolic link, if necessary:**

   The `libcuda.so` library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use `libcuda.so`, it may be useful to add a symbolic link from `libcuda.so` in the `/usr/lib{,64}` directory.

4. Perform the *post-installation actions*.

6.4.5. GCC Compatibility Package for Fedora

The Fedora version supported might ship with a newer compiler than what is actually supported by NVCC. This can be overcome by installing the GCC compatibility package and setting a few environment variables.

As an example, Fedora 41 ships with GCC 14 and also with a compatible GCC 13 version, which can be used for NVCC. To install and configure the local NVCC binary to use that version, proceed as follows.

1. Install the packages required:

   ```
   sudo dnf install gcc13-c++
   ```

   The binaries then appear on the system in the following way:

   ```
   /usr/bin/gcc-13
   /usr/bin/g++-13
   ```

2. Override the default **g++** compiler. Refer to the documentation for NVCC regarding the environment variables. For example:

   ```
   export NVCC_CCBIN='g++-13'
   ```

6.5. SLES

6.5.1. Prepare SLES

1. Perform the *pre-installation actions*.
2. **On SLES12 SP4, install the Mesa-libgl-devel Linux packages before proceeding.**

   See Mesa-libGL-devel.
3. **Add the user to the video group:**

   ```
   sudo usermod -a -G video <username>
   ```

4. **Remove Outdated Signing Key:**

   ```
   sudo rpm --erase gpg-pubkey-7fa2af80*
   ```

5. Choose an installation method: *local repo* or *network repo*.

6.5.2. Local Repo Installation for SLES

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.<arch>.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - `sles15`

   and `<arch>` should be replaced by one of the following:

   - `x86_64`
   - `aarch64`

6.5.3. Network Repo Installation for SLES

1. **Enable the network repo:**

   ```
   sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distro>/<arch>/cuda-<distro>.repo
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `sles15/sbsa`
   - `sles15/x86_64`

2. **Install the new CUDA public GPG key:**

   The new GPG public key for the CUDA repository (RPM-based distros) is d42d0685.

   On a fresh installation of SLES, the zypper package manager will prompt the user to accept new keys when installing packages the first time. Indicate you accept the change when prompted.

   For upgrades, you must also also fetch an updated .repo entry:

   ```
   sudo zypper removerepo cuda-<distro>-<arch>
   sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distro>/<arch>/cuda-<distro>.repo
   ```

3. **Refresh Zypper repository cache:**

   ```
   sudo SUSEConnect --product PackageHub/<SLES version number>/<arch>
   sudo zypper refresh
   ```

6.5.4. Common Installation Instructions for SLES

These instructions apply to both local and network installation for SLES.

1. **Install CUDA SDK:**

   ```
   sudo zypper install cuda-toolkit
   ```

2. **Reboot the system:**

   ```
   sudo reboot
   ```

3. Perform the *post-installation actions*.

6.6. OpenSUSE

6.6.1. Prepare OpenSUSE

1. Perform the *pre-installation actions*.
2. **Add the user to the video group:**

   ```
   sudo usermod -a -G video <username>
   ```

3. **Remove Outdated Signing Key:**

   ```
   sudo rpm --erase gpg-pubkey-7fa2af80*
   ```

4. Choose an installation method: *local repo* or *network repo*.

6.6.2. Local Repo Installation for OpenSUSE

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.x86_64.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - opensuse15

6.6.3. Network Repo Installation for OpenSUSE

1. **Enable the network repo:**

   ```
   sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distro>/x86_64/cuda-<distro>.repo
   ```

   where `<distro>` should be replaced by one of the following:

   - opensuse15

2. **Install the new CUDA public GPG key:**

   The new GPG public key for the CUDA repository (RPM-based distros) is d42d0685. On fresh installation of openSUSE, the zypper package manager will prompt the user to accept new keys when installing packages the first time. Indicate you accept the change when prompted.

   For upgrades, you must also also fetch an updated .repo entry:

   ```
   sudo zypper removerepo cuda-<distro>-x86_64
   sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distro>/x86_64/cuda-<distro>.repo
   ```

3. **Refresh Zypper repository cache:**

   ```
   sudo zypper refresh
   ```

6.6.4. Common Installation Instructions for OpenSUSE

These instructions apply to both local and network installation for OpenSUSE.

1. **Install CUDA SDK:**

   ```
   sudo zypper install cuda-toolkit
   ```

2. **Reboot the system:**

   ```
   sudo reboot
   ```

3. Perform the *post-installation actions*.

6.7. WSL

These instructions must be used if you are installing in a WSL environment.

6.7.1. Prepare WSL

1. Perform the *pre-installation actions*.
2. **Remove Outdated Signing Key:**

   ```
   sudo apt-key del 7fa2af80
   ```

3. Choose an installation method: *local repo* or *network repo*.

6.7.2. Local Repo Installation for WSL

1. **Install local repository on file system:**

   ```
   sudo dpkg -i cuda-repo-<distro>-X-Y-local_<version>*_amd64.deb
   ```

   where `<distro>` should be replaced by one of the following:

   - `wsl-ubuntu`

2. **Enroll ephemeral public GPG key:**

   ```
   sudo cp /var/cuda-repo-<distro>-X-Y-local/cuda-*-keyring.gpg /usr/share/keyrings/
   ```

3. **Add pin file to prioritize CUDA repository:**

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

6.7.3. Network Repo Installation for WSL

The new GPG public key for the CUDA repository (Debian-based distros) is 3bf863cc. This must be enrolled on the system, either using the `cuda-keyring` package or manually; the `apt-key` command is deprecated and not recommended.

1. **Install the new cuda-keyring package:**

   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/x86_64/
   ↪cuda-keyring_1.1-1_all.deb
   sudo dpkg -i cuda-keyring_1.1-1_all.deb
   ```

   where `<distro>` should be replaced by one of the following:

   - `wsl-ubuntu`

6.7.4. Common Installation Instructions for WSL

These instructions apply to both local and network installation for WSL.

1. **Update the Apt repository cache:**

   ```
   sudo apt-get update
   ```

2. **Install CUDA SDK:**

   ```
   sudo apt-get install cuda-toolkit
   ```

3. Perform the *post-installation actions*.

6.8. Ubuntu

6.8.1. Prepare Ubuntu

1. Perform the *pre-installation actions*.
2. **Remove Outdated Signing Key:**

   ```
   sudo apt-key del 7fa2af80
   ```

3. Choose an installation method: *local repo* or *network repo*.

6.8.2. Local Repo Installation for Ubuntu

1. **Install local repository on file system:**

   ```
   sudo dpkg -i cuda-repo-<distro>-X-Y-local_<version>*_<arch>.deb
   ```

   where `<distro>` should be replaced by one of the following:

   - `ubuntu2004`
   - `ubuntu2204`
   - `ubuntu2404`

   and `<arch>` should be replaced by one of the following:

   - `amd64`
   - `arm64`

2. **Enroll ephemeral public GPG key:**

   ```
   sudo cp /var/cuda-repo-<distro>-X-Y-local/cuda-*-keyring.gpg /usr/share/keyrings/
   ```

3. **Add pin file to prioritize CUDA repository:**

```
wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/<arch>/
↪cuda-<distro>.pin
sudo mv cuda-<distro>.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

6.8.3. Network Repo Installation for Ubuntu

The new GPG public key for the CUDA repository is 3bf863cc. This must be enrolled on the system, either using the `cuda-keyring` package or manually; the `apt-key` command is deprecated and not recommended.

1. **Install the new cuda-keyring package:**

   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/<arch>/
   ↪cuda-keyring_1.1-1_all.deb
   sudo dpkg -i cuda-keyring_1.1-1_all.deb
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `ubuntu2004/arm64`
   - `ubuntu2004/sbsa`
   - `ubuntu2004/x86_64`
   - `ubuntu2204/sbsa`
   - `ubuntu2204/x86_64`
   - `ubuntu2404/sbsa`
   - `ubuntu2404/x86_64`

   Note: arm64-Jetson repo:

   - native: `<distro>/arm64`

   ```
   sudo dpkg -i cuda-keyring_1.1-1_all.deb
   ```

6.8.4. Common Installation Instructions for Ubuntu

These instructions apply to both local and network installation for Ubuntu.

1. **Update the Apt repository cache:**

   ```
   sudo apt-get update
   ```

2. **Install CUDA SDK:**

   Note: These two commands must be executed separately.

   ```
   sudo apt-get install cuda-toolkit
   ```

To include all GDS packages:

```
sudo apt-get install nvidia-gds
```

a. **For native arm64-Jetson repos, install the additional packages:**

   ```
   sudo apt-get install cuda-compat
   ```

3. **Reboot the system**

   ```
   sudo reboot
   ```

4. Perform the *Post-installation Actions*.

6.9. Debian

6.9.1. Prepare Debian

1. Perform the *pre-installation actions*.
2. **Enable the contrib repository:**

   ```
   sudo add-apt-repository contrib
   ```

3. **Remove Outdated Signing Key:**

   ```
   sudo apt-key del 7fa2af80
   ```

4. Choose an installation method: *local repo* or *network repo*.

6.9.2. Local Repo Installation for Debian

1. **Install local repository on file system:**

   ```
   sudo dpkg -i cuda-repo-<distro>-X-Y-local_<version>*_amd64.deb
   ```

   where `<distro>` should be replaced by one of the following:

   - debian11
   - debian12

2. **Enroll ephemeral public GPG key:**

   ```
   sudo cp /var/cuda-repo-<distro>-X-Y-local/cuda-*-keyring.gpg /usr/share/keyrings/
   ```

6.9.3. Network Repo Installation for Debian

The new GPG public key for the CUDA repository (Debian-based distros) is 3bf863cc. This must be enrolled on the system, either using the cuda-keyring package or manually; the `apt-key` command is deprecated and not recommended.

1. **Install the new cuda-keyring package:**

   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/<arch>/
   ↪cuda-keyring_1.1-1_all.deb
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `debian11/x86_64`
   - `debian12/x86_64`

   ```
   sudo dpkg -i cuda-keyring_1.1-1_all.deb
   ```

6.9.4. Common Installation Instructions for Debian

These instructions apply to both local and network installation for Debian.

1. **Update the Apt repository cache:**

   ```
   sudo apt-get update
   ```

   Note: If you are using Debian 11, you may instead need to run:

   ```
   sudo apt-get --allow-releaseinfo-change update
   ```

2. **Install CUDA SDK:**

   ```
   sudo apt-get install cuda-toolkit
   ```

3. **Reboot the system:**

   ```
   sudo reboot
   ```

4. Perform the *post-installation actions*.

6.10. Amazon Linux

6.10.1. Prepare Amazon Linux

1. Perform the *pre-installation actions*.
2. Choose an installation method: *local repo* or *network repo*.

6.10.2. Local Repo Installation for Amazon Linux

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.x86_64.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - `amzn2023`

6.10.3. Network Repo Installation for Amazon Linux

1. **Enable the network repository:**

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/x86_64/cuda-<distro>.repo
   ```

2. **Clean DNF repository:**

   ```
   sudo dnf clean all
   ```

6.10.4. Common Installation Instructions for Amazon Linux

These instructions apply to both local and network installation for Amazon Linux.

1. **Install CUDA SDK:**

   ```
   sudo dnf install cuda-toolkit
   ```

2. **Install GPUDirect Filesystem:**

   ```
   sudo dnf install nvidia-gds
   ```

3. **Add libcuda.so symbolic link, if necessary:**

   The `libcuda.so` library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use `libcuda.so`, it may be useful to add a symbolic link from `libcuda.so` in the `/usr/lib{,64}` directory.

4. **Reboot the system:**

   ```
   sudo reboot
   ```

5. Perform the *post-installation actions*.

6.11. Azure Linux CM2

6.11.1. Prepare Azure Linux CM2

1. Perform the *pre-installation actions*.
2. Choose an installation method: *local repo* or *network repo*.

6.11.2. Local Repo Installation for Azure Linux

1. **Install local repository on file system:**

   ```
   sudo rpm --install cuda-repo-<distro>-X-Y-local-<version>*.x86_64.rpm
   ```

   where `<distro>` should be replaced by one of the following:

   - cm2

6.11.3. Network Repo Installation for Azure Linux

1. **Enable the network repository:**

   ```
   curl https://developer.download.nvidia.com/compute/cuda/repos/<distro>/x86_64/
   ↪cuda-<distro>.repo | sudo tee /etc/yum.repos.d/cuda-<distro>.repo
   ```

2. **Clean TDNF repository cache:**

   ```
   sudo tdnf clean expire-cache
   ```

6.11.4. Common Installation Instructions for Azure Linux

These instructions apply to both local and network installation for Azure Linux.

1. **Enable Mariner extended repo:**

   ```
   sudo tdnf install mariner-repos-extended
   ```

2. **Install Cuda SDK:**

   ```
   sudo tdnf install cuda-toolkit
   ```

3. **Install GPUDirect Filesystem:**

   ```
   sudo tdnf install nvidia-gds
   ```

4. **Reboot the system:**

```
sudo reboot
```

5. Perform the *post-installation actions <post-installation-actions>*.

6.12. Additional Package Manager Capabilities

Below are some additional capabilities of the package manager that users can take advantage of.

6.12.1. Available Packages

The recommended installation package is the cuda package. This package will install the full set of other CUDA packages required for native development and should cover most scenarios.

The cuda package installs all the available packages for native developments. That includes the compiler, the debugger, the profiler, the math libraries, and so on. For x86_64 platforms, this also includes Nsight Eclipse Edition and the visual profilers.

On supported platforms, the `cuda-cross-aarch64` and `cuda-cross-sbsa` packages install all the packages required for cross-platform development to arm64-Jetson and arm64-Server, respectively.

Note: 32-bit compilation native and cross-compilation is removed from CUDA 12.0 and later Toolkit. Use the CUDA Toolkit from earlier releases for 32-bit compilation. Hopper does not support 32-bit applications.

The packages installed by the packages above can also be installed individually by specifying their names explicitly. The list of available packages be can obtained with:

```
dnf --disablerepo="*" --enablerepo="cuda*" list available    # Amazon Linux / Fedora /
↪ KylinOS / RHEL / Rocky Linux
```

```
tdnf --disablerepo="*" --enablerepo="cuda-cm2-<cuda X-Y version>-local" list
↪available   # Azure Linux
```

```
zypper packages -r cuda                                      # OpenSUSE / SLES
```

```
cat /var/lib/apt/lists/*cuda*Packages | grep "Package:"      # Debian / Ubuntu
```

6.12.2. Meta Packages

Meta packages are RPM/Deb/Conda packages which contain no (or few) files but have multiple dependencies. They are used to install many CUDA packages when you may not know the details of the packages you want. The following table lists the meta packages.

Table 4: Meta Packages Available for CUDA 12.9

Meta Package	Purpose
cuda	Installs all CUDA Toolkit and Driver packages. Handles upgrading to the next version of the cuda package when it's released.
cuda-12-8	Installs all CUDA Toolkit and Driver packages. Remains at version 12.5 until an additional version of CUDA is installed.
cuda-toolkit-12-8	Installs all CUDA Toolkit packages required to develop CUDA applications. Does not include the driver.
cuda-toolkit-12	Installs all CUDA Toolkit packages required to develop applications. Will not upgrade beyond the 12.x series toolkits. Does not include the driver.
cuda-toolkit	Installs all CUDA Toolkit packages required to develop applications. Handles upgrading to the next 12.x version of CUDA when it's released. Does not include the driver.
cuda-tools-12-8	Installs all CUDA command line and visual tools.
cuda-runtime-12-8	Installs all CUDA Toolkit packages required to run CUDA applications, as well as the Driver packages.
cuda-compiler-12-8	Installs all CUDA compiler packages.
cuda-libraries-12-8	Installs all runtime CUDA Library packages.
cuda-libraries-dev-12-8	Installs all development CUDA Library packages.

6.12.3. Package Upgrades

The cuda package points to the latest stable release of the CUDA Toolkit. When a new version is available, use the following commands to upgrade the toolkit:

6.12.3.1 Amazon Linux

```
sudo dnf upgrade cuda-toolkit
```

6.12.3.2 Fedora

When upgrading the toolkit to a new **major** branch:

```
sudo dnf install cuda-toolkit
```

When upgrading the toolkit to a new **minor** branch:

```
sudo dnf upgrade cuda-toolkit
```

6.12.3.3 KylinOS / RHEL / Rocky Linux

```
sudo dnf install cuda-toolkit
```

6.12.3.4 Azure Linux

```
sudo tdnf install cuda-toolkit
```

6.12.3.5 OpenSUSE / SLES

```
sudo zypper install cuda-toolkit
```

6.12.3.6 Debian / Ubuntu

```
sudo apt-get install cuda-toolkit
```

6.12.3.7 Other Package Notes

The `cuda-cross-<arch>` packages can also be upgraded in the same manner.

Some desktop environments, such as GNOME or KDE, will display a notification alert when new packages are available.

To avoid any automatic upgrade, and lock down the toolkit installation to the X.Y release, install the `cuda-toolkit-X-Y` or `cuda-cross-<arch>-X-Y` package.

Side-by-side installations are supported. For instance, to install both the X.Y CUDA Toolkit and the X.Y+1 CUDA Toolkit, install the `cuda-toolkit-X.Y` and `cuda-toolkit-X.Y+1` packages.

Chapter 7. Driver Installation

More information about driver installation can be found in the Driver Installation Guide for Linux

Chapter 8. Runfile Installation

Basic instructions can be found in the Quick Start Guide. Read on for more detailed instructions.

This section describes the installation and configuration of CUDA when using the standalone installer. The standalone installer is a `.run` file and is completely self-contained.

8.1. Runfile Overview

The Runfile installation installs the CUDA Toolkit via an interactive ncurses-based interface.

The *installation steps* are listed below.

Finally, *advanced options* for the installer and *uninstallation steps* are detailed below.

The Runfile installation does not include support for cross-platform development. For cross-platform development, see the *CUDA Cross-Platform Environment* section.

8.2. Installation

1. Perform the *pre-installation actions*.
2. Reboot into text mode (runlevel 3).

   This can usually be accomplished by adding the number "3" to the end of the system's kernel boot parameters.

   Since the NVIDIA drivers are not yet installed, the text terminals may not display correctly. Temporarily adding "nomodeset" to the system's kernel boot parameters may fix this issue.

   Consult your system's bootloader documentation for information on how to make the above boot parameter changes.
3. Run the installer and follow the on-screen prompts:

   ```
   sudo sh cuda_<version>_linux.run
   ```

   The installer will prompt for the following:

   - EULA Acceptance
   - CUDA Toolkit installation, location, and `/usr/local/cuda` symbolic link

The default installation location for the toolkit is `/usr/local/cuda-12.6`:

The `/usr/local/cuda` symbolic link points to the location where the CUDA Toolkit was installed. This link allows projects to use the latest CUDA Toolkit without any configuration file update.

The installer must be executed with sufficient privileges to perform some actions. When the current privileges are insufficient to perform an action, the installer will ask for the user's password to attempt to install with root privileges. Actions that cause the installer to attempt to install with root privileges are:

- installing the CUDA Toolkit to a location the user does not have permission to write to
- creating the `/usr/local/cuda` symbolic link

Running the installer with **sudo**, as shown above, will give permission to install to directories that require root permissions. Directories and files created while running the installer with **sudo** will have root ownership.

4. Reboot the system to reload the graphical interface:

```
sudo reboot
```

5. Perform the *post-installation actions*.

8.3. Advanced Options

Action	Options Used	Explanation
Silent Installation	`--silent`	Required for any silent installation. Performs an installation with no further user-input and minimal command-line output based on the options provided below. Silent installations are useful for scripting the installation of CUDA. Using this option implies acceptance of the EULA. The following flags can be used to customize the actions taken during installation. At least one of `--driver`, `--uninstall`, and `--toolkit` must be passed if running with non-root permissions.
	`--driver`	Install the CUDA Driver.
	`--toolkit`	Install the CUDA Toolkit.
	`--toolkitpath=<path>`	Install the CUDA Toolkit to the <path> directory. If not provided, the default path of `/usr/local/cuda-12.6` is used.
	`--defaultroot=<path>`	Install libraries to the <path> directory. If the <path> is not provided, then the default path of your distribution is used. *This only applies to the libraries installed outside of the CUDA Toolkit path.*
Extraction	`--extract=<path>`	Extracts to the <path> the following: the driver runfile, the raw files of the toolkit to <path>. This is especially useful when one wants to install the driver using one or more of the command-line options provided by the driver installer which are not exposed in this installer.
Overriding Installation Checks	`--override`	Ignores compiler, third-party library, and toolkit detection checks which would prevent the CUDA Toolkit from installing.
No OpenGL Libraries	`--no-opengl-libs`	Prevents the driver installation from installing NVIDIA's GL libraries. Useful for systems where the display is driven by a non-NVIDIA GPU. In such systems, NVIDIA's GL libraries could prevent X from loading properly.
No man pages	`--no-man-page`	Do not install the man pages under `/usr/share/man`.
Overriding Kernel Source	`--kernel-source-path=<path>`	Tells the driver installation to use <path> as the kernel source directory when building the NVIDIA kernel module. Required for systems where the kernel source is installed to a non-standard location.
Running nvidia-xconfig	`--run-nvidia-xconfig`	Tells the driver installation to run nvidia-xconfig to update the system X configuration file so that the NVIDIA X driver is used. The pre-existing X configuration file will be backed

8.4. Uninstallation

To uninstall the CUDA Toolkit, run the uninstallation script provided in the bin directory of the toolkit. By default, it is located in `/usr/local/cuda-12.6/bin`:

```
sudo /usr/local/cuda-12.6/bin/cuda-uninstaller
```

Chapter 9. Conda Installation

This section describes the installation and configuration of CUDA when using the Conda installer. The Conda packages are available at https://anaconda.org/nvidia.

9.1. Conda Overview

The Conda installation installs the CUDA Toolkit. The installation steps are listed below.

9.2. Installing CUDA Using Conda

To perform a basic install of all CUDA Toolkit components using Conda, run the following command:

```
conda install cuda -c nvidia
```

Note: Install CUDA in a dedicated Conda environment instead of the base environment to avoid installation issues.

9.3. Uninstalling CUDA Using Conda

To uninstall the CUDA Toolkit using Conda, run the following command:

```
conda remove cuda
```

9.4. Installing Previous CUDA Releases

All Conda packages released under a specific CUDA version are labeled with that release version. To install a previous version, include that label in the `install` command such as:

```
conda install cuda -c nvidia/label/cuda-12.4.0
```

9.5. Upgrading from cudatoolkit Package

If you had previously installed CUDA using the `cudatoolkit` package and want to maintain a similar install footprint, you can limit your installation to the following packages:

- `cuda-libraries-dev`
- `cuda-nvcc`
- `cuda-nvtx`
- `cuda-cupti`

Note: Some extra files, such as headers, will be included in this installation which were not included in the `cudatoolkit` package. If you need to reduce your installation further, replace `cuda-libraries-dev` with the specific libraries you need.

Chapter 10. Pip Wheels

NVIDIA provides Python Wheels for installing CUDA through pip, primarily for using CUDA with Python. These packages are intended for runtime use and do not currently include developer tools (these can be installed separately).

Please note that with this installation method, CUDA installation environment is managed via pip and additional care must be taken to set up your host environment to use CUDA outside the pip environment.

10.1. Prerequisites

To install Wheels, you must first install the `nvidia-pyindex` package, which is required in order to set up your pip installation to fetch additional Python modules from the NVIDIA NGC PyPI repo. If your pip and setuptools Python modules are not up-to-date, then use the following command to upgrade these Python modules. If these Python modules are out-of-date then the commands which follow later in this section may fail.

```
python3 -m pip install --upgrade setuptools pip wheel
```

You should now be able to install the `nvidia-pyindex` module.

```
python3 -m pip install nvidia-pyindex
```

If your project is using a `requirements.txt` file, then you can add the following line to your `requirements.txt` file as an alternative to installing the `nvidia-pyindex` package:

```
--extra-index-url https://pypi.org/simple
```

10.2. Procedure

Install the CUDA runtime package:

```
python3 -m pip install nvidia-cuda-runtime-cu12
```

Optionally, install additional packages as listed below using the following command:

```
python3 -m pip install nvidia-<library>
```

10.3. Metapackages

The following metapackages will install the latest version of the named component on Linux for the indicated CUDA version. “cu12” should be read as “cuda12”.

- nvidia-cublas-cu12
- nvidia-cuda-cccl-cu12
- nvidia-cuda-cupti-cu12
- nvidia-cuda-nvcc-cu12
- nvidia-cuda-nvrtc-cu12
- nvidia-cuda-opencl-cu12
- nvidia-cuda-runtime-cu12
- nvidia-cuda-sanitizer-api-cu12
- nvidia-cufft-cu12
- nvidia-curand-cu12
- nvidia-cusolver-cu12
- nvidia-cusparse-cu12
- nvidia-npp-cu12
- nvidia-nvfatbin-cu12
- nvidia-nvjitlink-cu12
- nvidia-nvjpeg-cu12
- nvidia-nvml-dev-cu12
- nvidia-nvtx-cu12

These metapackages install the following packages:

- nvidia-cublas-cu129
- nvidia-cuda-cccl-cu129
- nvidia-cuda-cupti-cu129
- nvidia-cuda-nvcc-cu129
- nvidia-cuda-nvrtc-cu129
- nvidia-cuda-opencl-cu129
- nvidia-cuda-runtime-cu129
- nvidia-cuda-sanitizer-api-cu129
- nvidia-cufft-cu129
- nvidia-curand-cu129
- nvidia-cusolver-cu129
- nvidia-cusparse-cu129
- nvidia-npp-cu129

- nvidia-nvfatbin-cu129
- nvidia-nvjitlink-cu129
- nvidia-nvjpeg-cu129
- nvidia-nvml-dev-cu129
- nvidia-nvtx-cu129

Chapter 11. CUDA Cross-Platform Environment

Cross development for arm64-sbsa is supported on Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04, KylinOS 10, RHEL 8, RHEL 9, and SLES 15.

Cross development for arm64-Jetson is only supported on Ubuntu 22.04

We recommend selecting a host development environment that matches the supported cross-target environment. This selection helps prevent possible host/target incompatibilities, such as GCC or GLIBC version mismatches.

11.1. CUDA Cross-Platform Installation

Some of the following steps may have already been performed as part of the *native installation sections*. Such steps can safely be skipped.

These steps should be performed on the x86_64 host system, rather than the target system. To install the native CUDA Toolkit on the target system, refer to the native installation sections in *Package Manager Installation*.

11.1.1. Ubuntu

1. Perform the *pre-installation actions.*
2. Choose an installation method: *local repo* or *network repo*.

11.1.1.1 Local Cross Repo Installation for Ubuntu

1. Install repository meta-data package with:

   ```
   sudo dpkg -i cuda-repo-cross-<arch>-<distro>-X-Y-local-<version>*_all.deb
   ```

   where `<arch>-<distro>` should be replaced by one of the following:

   - `aarch64-ubuntu2204`
   - `sbsa-ubuntu2004`
   - `sbsa-ubuntu2204`

- sbsa-ubuntu2404

11.1.1.2 Network Cross Repo Installation for Ubuntu

The new GPG public key for the CUDA repository is 3bf863cc. This must be enrolled on the system, either using the `cuda-keyring` package or manually; the `apt-key` command is deprecated and not recommended.

1. Install the new cuda-keyring package:

   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/<arch>/
   ↪cuda-keyring_1.1-1_all.deb
   sudo dpkg -i cuda-keyring_1.1-1_all.deb
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - ubuntu2004/cross-linux-sbsa
   - ubuntu2204/cross-linux-aarch64
   - ubuntu2204/cross-linux-sbsa
   - ubuntu2404/cross-linux-sbsa

11.1.1.3 Common Installation Instructions for Ubuntu

1. Update the Apt repository cache:

   ```
   sudo apt-get update
   ```

2. Install the appropriate cross-platform CUDA Toolkit:

   a. For arm64-sbsa:

      ```
      sudo apt-get install cuda-cross-sbsa
      ```

   b. For arm64-Jetson

      ```
      sudo apt-get install cuda-cross-aarch64
      ```

   c. For QNX:

      ```
      sudo apt-get install cuda-cross-qnx
      ```

3. Perform the *post-installation actions.*

11.1.2. KylinOS / RHEL / Rocky Linux

1. Perform the *pre-installation actions.*
2. Choose an installation method: *local repo* or *network repo*.

11.1.2.1 Local Cross Repo Installation for KylinOS / RHEL / Rocky Linux

1. Install repository meta-data package with:

   ```
   sudo rpm -i cuda-repo-cross-<arch>-<distro>-X-Y-local-<version>*.noarch.rpm
   ```

   where `<arch>-<distro>` should be replaced by one of the following:

   - `sbsa-kylin10`
   - `sbsa-rhel8`
   - `sbsa-rhel9`

11.1.2.2 Network Cross Repo Installation for KylinOS / RHEL / Rocky Linux

1. **Enable the network repo:**

   ```
   sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/
   ↪cuda/repos/<distro>/<arch>/cuda-<distro>-cross-linux-sbsa.repo
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `kylin10/cross-linux-sbsa`
   - `rhel8/cross-linux-sbsa`
   - `rhel9/cross-linux-sbsa`

11.1.2.3 Common Installation Instructions for KylinOS / RHEL / Rocky Linux

1. **Clean DNF repository:**

   ```
   sudo dnf clean all
   ```

2. **Install CUDA tool:**

   ```
   sudo dnf install cuda-cross-sbsa
   ```

11.1.3. SLES

1. Perform the *pre-installation actions.*
2. Choose an installation method: *local repo* or *network repo*.

11.1.3.1 Local Cross Repo Installation for SLES

1. Install repository meta-data package with:

   ```
   sudo rpm -i cuda-repo-cross-<arch>-<distro>-X-Y-local-<version>*.noarch.rpm
   ```

   where `<arch>-<distro>` should be replaced by one of the following:

   - `sbsa-sles15`

11.1.3.2 Network Cross Repo Installation for SLES

1. **Enable the network repo:**

   ```
   sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distro>/<arch>/cuda-<distro>-cross-linux-sbsa.repo
   ```

   where `<distro>/<arch>` should be replaced by one of the following:

   - `sles15/cross-linux-sbsa`

11.1.3.3 Common Installation Instructions for SLES

1. **Refresh Zypper repository cache:**

   ```
   sudo zypper refresh
   ```

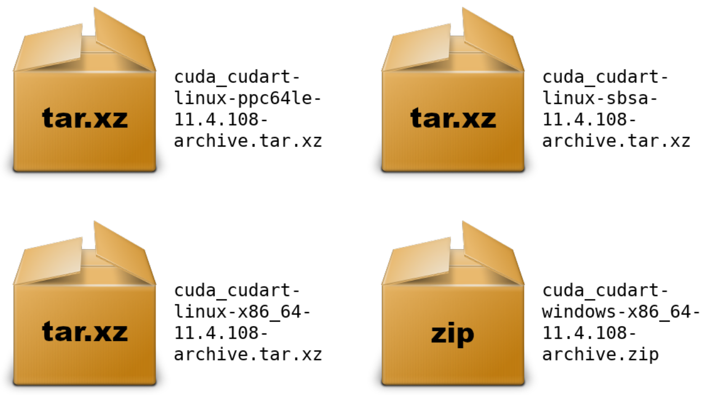
2. **Install CUDA tool:**

   ```
   sudo zypper install cuda-cross-sbsa
   ```

Chapter 12. Tarball and Zip Archive Deliverables

In an effort to meet the needs of a growing customer base requiring alternative installer packaging formats, as well as a means of input into community CI/CD systems, tarball and zip archives are available for each component.

These tarball and zip archives, known as binary archives, are provided at https://developer.download.nvidia.com/compute/cuda/redist/.


tar.xz
cuda_cudart-linux-ppc64le-11.4.108-archive.tar.xz
tar.xz
cuda_cudart-linux-sbsa-11.4.108-archive.tar.xz
tar.xz
cuda_cudart-linux-x86_64-11.4.108-archive.tar.xz
zip
cuda_cudart-windows-x86_64-11.4.108-archive.zip

These component .tar.xz and .zip binary archives do not replace existing packages such as .deb, .rpm, runfile, conda, etc. and are not meant for general consumption, as they are not installers. However this standardized approach will replace existing .txz archives.

For each release, a JSON manifest is provided such as **redistrib_11.4.2.json**, which corresponds to the CUDA 11.4.2 release label (CUDA 11.4 update 2) which includes the release date, the name of each component, license name, relative URL for each platform and checksums.

Package maintainers are advised to check the provided LICENSE for each component prior to redistribution. Instructions for developers using CMake and Bazel build systems are provided in the next sections.

12.1. Parsing Redistrib JSON

The following example of a JSON manifest contains keys for each component: name, license, version, and a platform array which includes relative_path, sha256, md5, and size (bytes) for each archive.

```
{
    "release_date": "2021-09-07",
    "cuda_cudart": {
        "name": "CUDA Runtime (cudart)",
        "license": "CUDA Toolkit",
        "version": "11.4.108",
        "linux-x86_64": {
            "relative_path": "cuda_cudart/linux-x86_64/cuda_cudart-linux-x86_64-11.4.
↪108-archive.tar.xz",
            "sha256":
↪"d08a1b731e5175aa3ae06a6d1c6b3059dd9ea13836d947018ea5e3ec2ca3d62b",
            "md5": "da198656b27a3559004c3b7f20e5d074",
            "size": "828300"
        },
        "linux-ppc64le": {
            "relative_path": "cuda_cudart/linux-ppc64le/cuda_cudart-linux-ppc64le-11.
↪4.108-archive.tar.xz",
            "sha256":
↪"831dffe062ae3ebda3d3c4010d0ee4e40a01fd5e6358098a87bb318ea7c79e0c",
            "md5": "ca73328e3f8e2bb5b1f2184c98c3a510",
            "size": "776840"
        },
        "linux-sbsa": {
            "relative_path": "cuda_cudart/linux-sbsa/cuda_cudart-linux-sbsa-11.4.108-
↪archive.tar.xz",
            "sha256":
↪"2ab9599bbaebdcf59add73d1f1a352ae619f8cb5ccec254093c98efd4c14553c",
            "md5": "aeb5c19661f06b6398741015ba368102",
            "size": "782372"
        },
        "windows-x86_64": {
            "relative_path": "cuda_cudart/windows-x86_64/cuda_cudart-windows-x86_64-
↪11.4.108-archive.zip",
            "sha256":
↪"b59756c27658d1ea87a17c06d064d1336576431cd64da5d1790d909e455d06d3",
            "md5": "7f6837a46b78198402429a3760ab28fc",
            "size": "2897751"
        }
    }
}
```

A JSON schema is provided at https://developer.download.nvidia.com/compute/redist/redistrib-v2.schema.json.

A sample script that parses these JSON manifests is available on GitHub:

- Downloads each archive
- Validates SHA256 checksums
- Extracts archives
- Flattens into a collapsed directory structure

Table 5: Available Tarball and Zip Archives

Product	Example
CUDA Toolkit	`./parse_redist.py --product cuda --label 12.6.0`
cuBLASMp	`./parse_redist.py --product cublasmp --label 0.2.1`
cuDNN	`./parse_redist.py --product cudnn --label 9.2.1`
cuDSS	`./parse_redist.py --product cudss --label 0.3.0`
cuQuantum	`./parse_redist.py --product cuquantum --label 24.03.0`
cuSPARSELt	`./parse_redist.py --product cusparselt --label 0.6.2`
cuTENSOR	`./parse_redist.py --product cutensor --label 2.0.2.1`
NVIDIA driver	`./parse_redist.py --product nvidia-driver --label 550.90.07`
nvJPEG2000	`./parse_redist.py --product nvjpeg2000 --label 0.7.5`
NVPL	`./parse_redist.py --product nvpl --label 24.7`
nvTIFF	`./parse_redist.py --product nvtiff --label 0.3.0`

12.2. Importing Tarballs into CMake

The recommended module for importing these tarballs into the CMake build system is via FindCUDA-Toolkit (3.17 and newer).

Note: The FindCUDA module is deprecated.

The path to the extraction location can be specified with the `CUDAToolkit_ROOT` environmental variable. For example `CMakeLists.txt` and commands, see cmake/1_FindCUDAToolkit/.

For older versions of CMake, the ExternalProject_Add module is an alternative method. For example `CMakeLists.txt` file and commands, see cmake/2_ExternalProject/.

12.3. Importing Tarballs into Bazel

The recommended method of importing these tarballs into the Bazel build system is using http_archive and pkg_tar.

For an example, see bazel/1_pkg_tar/.

Chapter 13. Post-installation Actions

The post-installation actions must be manually performed. These actions are split into mandatory, recommended, and optional sections.

13.1. Mandatory Actions

Some actions must be taken after the installation before the CUDA Toolkit can be used.

13.1.1. Environment Setup

The PATH variable needs to include `export PATH=/usr/local/cuda-12.9/bin${PATH:+:${PATH}}`. Nsight Compute has moved to `/opt/nvidia/nsight-compute/` only in rpm/deb installation method. When using `.run` installer it is still located under `/usr/local/cuda-12.6/`.

To add this path to the PATH variable:

```
export PATH=/usr/local/cuda-12.6/bin${PATH:+:${PATH}}
```

In addition, when using the runfile installation method, the LD_LIBRARY_PATH variable needs to contain `/usr/local/cuda-12.9/lib64` on a 64-bit system, or `/usr/local/cuda-12.9/lib` on a 32-bit system

- To change the environment variables for 64-bit operating systems:

  ```
  export LD_LIBRARY_PATH=/usr/local/cuda-12.6/lib64\
                           ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
  ```

- To change the environment variables for 32-bit operating systems:

  ```
  export LD_LIBRARY_PATH=/usr/local/cuda-12.6/lib\
                           ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
  ```

Note that the above paths change when using a custom install path with the runfile installation method.

13.2. Recommended Actions

Other actions are recommended to verify the integrity of the installation.

13.2.1. Install Writable Samples

CUDA Samples are now located in https://github.com/nvidia/cuda-samples, which includes instructions for obtaining, building, and running the samples.

13.2.2. Verify the Installation

Before continuing, it is important to verify that the CUDA toolkit can find and communicate correctly with the CUDA-capable hardware. To do this, you need to compile and run some of the sample programs, located in https://github.com/nvidia/cuda-samples.

Note: Ensure the PATH and, if using the runfile installation method, `LD_LIBRARY_PATH` variables are set correctly.

13.2.2.1 Running the Binaries

After compilation, find and run `deviceQuery`from https://github.com/nvidia/cuda-samples. If the CUDA software is installed and configured correctly, the output for `deviceQuery` should look similar to that shown in Figure 1.

The exact appearance and the output lines might be different on your system. The important outcomes are that a device was found (the first highlighted line), that the device matches the one on your system (the second highlighted line), and that the test passed (the final highlighted line).

If a CUDA-capable device is installed but `deviceQuery` reports that no CUDA-capable devices are present, this likely means that the `/dev/nvidia*` files are missing or have the wrong permissions.

On systems where `SELinux` is enabled, you might need to temporarily disable this security feature to run `deviceQuery`. To do this, type:

```
setenforce 0
```

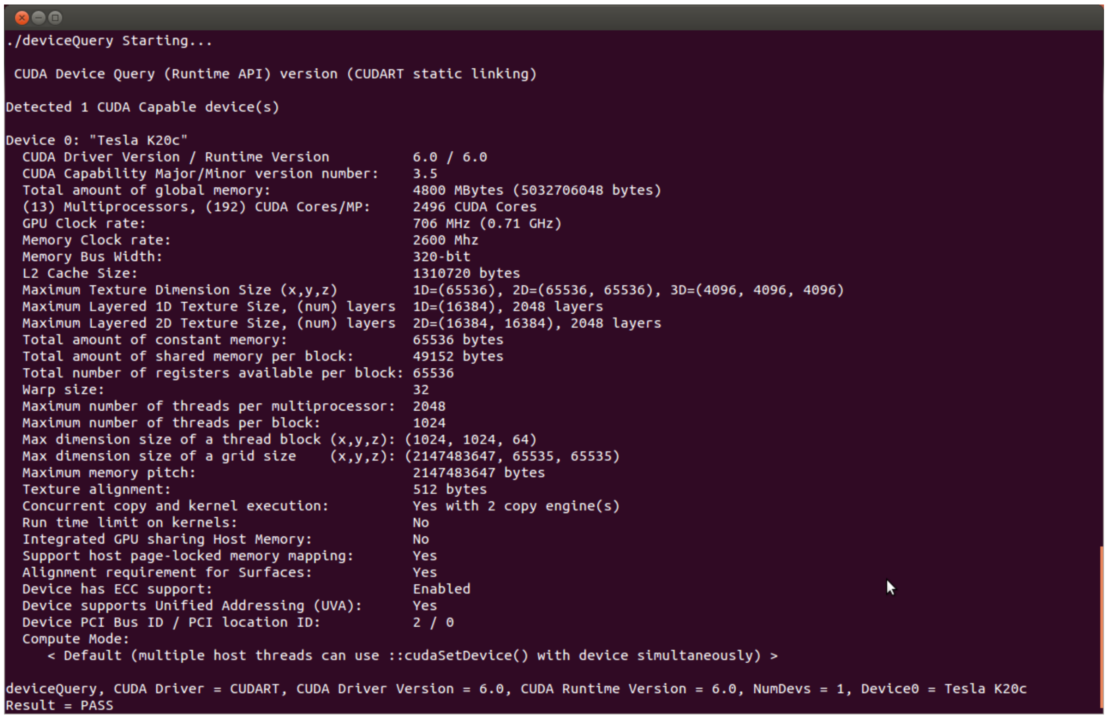
from the command line as the superuser.

Running the `bandwidthTest` program ensures that the system and the CUDA-capable device are able to communicate correctly. Its output is shown in *Figure 2*.

Note that the measurements for your CUDA-capable device description will vary from system to system. The important point is that you obtain measurements, and that the second-to-last line (in *Figure 2*) confirms that all necessary tests passed.

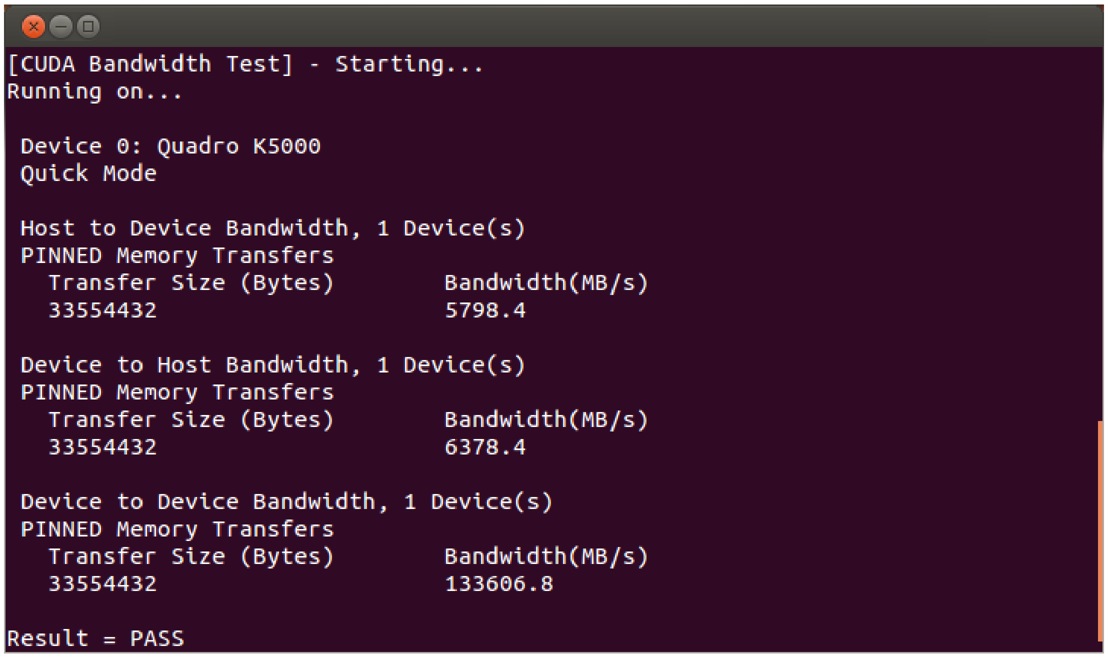
Should the tests not pass, make sure you have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

If you run into difficulties with the link step (such as libraries not being found), consult the Linux Release Notes found in https://github.com/nvidia/cuda-samples.



```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla K20c"
  CUDA Driver Version / Runtime Version          6.0 / 6.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 4800 MBytes (5032706048 bytes)
  (13) Multiprocessors, (192) CUDA Cores/MP:     2496 CUDA Cores
  GPU Clock rate:                                706 MHz (0.71 GHz)
  Memory Clock rate:                             2600 Mhz
  Memory Bus Width:                              320-bit
  L2 Cache Size:                                 1310720 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.0, CUDA Runtime Version = 6.0, NumDevs = 1, Device0 = Tesla K20c
Result = PASS
```

Figure 1: Valid Results from deviceQuery CUDA Sample



```
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: Quadro K5000
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     5798.4

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     6378.4

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     133606.8

Result = PASS
```

Figure 2: Valid Results from bandwidthTest CUDA Sample

13.2.3. Install Nsight Eclipse Plugins

To install Nsight Eclipse plugins, an installation script is provided:

```
/usr/local/cuda-12.6/bin/nsight_ee_plugins_manage.sh install <eclipse-dir>
```

Refer to Nsight Eclipse Plugins Installation Guide for more details.

13.2.4. Local Repo Removal

Removal of the local repo installer is recommended after installation of **CUDA SDK**.

Debian / Ubuntu

```
sudo apt-get remove --purge "cuda-repo-<distro>-X-Y-local*"
```

Amazon Linux / Fedora / KylinOS / RHEL / Rocky Linux

```
sudo dnf remove "cuda-repo-<distro>-X-Y-local*"
```

Azure Linux

```
sudo tdnf remove "cuda-repo-<distro>-X-Y-local*"
```

OpenSUSE / SLES

```
sudo zypper remove "cuda-repo-<distro>-X-Y-local*"
```

13.3. Optional Actions

Other options are not necessary to use the CUDA Toolkit, but are available to provide additional features.

13.3.1. Install Third-party Libraries

Some CUDA samples use third-party libraries which may not be installed by default on your system. These samples attempt to detect any required libraries when building.

If a library is not detected, it waives itself and warns you which library is missing. To build and run these samples, you must install the missing libraries. In cases where these dependencies are not installed, follow the instructions below.

Amazon Linux / Fedora / KylinOS / RHEL / Rocky Linux

```
sudo dnf install freeglut-devel libX11-devel libXi-devel libXmu-devel \
    make mesa-libGLU-devel freeimage-devel libglfw3-devel
```

SLES

```
sudo zypper install libglut3 libX11-devel libXi6 libXmu6 libGLU1 make
```

OpenSUSE

```
sudo zypper install freeglut-devel libX11-devel libXi-devel libXmu-devel \
    make Mesa-libGL-devel freeimage-devel
```

Debian / Ubuntu

```
sudo apt-get install g++ freeglut3-dev build-essential libx11-dev \
    libxmu-dev libxi-dev libglu1-mesa-dev libfreeimage-dev libglfw3-dev
```

13.3.2. Install the Source Code for cuda-gdb

The `cuda-gdb` source must be explicitly selected for installation with the runfile installation method. During the installation, in the component selection page, expand the component “CUDA Tools 12.9” and select `cuda-gdb-src` for installation. It is unchecked by default.

To obtain a copy of the source code for `cuda-gdb` using the RPM and Debian installation methods, the `cuda-gdb-src` package must be installed.

The source code is installed as a tarball in the `/usr/local/cuda-12.9/extras` directory.

13.3.3. Select the Active Version of CUDA

For applications that rely on the symlinks `/usr/local/cuda` and `/usr/local/cuda-MAJOR`, you may wish to change to a different installed version of CUDA using the provided alternatives.

To show the active version of CUDA and all available versions:

```
update-alternatives --display cuda
```

To show the active minor version of a given major CUDA release:

```
update-alternatives --display cuda-12
```

To update the active version of CUDA:

```
sudo update-alternatives --config cuda
```

Chapter 14. Removing CUDA Toolkit

Follow the below steps to properly uninstall the CUDA Toolkit from your system. These steps will ensure that the uninstallation will be clean.

Amazon Linux / Fedora / Kylin OS / RHEL / Rocky Linux

To remove CUDA Toolkit:

```
sudo dnf remove "cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*" \
                    "*cusolver*" "*cusparse*" "*gds-tools*" "*npp*" "*nvjpeg*"
↪"nsight*" "*nvvm*"
```

Azure Linux

To remove CUDA Toolkit:

```
sudo tdnf remove "cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*" "*cusolver*"
↪"*cusparse*" "*gds-tools*" "*npp*" "*nvjpeg*" "nsight*" "*nvvm*"
```

To clean up the uninstall:

```
sudo tdnf autoremove
```

OpenSUSE / SLES

To remove CUDA Toolkit:

```
sudo zypper remove "cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*" \
 "*cusolver*" "*cusparse*" "*gds-tools*" "*npp*" "*nvjpeg*" "nsight*" "*nvvm*"
```

Debian / Ubuntu

To remove CUDA Toolkit:

```
sudo apt-get --purge remove "*cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*" \
 "*cusolver*" "*cusparse*" "*gds-tools*" "*npp*" "*nvjpeg*" "nsight*" "*nvvm*"
```

To clean up the uninstall:

```
sudo apt-get autoremove --purge -V
```

Chapter 15. Advanced Setup

Below is information on some advanced setup scenarios which are not covered in the basic instructions above.

Table 6: Advanced Setup Scenarios when Installing CUDA

Scenario	Instructions
Install GPUDirect Storage	Refer to Installing GPUDirect Storage. GDS is supported in two different modes: ▶ GDS (default/full perf mode) ▶ Compatibility mode. Installation instructions for them differ slightly. Compatibility mode is the only mode that is supported on certain distributions due to software dependency limitations. Full GDS support is restricted to the following Linux distros: ▶ Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04 ▶ RHEL 8.y (y <= 10), RHEL 9.y (y <= 5)

Install CUDA to a specific directory using the Package Manager installation method.	**RPM**
	The RPM packages don't support custom install locations through the package managers (Yum and Zypper), but it is possible to install the RPM packages to a custom location using rpm's `--relocate` parameter:
	`sudo rpm --install --relocate /usr/local/cuda-12.6=/new/toolkit package.rpm`
	You will need to install the packages in the correct dependency order; this task is normally taken care of by the package managers. For example, if package "foo" has a dependency on package "bar", you should install package "bar" first, and package "foo" second. You can check the dependencies of a RPM package as follows:
	`rpm -qRp package.rpm`
	Note that the driver packages cannot be relocated.
	Deb
	The Deb packages do not support custom install locations. It is however possible to extract the contents of the Deb packages and move the files to the desired install location. See the next scenario for more details one xtracting Deb packages.

Extract the contents of the installers.	**Runfile**
	The Runfile can be extracted into the standalone Toolkit Runfiles by using the `--extract` parameter. The Toolkit standalone Runfiles can be further extracted by running:
	`./runfile.run --tar mxvf`
	`./runfile.run -x`
	RPM
	The RPM packages can be extracted by running:
	`rpm2cpio package.rpm | cpio -idmv`
	Deb
	The Deb packages can be extracted by running:
	`dpkg-deb -x package.deb output_dir`

Chapter 16. Additional Considerations

Now that you have CUDA-capable hardware and the NVIDIA CUDA Toolkit installed, you can examine and enjoy the numerous included programs. To begin using CUDA to accelerate the performance of your own applications, consult the CUDA C++ Programming Guide, located in `/usr/local/cuda-12.9/doc`.

A number of helpful development tools are included in the CUDA Toolkit to assist you as you develop your CUDA programs, such as NVIDIA® Nsight™ Eclipse Edition, NVIDIA Visual Profiler, CUDA-GDB, and CUDA-MEMCHECK.

For technical support on programming questions, consult and participate in the developer forums at https://forums.developer.nvidia.com/c/accelerated-computing/cuda/206.

Chapter 17. Frequently Asked Questions

17.1. How do I install the Toolkit in a different location?

The Runfile installation asks where you wish to install the Toolkit during an interactive install. If installing using a non-interactive install, you can use the `--toolkitpath` parameter to change the install location:

```
./runfile.run --silent \
                --toolkit --toolkitpath=/my/new/toolkit
```

The RPM and Deb packages cannot be installed to a custom install location directly using the package managers. See the "Install CUDA to a specific directory using the Package Manager installation method" scenario in the *Advanced Setup* section for more information.

17.2. Why do I see "nvcc: No such file or directory" when I try to build a CUDA application?

Your PATH environment variable is not set up correctly. Ensure that your PATH includes the bin directory where you installed the Toolkit, usually `/usr/local/cuda-12.9/bin`.

```
export PATH=/usr/local/cuda-12.6/bin${PATH:+:${PATH}}
```

17.3. Why do I see "error while loading shared libraries: <lib name>: cannot open shared object file: No such file or directory" when I try to run a CUDA application that uses a CUDA library?

Your LD_LIBRARY_PATH environment variable is not set up correctly. Ensure that your LD_LIBRARY_PATH includes the lib and/or lib64 directory where you installed the Toolkit, usually `/usr/local/cuda-12.9/lib{,64}`:

```
export LD_LIBRARY_PATH=/usr/local/cuda-12.9/lib\
                         ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

17.4. Why do I see multiple "404 Not Found" errors when updating my repository meta-data on Ubuntu?

These errors occur after adding a foreign architecture because apt is attempting to query for each architecture within each repository listed in the system's sources.list file. Repositories that do not host packages for the newly added architecture will present this error. While noisy, the error itself does no harm. Please see the *Advanced Setup* section for details on how to modify your `sources.list` file to prevent these errors.

17.5. How can I tell X to ignore a GPU for compute-only use?

To make sure X doesn't use a certain GPU for display, you need to specify which **other** GPU to use for display. For more information, please refer to the "Use a specific GPU for rendering the display" scenario in the Advanced Setup section.

17.6. Why doesn't the cuda-repo package install the CUDA Toolkit?

When using RPM or Deb, the downloaded package is a repository package. Such a package only informs the package manager where to find the actual installation packages, but will not install them.

See the *Package Manager Installation* section for more details.

17.7. How do I install an older CUDA version using a network repo?

Depending on your system configuration, you may not be able to install old versions of CUDA using the cuda metapackage. In order to install a specific version of CUDA, you may need to specify all of the packages that would normally be installed by the cuda metapackage at the version you want to install.

If you are using yum to install certain packages at an older version, the dependencies may not resolve as expected. In this case you may need to pass "`--setopt=obsoletes=0`" to yum to allow an install of packages which are obsoleted at a later version than you are trying to install.

17.8. How do I handle "Errors were encountered while processing: glx-diversions"?

This sometimes occurs when trying to uninstall CUDA after a clean .deb installation. Run the following commands:

```
sudo apt-get install glx-diversions --reinstall
sudo apt-get remove nvidia-alternative
```

Then re-run the commands from *Removing CUDA Toolkit*.

Chapter 18. Notices

18.1. Notice

18.2. OpenCL

18.3. Trademarks

Chapter 19. Copyright

Copyright