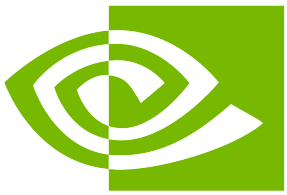


NVIDIA Driver Installation Guide

Release r575

NVIDIA Corporation

May 28, 2025

Contents

Driver Installation Guide

The installation instructions for the NVIDIA Driver on Linux.

Chapter 1. System Requirements

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

- NVIDIA GPU
- A supported version of Linux with a `gcc` compiler and toolchain

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

The columns with `$distro`, `$arch`, and `$arch_ext` can be used to replace the occurrences of the same variables across this document.

Table 1: Supported Linux Distributions

Distribution	$distro	$arch	$arch_ext
x86_64			
Red Hat Enterprise Linux 9	rhel9	x86_64	x86_64
Red Hat Enterprise Linux 8	rhel8	x86_64	x86_64
OpenSUSE Leap 15 SP6	opensuse15	x86_64	x86_64
Rocky Linux 9	rhel9	x86_64	x86_64
Rocky Linux 8	rhel8	x86_64	x86_64
SUSE Linux Enterprise Server 15 SP6	sles15	x86_64	x86_64
Ubuntu 24.04 LTS	ubuntu2404	x86_64	amd64
Ubuntu 22.04 LTS	ubuntu2204	x86_64	amd64
Ubuntu 20.04 LTS	ubuntu2004	x86_64	amd64
Debian 12	debian12	x86_64	amd64
Fedora 41	fedora41	x86_64	x86_64
KylinOS V10 SP3 2403	kylin10	x86_64	x86_64
Azure Linux 2.0 (CBL Mariner 2.0)	cm2	x86_64	x86_64
Azure Linux 3.0	azl3	x86_64	x86_64
Amazon Linux 2023	amzn2023	x86_64	x86_64
Oracle Linux 9	rhel9	x86_64	x86_64

continues on next page

Table 1 – continued from previous page

Distribution	$distro	$arch	$arch_ext
Oracle Linux 8	rhel8	x86_64	x86_64
arm64-sbsa			
Red Hat Enterprise Linux 9	rhel9	sbsa	aarch64
Red Hat Enterprise Linux 8	rhel8	sbsa	aarch64
SUSE Linux Enterprise Server 15 SP6	sles15	sbsa	aarch64
Kylin V10 SP3 2403	kylin10	sbsa	aarch64
Ubuntu 24.04 LTS	ubuntu2404	sbsa	arm64
Ubuntu 22.04 LTS	ubuntu2204	sbsa	arm64
Ubuntu 20.04 LTS	ubuntu2004	sbsa	arm64
Azure Linux 3.0	azl3	sbsa	aarch64
Amazon Linux 2023	amzn2023	sbsa	aarch64

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.

Chapter 2. OS Support Policy

Support for the different operating systems will be until the standard EOSS/EOL date as defined for each operating system.

Please refer to the support lifecycle for these operating systems to know their support timelines and plan to move to newer releases accordingly.

Chapter 3. Administrative Privileges

This document is intended for readers familiar with the Linux environment.

- Commands which can be executed as a normal user will be prefixed by a $ at the beginning of the line
- Commands which require administrative privilege (`root`) will be prefixed by a # at the beginning of the line

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 or a `sudo` shell (`sudo -i`) for all the necessary commands.

Chapter 4. Pre-installation Actions

Some actions must be taken before the NVIDIA driver can be installed on Linux:

- Verify the system is running a supported version of Linux.
- Verify the system has the correct kernel headers and development packages installed.
- Handle conflicting installation methods.

4.1. Verify You Have a Supported Distribution of Linux

The NVIDIA driver packages are supported on some specific distributions of Linux. These are listed in the NIDIA Driver release notes.

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

```
$ hostnamectl
```

4.2. Verify the System has the Correct Kernel Packages Installed

The NVIDIA driver requires that the kernel headers and development packages for the running version of the kernel be installed at the time of the driver installation, as well whenever the driver is rebuilt. For example, if your system is running kernel version 3.17.4-301, the 3.17.4-301 kernel headers and development packages must also be installed.

The `rpm` and `deb` installations of the driver will make an attempt to install the kernel header and development packages if no version of these packages is currently installed. However, it will install the latest version of these packages, which may or may not match the version of the kernel your system is using.

Therefore, **it is best to manually ensure the correct version of the kernel headers and development packages are installed prior to installing the NVIDIA driver**, as well as whenever you change the kernel version.

The version of the kernel your system is running can be found by running the following command:

```
$ uname -r
```

This is the version of the kernel headers and development packages that must be installed prior to installing the NVIDIA drivers. This command will be used multiple times below to specify the version of the packages to install. Note that below are the common-case scenarios for kernel usage. More advanced cases, such as custom kernel branches, should ensure that their kernel headers and sources match the kernel build they are running.

Note: If you perform a system update which changes the version of the Linux kernel being used, the packages should automatically rebuild the kernel modules unless precompiled modules are being used.

Chapter 5. Choose an Installation Method

The NVIDIA driver can be installed using distribution-specific packages (`rpm` and Debian packages).

The distribution-independent package has the advantage of working across a wider set of Linux distributions, but does not update with 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.

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`, `dnf`, `tdnf`, `yum`, or `zypper`.

Distribution-specific instructions detail how to install NVIDIA driver:

- *Red Hat Enterprise Linux*
- *KylinOS*
- *Fedora*
- *SUSE*
- *Ubuntu*
- *Debian*
- *Amazon Linux*
- *Azure Linux*

Finally, some helpful package manager capabilities are detailed.

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.

Chapter 6. Recent Updates

6.1. Compute-only HPC Node

Starting in CUDA 12.8, the `cuda` meta-package now installs the *Compute only* part driver. By default, if nothing is specified on the command line, the installation will prefer the Open GPU kernel modules.

This is achieved by having the final dependency of `cuda-runtime-X-Y` on `nvidia-driver-cuda/nvidia-compute-G06/libnvidia-compute`, depending on the distribution.

6.2. Driver Helper Script

A new script is available to detect and install the best NVIDIA driver packages for the user's system. This piece of software is meant to help users decide on which NVIDIA graphics driver to install, based on the detected system's hardware.

To install the driver helper script, install the `nvidia-driver-assistant` package using `apt/dnf/tdnf/zypper`.

The following table explains the different flags for the driver helper script:

Table 2: Driver helper script flags

Flags Used	Description
`--install`	Install the recommended driver.
`--branch [BRANCH]`	Specify an NVIDIA Driver branch.
`--list-supported-distros`	Print out the list of the supported Linux distributions.
`--supported-gpus [SUPPORTED_GPUS]`	Use a different `supported-gpus.json` file.
`--sys-path [SYS_PATH]`	Use a different `/sys` path. Useful for testing.
`--os-release-path [OS_RELEASE_PATH]`	Use a different path for the `os-release` file. Useful for testing.
`--distro [DISTRO]`	Specify a Linux distro using the `DISTRO:VERSION` or `DISTRO` patter
`--module-flavor [MODULE_FLAVOR]`	Specify a kernel module flavor; `open` and `closed` are accepted value
`--verbose`	[OPTIONAL] Verbose output.

The following are example command outputs:

Table 3: Command outputs

Command	Example Output
`$ nvidia-driver-assistant`	**Detected GPUs:** NVIDIA GeForce RTX 3070 - (pci_id 0x2484) **Detected system:** Ubuntu 24.04 **Please copy and paste the following command to install the open kernel module flavour:** sudo apt install -y nvidia-open
`$ nvidia-driver-assistant --install`	**Detected GPUs:** NVIDIA GeForce RTX 3070 - (pci_id 0x2484) **Detected system:** Ubuntu 24.04 Using the NVIDIA driver implies acceptance of the NVIDIA Software License Agreement, contained in the "LICENSE" file in the "/usr/share/nvidia-driver-assistant/driver_eula" directory **Installing the following package for the open kernel module flavour:** sudo apt install -y nvidia-open
`$ nvidia-driver-assistant --install --module-flavor closed`	**Detected GPUs:** NVIDIA GeForce RTX 3070 - (pci_id 0x2484) **Detected system:** Ubuntu 24.04 Using the NVIDIA driver implies acceptance of the NVIDIA Software License Agreement, contained in the "LICENSE" file in the "/usr/share/nvidia-driver-assistant/driver_eula" directory Installing the following package for the legacy kernel module flavour: sudo apt install -y cuda-drivers

6.2.1. Auto Hardware Selection

Starting in 560, the standalone NVIDIA driver runfile will use hardware detection to auto-select between installation of Open GPU or proprietary kernel modules based on the detected SKUs.

The CUDA runfile bundles an intact NVIDIA driver runfile and passes the `--silent` flag if the driver is selected (default) via ncurses or the CLI.

Therefore, by default the user will install the auto-detected flavor of the kernel modules using the CUDA runfile. Additionally, we expose overrides to select the open GPU or proprietary kernel modules in the `ncurses` UI advanced options, or via `--kernel-module-type=open` and `--kernel-module-type=proprietary` flags.

Chapter 7. Kernel Modules

The NVIDIA Linux GPU Driver contains several kernel modules:

- `nvidia.ko`
- `nvidia-modeset.ko`
- `nvidia-uvm.ko`
- `nvidia-drm.ko`
- `nvidia-peermem.ko`

Starting in the 515 driver release series, two “flavors” of these kernel modules are provided:

- **Proprietary** - This is the flavor that NVIDIA has historically shipped. For older GPUs from the Maxwell, Pascal, or Volta architectures. The open-source GPU kernel modules are not compatible with your platform, so the proprietary modules is what you are required to use.
- **Open-source** - Published kernel modules that are dual licensed MIT/GPLv2. With every driver release, the source code to the open kernel modules will be published at https://github.com/NVIDIA/open-gpu-kernel-modules and a tarball will be provided at https://download.nvidia.com/XFree86/NVIDIA-kernel-module-source/. These are only for Turing and newer architectures, and this is what you should use if you have one of those architectures.

Starting in the 560 driver release series, the open kernel module flavor is the default installation.

Open GPU kernel modules are supported only on Turing and newer generations. To verify that your NVIDIA GPU is at least Turing or newer:

```
$ lspci | grep VGA
```

7.1. Open GPU Kernel Modules Installation

For simplification, we’ve condensed the package manager recommendations in table format. All releases beyond driver version 570 will use these packaging conventions.

The following commands will install a *full* driver, which will include all desktop components as well as CUDA libraries and tools for computational workloads.

Table 4: Package manager installation recommendations

Distribution	Install the Latest	Install a Specific Release
Red Hat Enterprise Linux 8/9, Rocky Linux 8/9 Oracle Linux 8/9, KylinOS 10, Amazon Linux 2023	`# dnf module enable nvidia-driver:open-dkms` `# dnf install nvidia-open`	`# dnf module enable nvidia-driver:575-open` `# dnf install nvidia-open-575`
Fedora 41	`# dnf install nvidia-open`	`# dnf install nvidia-open-575`
Azure Linux 2 / Azure Linux 3	`# tdnf install nvidia-open`	`# tdnf install nvidia-open-575`
openSUSE Leap 15	`# zypper install nvidia-open`	`# zypper install nvidia-open-575`
SUSE Linux Enterprise Server 15 (x86_64)	`# zypper install nvidia-open`	`# zypper install nvidia-open-575`
SUSE Linux Enterprise Server 15 (aarch64)	`# zypper install nvidia-open`	`# zypper install nvidia-open-575`
Debian 12	`# apt install nvidia-open`	`# apt install nvidia-open-575`
Ubuntu 20.04/22.04/24.04	`# apt install nvidia-open`	`# apt install nvidia-open-575`

7.2. Proprietary GPU Kernel Modules Installation

For simplification, we've condensed the package manager recommendations in table format. All releases beyond driver version 570 will use these packaging conventions.

The following commands will install a *full* driver, which will include all desktop components as well as CUDA libraries and tools for computational workloads.

Table 5: Proprietary GPU Kernel Module Installation

Distribution	Install the Latest	Install a Specific Release
Red Hat Enterprise Linux 8/9, Rocky Linux 8/9 Oracle Linux 8/9, KylinOS 10, Amazon Linux 2023	`# dnf module enable nvidia-driver:latest-dkms` `# dnf install cuda-drivers`	`# dnf module enable nvidia-driver:575-dkms` `# dnf install cuda-drivers-575`
Fedora 41	`# dnf install cuda-drivers`	`# dnf install cuda-drivers-575`
Azure Linux 2 / Azure Linux 3	Only the open kernel modules are supported.	Only the open kernel modules are supported.
openSUSE Leap 15	`# zypper install cuda-drivers`	`# zypper install cuda-drivers-575`
SUSE Linux Enterprise Server 15	`# zypper install cuda-drivers`	`# zypper install cuda-drivers-575`
Debian 12	`# apt install cuda-drivers`	`# apt install cuda-drivers-575`
Ubuntu 20.04/22.04/24.04	`# apt install cuda-drivers`	`# apt install cuda-drivers-575`

Chapter 8. Red Hat Enterprise Linux

This section covers:

- Red Hat Enterprise Linux 8
- Red Hat Enterprise Linux 9
- Rocky Linux 8
- Rocky Linux 9
- Rocky Linux 10
- Oracle Linux 8
- Oracle Linux 9
- Oracle Linux 10

8.1. Preparation

1. Perform the *Pre-installation Actions*.
2. Precompiled streams **do not** require kernel headers or DKMS. If precompiled kernel modules are not required, the kernel headers and development packages for the currently running kernel can be installed with:
   - **Red Hat Enterprise Linux 9, Rocky Linux 9, Oracle Linux 9:**

     ```
     # dnf install kernel-devel-matched kernel-headers
     ```

   - **Red Hat Enterprise Linux 8, Rocky Linux 8, Oracle Linux 8:**

     ```
     # dnf install kernel-devel-$(uname -r) kernel-headers
     ```

   Note: At the moment of writing, **the Oracle UEK kernel is not supported on Oracle Linux**. The package containing the required kernel headers (should be called `kernel-uek-headers`) is missing, so the NVIDIA modules can not be built for the Oracle UEK kernel variant.

3. Satisfy third-party package dependencies:

   The NVIDIA driver `rpm` packages depend on other external packages. Those packages are only available on third-party repositories, such as EPEL. Any such third-party repositories must be

added to the package manager repository database before installing the NVIDIA driver `rpm` packages, or missing dependencies will prevent the installation from proceeding.

- **Red Hat Enterprise Linux 9:**

```
# subscription-manager repos --enable=rhel-9-for-$arch-appstream-rpms
# subscription-manager repos --enable=rhel-9-for-$arch-baseos-rpms
# subscription-manager repos --enable=codeready-builder-for-rhel-9-$arch-rpms
# dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.
↪noarch.rpm
```

- **Red Hat Enterprise Linux 8:**

```
# subscription-manager repos --enable=rhel-8-for-$arch-appstream-rpms
# subscription-manager repos --enable=rhel-8-for-$arch-baseos-rpms
# subscription-manager repos --enable=codeready-builder-for-rhel-8-$arch-rpms
# dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.
↪noarch.rpm
```

- **Rocky Linux 9:**

```
# dnf config-manager --set-enabled crb
# dnf install epel-release
```

- **Rocky Linux 8:**

```
# dnf config-manager --set-enabled powertools
# dnf install epel-release
```

- **Oracle Linux 9:**

```
# dnf config-manager --set-enabled ol9_codeready_builder
# dnf install oracle-epel-release-el9
```

- **Oracle Linux 8:**

```
# dnf config-manager --set-enabled ol8_codeready_builder
# dnf install oracle-epel-release-el8
```

4. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

8.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

8.3. Network Repository Installation

1. Enable the network repository:

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

2. Clean DNF repository cache:

```
# dnf clean expire-cache
```

8.4. DNF module enablement

This is required for distributions where content is not distributed as a flat repository but as a repository containing DNF module streams.

These instructions apply to both local and network installations and only for the following distributions:q

- Red Hat Enterprise Linux 8
- Red Hat Enterprise Linux 9
- Rocky Linux 8
- Rocky Linux 9
- Oracle Linux 8
- Oracle Linux 9

Open Kernel Modules

```
# dnf module enable nvidia-driver:open-dkms
```

- Example DKMS streams: `575-open` or `open-dkms`

Proprietary Kernel Modules

```
# dnf module enable nvidia-driver:latest-dkms
```

- Example DKMS streams: `575-dkms` or `latest-dkms`
- Example precompiled streams: `575` or `latest`

8.5. Driver Installation

These instructions apply to both local and network installations.

Open Kernel Modules

```
# dnf install nvidia-open
```

Proprietary Kernel Modules

```
# dnf install cuda-drivers
```

8.6. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

8.6.1. Compute-only System

Open Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-latest-dkms
```

8.6.2. Desktop-only System

Open Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-latest-dkms
```

8.7. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 9. KylinOS

This section covers:

- KylinOS 10

9.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# dnf install kernel-devel-$(uname -r) kernel-headers
```

3. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

9.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

9.3. Network Repository Installation

1. Enable the network repository:

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

2. Clean DNF repository cache:

```
# dnf clean expire-cache
```

9.4. Driver Installation

These instructions apply to both local and network installations.

Module Streams

```
# dnf module install nvidia-driver:<stream>/<profile>
```

Open Kernel Modules

```
# dnf module enable nvidia-driver:open-dkms
# dnf install nvidia-open
```

- Example DKMS streams: `575-open` or `open-dkms`

Proprietary Kernel Modules

```
# dnf module enable nvidia-driver:latest-dkms
# dnf install cuda-drivers
```

where profile by default is `default` and does not need to be specified.

- Example DKMS streams: `575-dkms` or `latest-dkms`

9.5. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

9.5.1. Compute-only System

Open Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-latest-dkms
```

9.5.2. Desktop-only System

Open Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-latest-dkms
```

9.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 10. Fedora

This section covers:

- Fedora 41

10.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# dnf install kernel-devel-matched kernel-headers
```

3. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

10.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

10.3. Network Repository Installation

1. Enable the network repository:

```
# dnf config-manager addrepo --from-repofile=https://developer.download.nvidia.
↪com/compute/cuda/repos/$distro/$arch/cuda-$distro.repo
```

2. Clean DNF repository cache:

```
# dnf clean expire-cache
```

10.4. Driver Installation

These instructions apply to both local and network installations.

Branch stickiness

Please refer to the *DNF 5* paragraph of the *Version locking* section.

Open Kernel Modules

```
# dnf install nvidia-open
```

Proprietary Kernel Modules

```
# dnf install cuda-drivers
```

10.5. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

10.5.1. Compute-only System

Open Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver-cuda kmod-nvidia-latest-dkms
```

10.5.2. Desktop-only System

Open Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-open-dkms
```

Proprietary Kernel Modules

```
# dnf install nvidia-driver kmod-nvidia-latest-dkms
```

10.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 11. SUSE

This section covers:

- SUSE Linux Enterprise 15
- OpenSUSE Leap 15

11.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel development packages for the currently running kernel can be installed with:

   ```
   # zypper install -y kernel-<variant>-devel=<version>
   ```

   Note: `<variant>` may be: `default`, `64k` for `aarch64` or `default`, `azure` for `x86_64`.

   To run the above command, you will need the variant and version of the currently running kernel. Use the output of the `uname` command to determine the currently running kernel's variant and version:

   ```
   $ uname -r
   3.16.6-2-default
   ```

   In the above example, the variant is `default` and version is `3.16.6-2`.

   The kernel development packages for the default kernel variant can be installed with:

   ```
   # zypper install -y kernel-default-devel=$(uname -r | sed 's/\-default//')
   ```

3. The kernel headers and development packages for the currently running kernel can be installed with:

   ```
   # zypper install -y kernel-<variant>-devel=$(uname -r | sed 's/\-default//')
   ```

4. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

11.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

11.3. Network Repository Installation

1. Enable the network repository:

```
# zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/$distro/
↪$arch/cuda-$distro.repo
```

2. Refresh Zypper repository cache:

```
# SUSEConnect --product PackageHub/15/<architecture>
# zypper refresh
```

11.4. Driver Installation

These instructions apply to both local and network installations.

Open Kernel Modules

```
# zypper -v install nvidia-open
```

Proprietary Kernel Modules

```
# zypper -v install cuda-drivers
```

11.5. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

11.5.1. Compute-only System

Open Kernel Modules

```
# zypper -v install nvidia-compute-G06 nvidia-open-driver-G06
```

Proprietary Kernel Modules

```
# zypper -v install nvidia-compute-G06 nvidia-driver-G06
```

11.5.2. Desktop-only System

Open Kernel Modules

```
# zypper -v install nvidia-video-G06 nvidia-open-driver-G06
```

Proprietary Kernel Modules

```
# zypper -v install nvidia-video-G06 nvidia-driver-G06
```

11.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 12. Ubuntu

This section covers:

- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

12.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# apt install linux-headers-$(uname -r)
```

3. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

12.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro-$version_$arch_ext.deb
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# dpkg -i nvidia-driver-local-repo-$distro-$version_$arch_ext.deb
# apt update
```

3. Enroll ephemeral public GPG key:

```
# cp /var/nvidia-driver-local-repo-$distro-$version/nvidia-driver-*-keyring.gpg /
↪usr/share/keyrings/
```

12.3. Network Repository Installation

Install the new cuda-keyring package:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/$distro/$arch/cuda-
↪keyring_1.1-1_all.deb
# dpkg -i cuda-keyring_1.1-1_all.deb
# apt update
```

If you are unable to install the `cuda-keyring` package you can follow these instructions:

1. Enroll the new signing key:

   ```
   $ wget https://developer.download.nvidia.com/compute/cuda/repos/$distro/$arch/
   ↪cuda-archive-keyring.gpg
   # mv cuda-archive-keyring.gpg /usr/share/keyrings/cuda-archive-keyring.gpg
   ```

2. Enable the network repository:

   ```
   # echo "deb [signed-by=/usr/share/keyrings/cuda-archive-keyring.gpg] https://
   ↪developer.download.nvidia.com/compute/cuda/repos/$distro/$arch/ /" \
       | tee /etc/apt/sources.list.d/cuda-$distro-$arch.list
   ```

12.4. Driver Installation

These instructions apply to both local and network installations.

Open Kernel Modules

```
# apt install nvidia-open
```

Proprietary Kernel Modules

```
# apt install cuda-drivers
```

12.5. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

12.5.1. Compute-only System

Open Kernel Modules

```
# apt -V install libnvidia-compute-575 nvidia-dkms-575-open
```

Proprietary Kernel Modules

```
# apt -V install libnvidia-compute-575 nvidia-dkms-575
```

12.5.2. Desktop-only System

Open Kernel Modules

```
# apt -V install libnvidia-gl-575 nvidia-dkms-575-open
```

Proprietary Kernel Modules

```
# apt -V install libnvidia-gl-575 nvidia-dkms-575
```

12.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 13. Debian

This section covers:

- Debian 12

13.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# apt install linux-headers-$(uname -r)
```

3. Enable the contrib repository:

```
# add-apt-repository contrib
```

4. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

13.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro-$version_$arch_ext.deb
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# dpkg -i nvidia-driver-local-repo-$distro-$version*_$arch_ext.deb
# apt update
```

3. Enroll ephemeral public GPG key:

```
# cp /var/nvidia-driver-local-repo-$distro-$version/nvidia-driver-*-keyring.gpg /
↪usr/share/keyrings/
```

13.3. Network Repository Installation

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
# dpkg -i cuda-keyring_1.1-1_all.deb
# apt update
```

If you are unable to install the `cuda-keyring` package you can follow these instructions:

1. Enroll the new signing key:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/$distro/$arch/
↪cuda-archive-keyring.gpg
# mv cuda-archive-keyring.gpg /usr/share/keyrings/cuda-archive-keyring.gpg
```

2. Enable the network repository:

```
# echo "deb [signed-by=/usr/share/keyrings/cuda-archive-keyring.gpg] https://
↪developer.download.nvidia.com/compute/cuda/repos/$distro/$arch/ /" \
    | tee /etc/apt/sources.list.d/cuda-$distro-$arch.list
```

13.4. Driver Installation

These instructions apply to both local and network installations.

Open Kernel Modules

```
# apt -V install nvidia-open
```

Proprietary Kernel Modules

```
# apt -V install cuda-drivers
```

13.5. Compute-only (Headless) and Desktop-only (no Compute) Installation

It's possible to install the driver without all the desktop components (GL, EGL, Vulkan, X drivers, and so on) to limit the footprint and dependencies on the system. With the same logic it's possible to install a desktop system without any compute component.

Note: The components excluded at installation time can always be added at a later stage. This will pull in all the additional dependencies required.

13.5.1. Compute-only System

Open Kernel Modules

```
# apt -V install nvidia-driver-cuda nvidia-kernel-open-dkms
```

Proprietary Kernel Modules

```
# apt -V install nvidia-driver-cuda nvidia-kernel-dkms
```

13.5.2. Desktop-only System

Open Kernel Modules

```
# apt -V install nvidia-driver nvidia-kernel-open-dkms
```

Proprietary Kernel Modules

```
# apt -V install nvidia-driver nvidia-kernel-dkms
```

13.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 14. Amazon Linux

This section covers:

- Amazon Linux 2023

14.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# dnf install kernel-devel-$(uname -r) kernel-headers-$(uname -r)
```

3. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

14.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/local_
↪installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

   where `$version` is the NVIDIA driver version

2. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

14.3. Network Repository Installation

1. Enable the network repository:

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

2. Clean DNF repository cache:

```
# dnf clean expire-cache
```

14.4. Driver Installation

These instructions apply to both local and network installations.

Module Streams

```
# dnf module install nvidia-driver:<stream>/<profile>
```

Open Kernel Modules

```
# dnf module enable nvidia-driver:open-dkms
# dnf install nvidia-open
```

where profile by default is `default` and does not need to be specified.

- Example dkms streams: `575-open` or `open-dkms`

Proprietary Kernel Modules

```
# dnf module enable nvidia-driver:latest-dkms
# dnf install cuda-drivers
```

where profile by default is `default` and does not need to be specified.

- Example dkms streams: `575-dkms` or `latest-dkms`

14.5. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

Chapter 15. Azure Linux

This section covers:

- Azure Linux 2 (CBL Mariner 2.0)
- Azure Linux 3

15.1. Preparation

1. Perform the *Pre-installation Actions*.
2. The kernel headers and development packages for the currently running kernel can be installed with:

```
# tdnf install kernel-devel-$(uname -r) kernel-headers-$(uname -r) kernel-modules-
↪extra-$(uname -r)
```

3. Enable the extended repository:

   Azure Linux 2:

```
# tdnf install mariner-repos-extended
```

   Azure Linux 3:

```
# tdnf install azurelinux-repos-extended
```

4. Choose an installation method: *Local Repository Installation* or *Network Repository Installation*.

15.2. Local Repository Installation

1. Download the NVIDIA driver:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/$version/
↪local_installers/nvidia-driver-local-repo-$distro.$version.$arch_ext.rpm
```

   where `$version` is the NVIDIA driver version

1. Install local repository on file system:

```
# rpm --install nvidia-driver-local-repo-$distro.$version*.$arch_ext.rpm
```

15.3. Network Repository Installation

1. Enable the network repository:

```
# tdnf config-manager --add-repo https://developer.download.nvidia.com/compute/
↪cuda/repos/$distro/$arch/cuda-$distro.repo
```

2. Clean DNF repository cache:

```
# tdnf clean expire-cache
```

15.4. Driver Installation

These instructions apply to both local and network installations.

```
# tdnf install nvidia-open
```

15.5. Reboot the System

```
# reboot
```

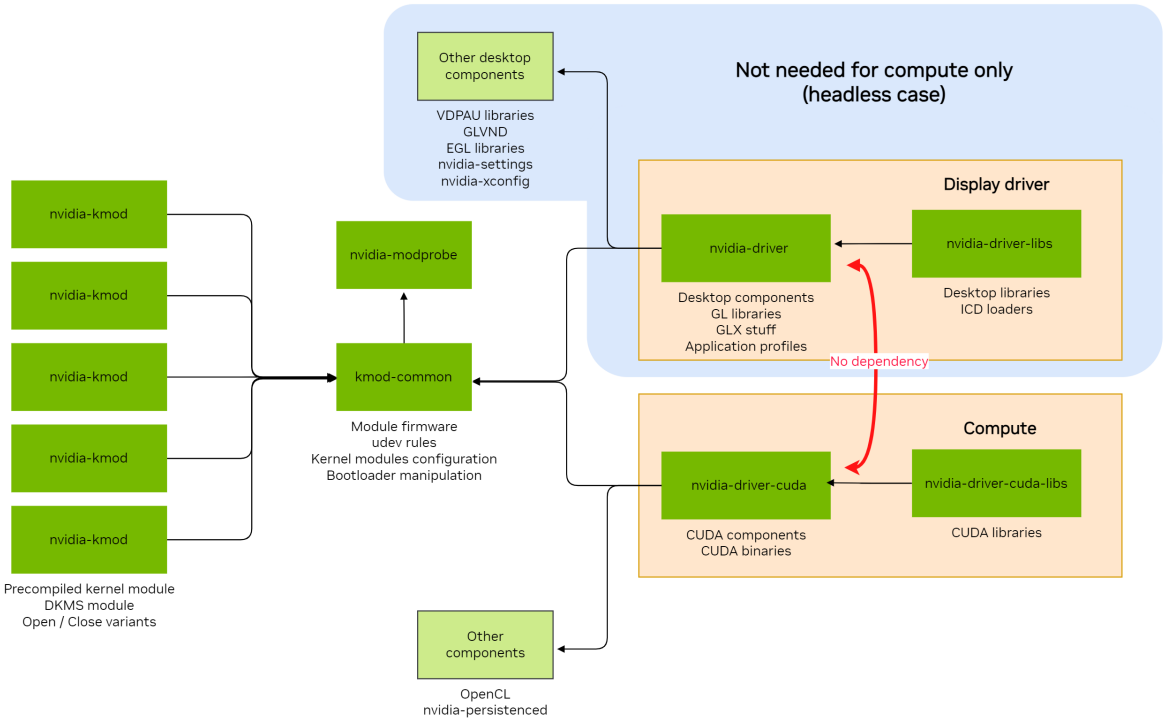
Perform the *Post-installation Actions*.

Chapter 16. Compute-only and Desktop Installation

Starting in 560, the driver allows a new custom installation method which includes only part of the driver for different use cases. This allows for a more granular installation with fewer dependencies, especially for compute only systems where the desktop components would pull in a lot of extra libraries that then would go unused.

Depending on the operating system, it is now possible to install the driver in the following configuration:

- Desktop: Contains all the X/Wayland drivers and libraries to allow running a GPU with power management enabled on a desktop system (laptop, workstation, and so on) but does not include any CUDA component.
- Compute-only, or “headless”: Contains everything required to run CUDA applications on a GPU system where the GPU is not used to drive a display: a computational cluster, a workstation with a dedicated NVIDIA GPU, and so on.
- Desktop and Compute: The canonical way of installing the driver, with every possible library and display component. This might be required in cross functional combinations, for CUDA accelerated video encoding/decoding.



This option is now supported for the following operating systems, with more to follow in future releases:

- Red Hat Enterprise Linux 8 / Rocky Linux 8 / Oracle Linux 8
- Red Hat Enterprise Linux 9 / Rocky Linux 9 / Oracle Linux 9
- Kylin 10
- Fedora 41
- OpenSUSE Leap 15
- SUSE Linux Enterprise Server 15
- Debian 12
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

The installation of the driver for the following operating systems is provided exclusively in compute only/headless mode:

- Azure Linux 2 (CBL Mariner 2.0)
- Azure Linux 3
- Amazon Linux 2023
  - Upgrading the driver on Amazon Linux 2023 to version 560 or newer will remove all the unused desktop components as part of the upgrade.

More information is available in the respective sections.

Chapter 17. Wayland-only Desktop Installation

This option is now supported for the following operating systems, with more to follow in future releases:

- Red Hat Enterprise Linux 8 / Rocky Linux 8 / Oracle Linux 8
- Red Hat Enterprise Linux 9 / Rocky Linux 9 / Oracle Linux 9
- Fedora 41

To drop support for X.org server, its desktop sessions and all relevant user space X driver packages, just remove the X.org server package. There is a reverse dependency on the X.org package for having X components installed with the driver, so by removing it, you're removing everything that is not required to run an X.org session.

```
# dnf remove xorg-x11-server-Xorg
```

Chapter 18. GNOME Software Integration

The Gnome Software installation is geared towards non technical users and installs only the desktop part of the driver, leaving the compute components out. The process guides the user on installing the driver and also enrolling the Secure Boot keys.

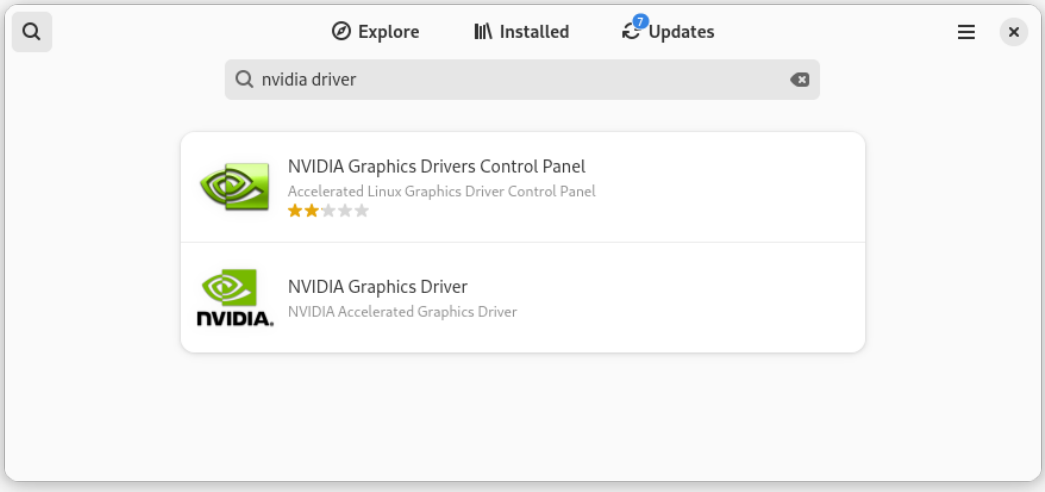
At the moment this process is enabled for the following distributions:

- Fedora 41

This requires to **have the NVIDIA repository already configured/added** to have the information appear in GNOME Software as part of the PackageKit downloads.

18.1. Driver Installation

1. After adding the NVIDIA repository, search for “nvidia driver”:

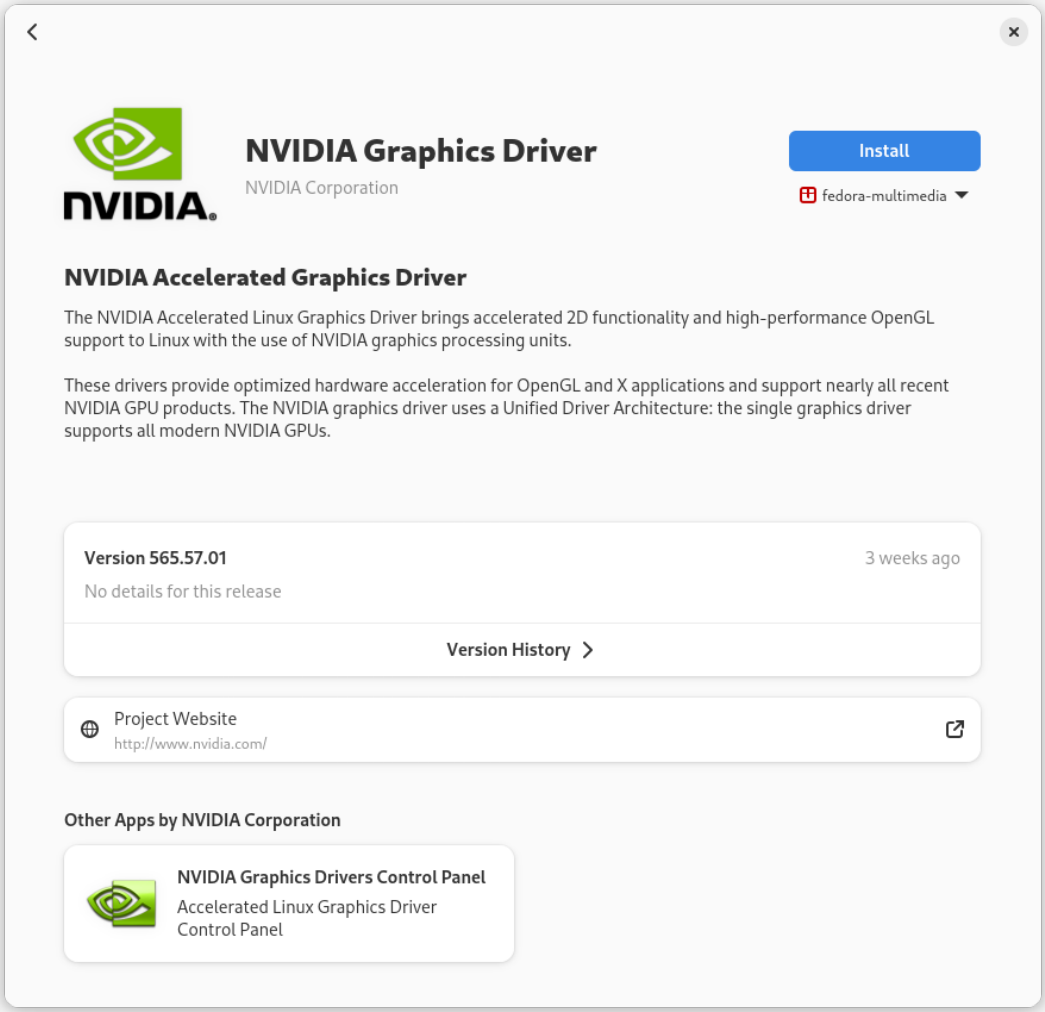


- If nothing is found, then run the following command to force a PackageKit metadata refresh out of its normal regular schedule:

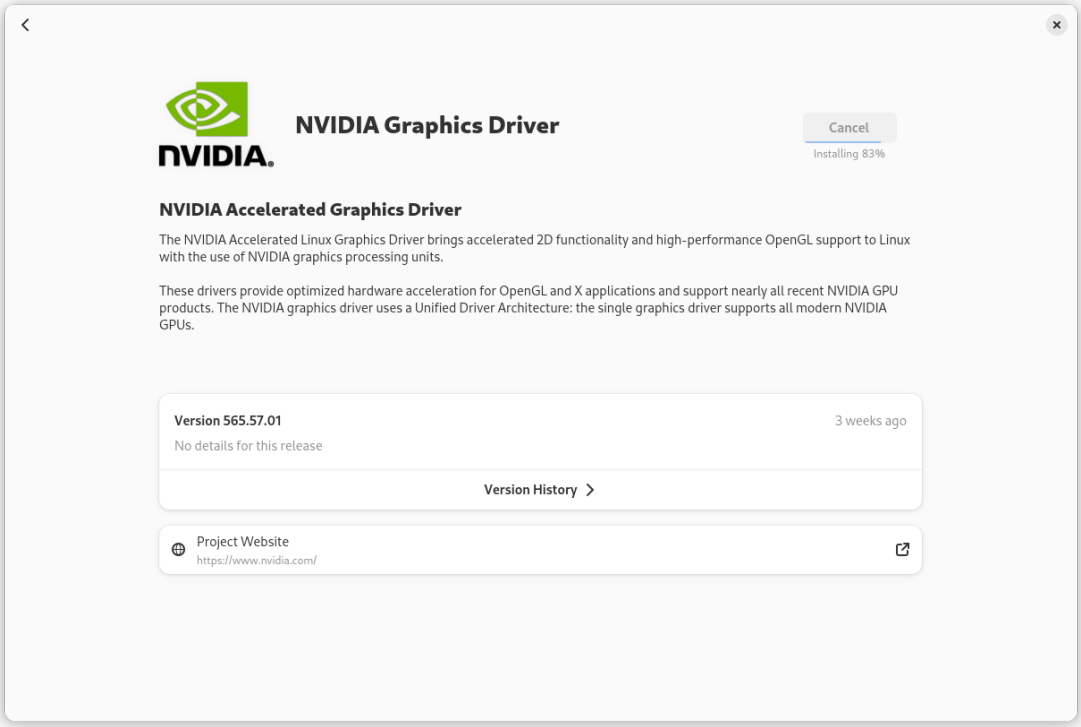
```
# pkcon refresh force
```

2. Click on the *NVIDIA Graphics Driver* item.
3. Click on the *Install* button and insert the Administrator password (`root`).

Note: Multiple repositories can provide the same metadata for the same package set. The driver is also shipped by RPMFusion and other repositories; so you must select the **CUDA repository** from the top right drop down menu under the *Install* button.

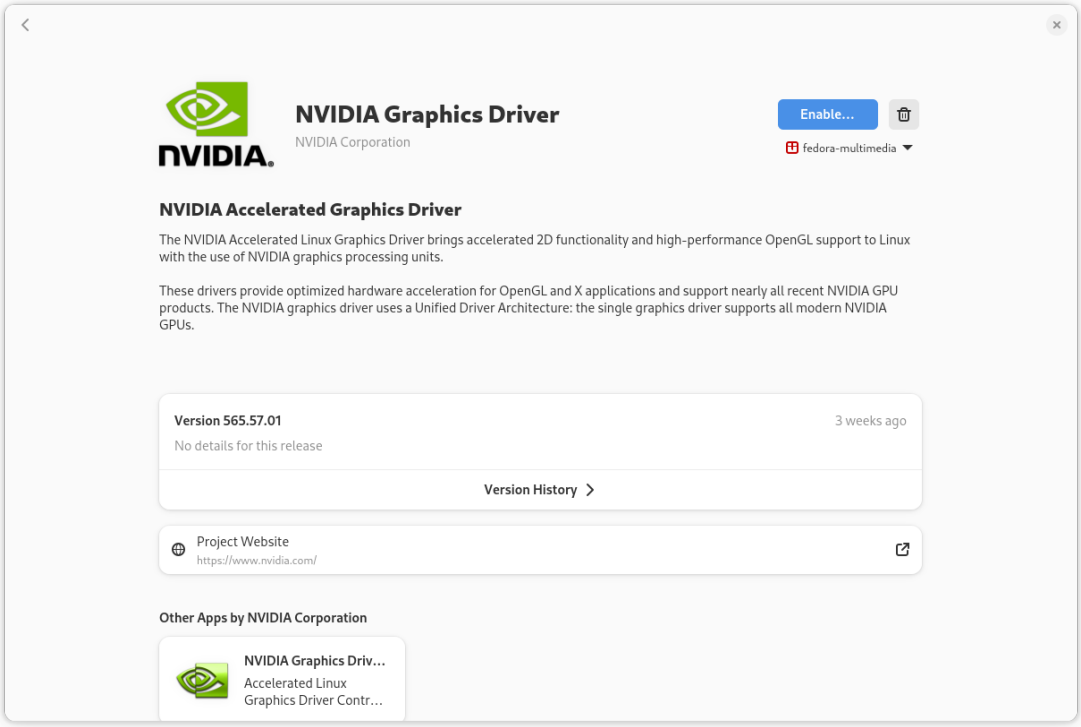


4. The installation will now begin. You can check the progress below the greyed out button.

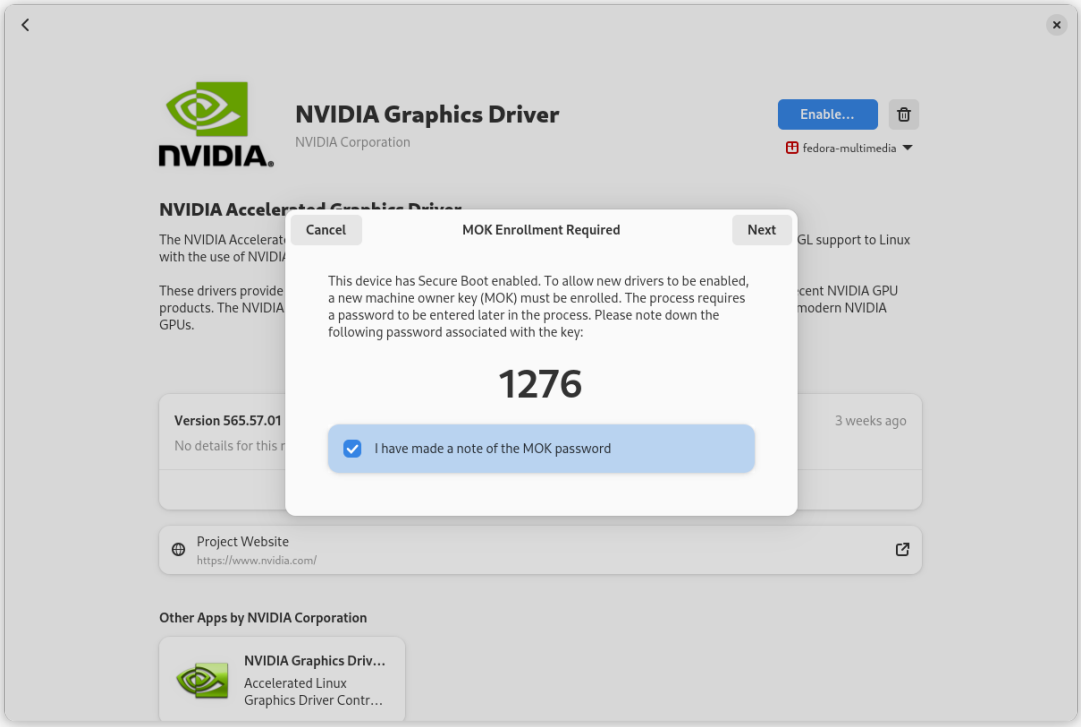


18.2. Secure Boot Preparation

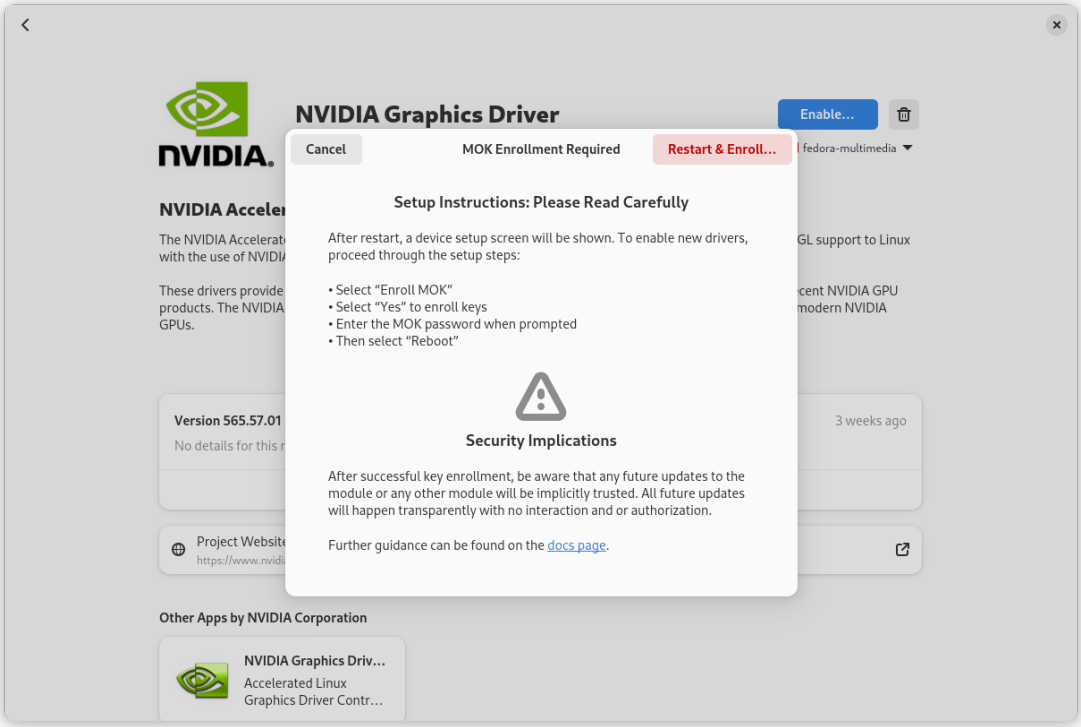
1. Once the driver is installed, the button will switch to *Enable…*. Click on the button.



2. The next screen you will be presented with the MOK enrollment prompt.
   - Note down the number. This will be the MOK password used later.
   - Check the *I have made a note of the MOK password* box.
   - Click *Next*.



3. Click the *Restart & Enroll MOK* button.
   - You will be asked again for the Administrator password (`root`) and the system will restart.



Then, you can proceed to enroll the locally generated MOK. Refer to the *Machine Owner Key Enrollment* section to proceed.

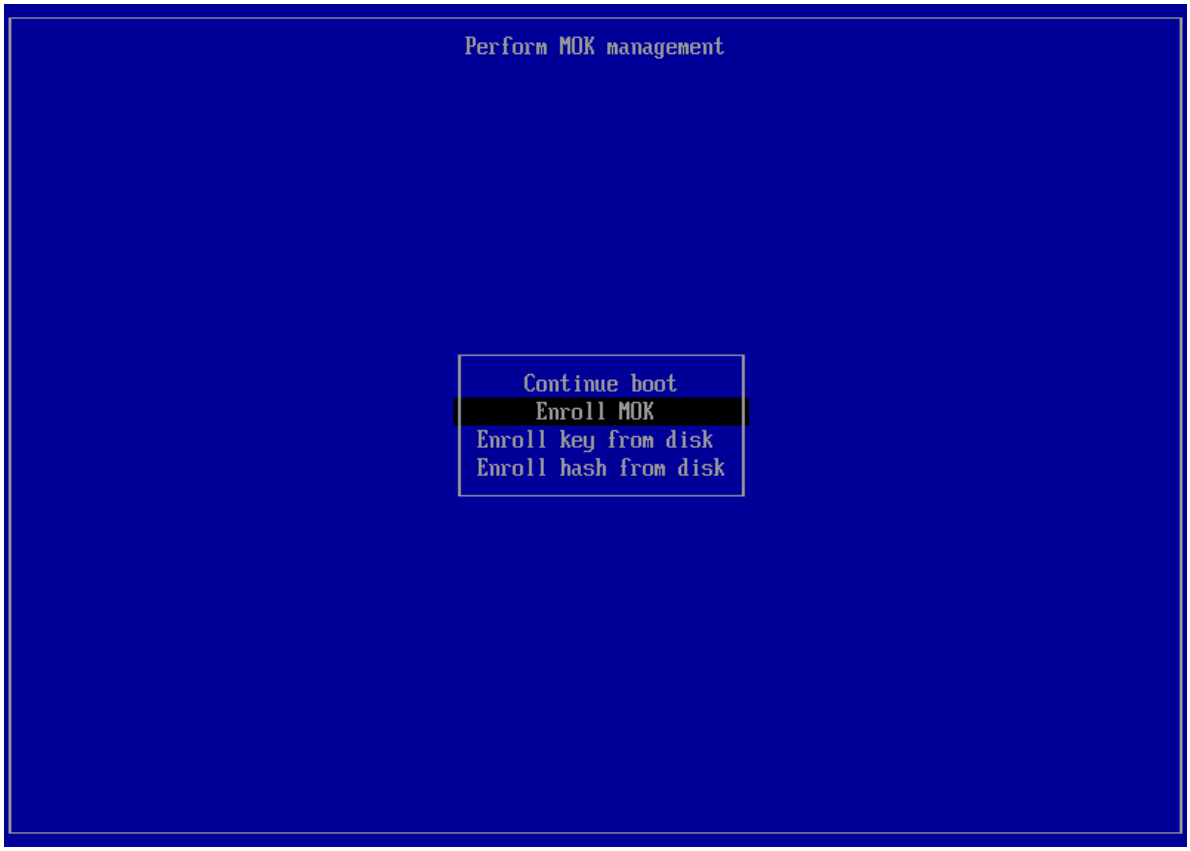
18.3. Machine Owner Key Enrollment

Note: This section is a copy of what is published on Fedora's Machine Owner Key enrollment documentation page.

In order to successfully reboot after the NVIDIA driver installation, you have to enroll the Machine Owner Key you created during installation in GNOME Software. During rebooting you'll be presented with the `mokutil` tool:

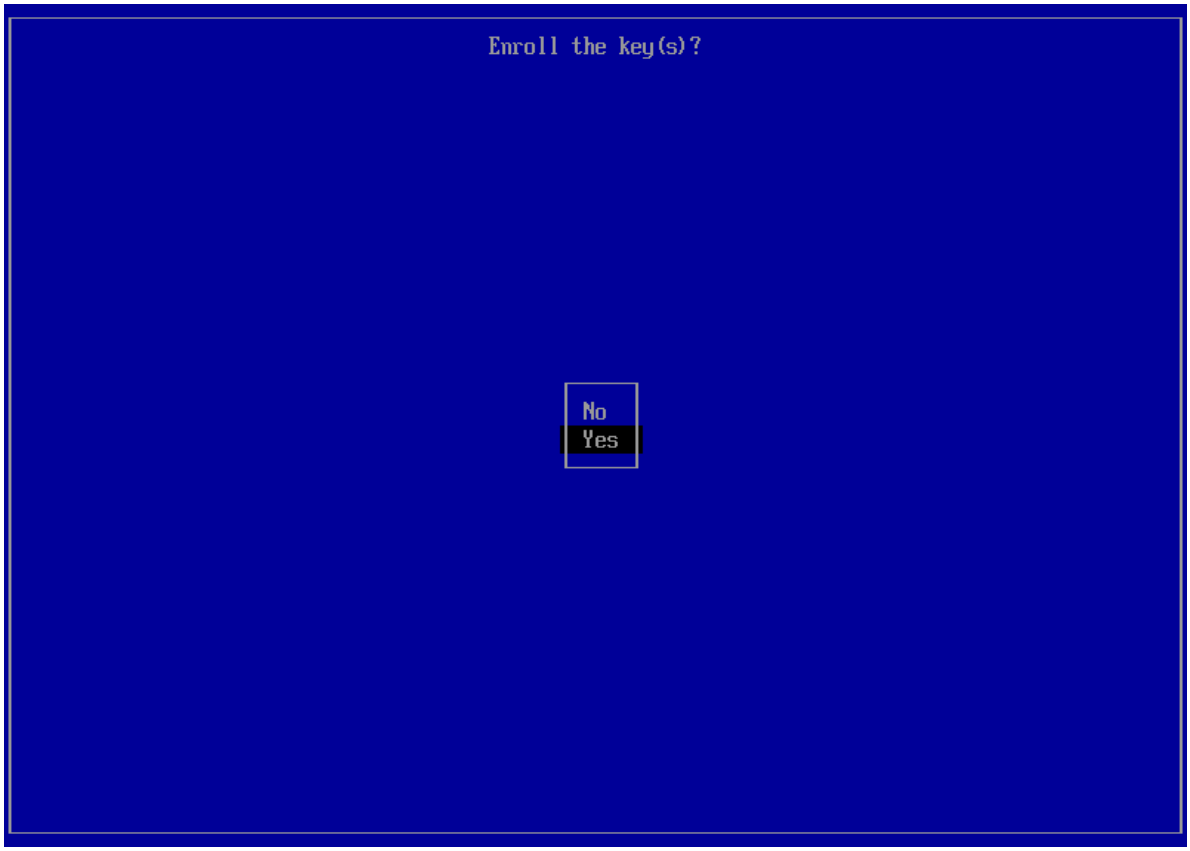
1. Press any key to begin.

2. Select *Enroll MOK*:



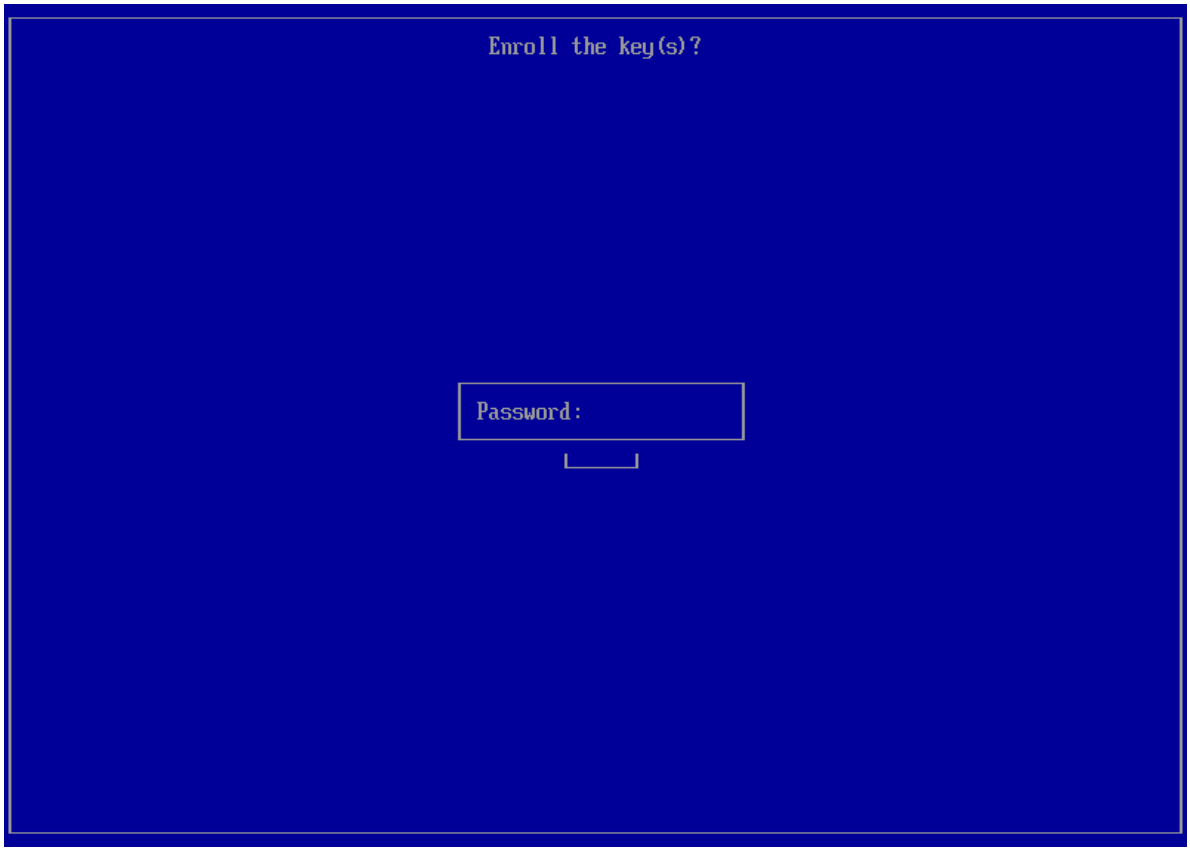
3. Select *Continue* to proceed:

4. Select *Yes* to enroll the key.

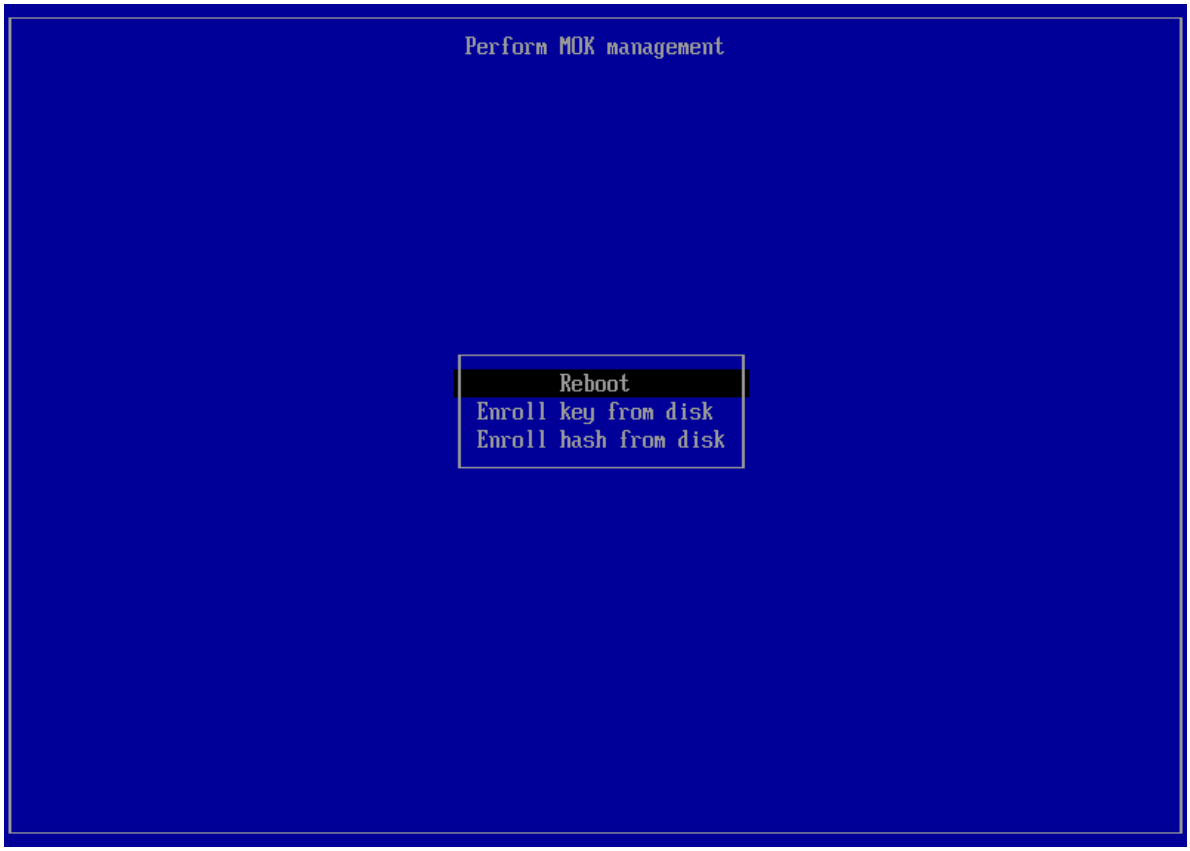


5. Type the MOK password you created for the key during installation.

Note: Please note that there will be no feedback on the screen as you type the characters.

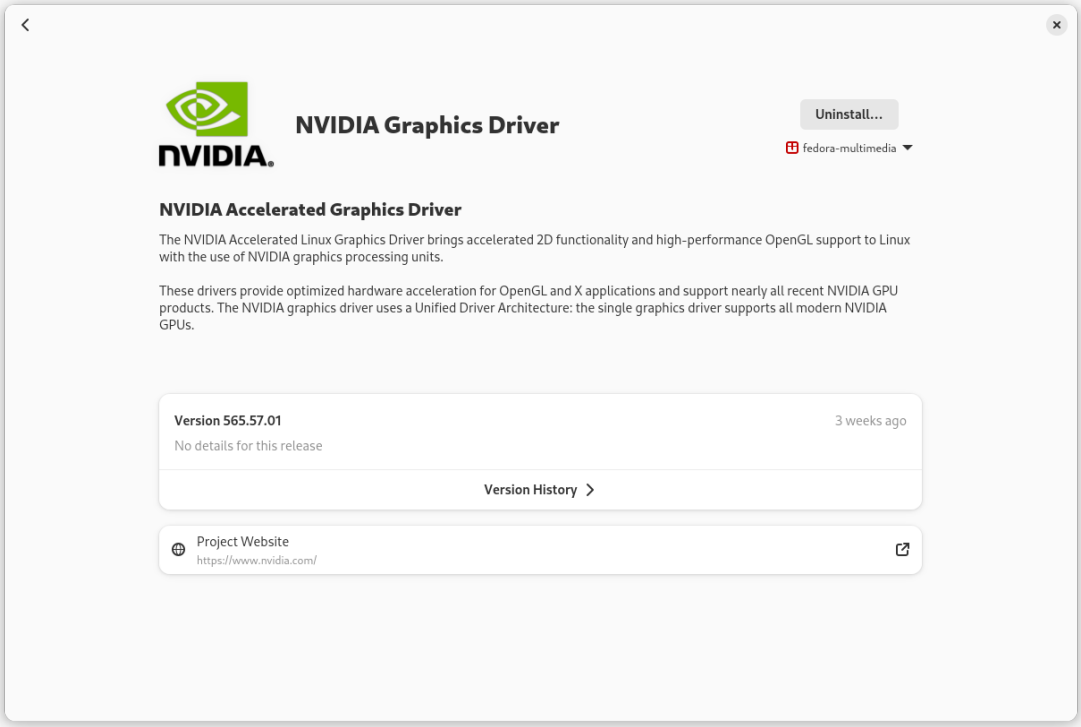


6. Select *Reboot* to reboot into the operating system with the NVIDIA drivers and Secure Boot enabled.

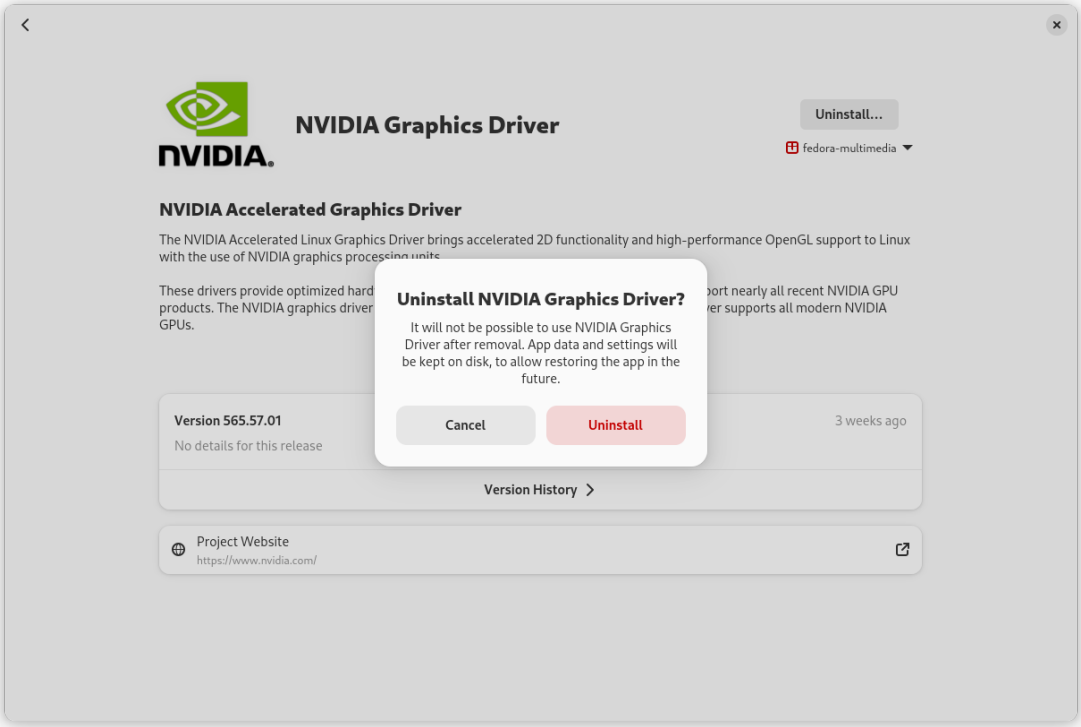


18.4. Uninstallation

1. Remove the driver by clicking *Uninstall* in the GNOME Software window:



2. In the uninstall prompt, click again on the *Uninstall* button,
   - Enter the Administrator password (`root`) and GNOME Software will proceed to uninstall the driver:



3. Optionally, you can remove the enrolled MOK by typing the following command in a terminal:

```
# mokutil --delete /var/lib/dkms/mok.pub
```

Chapter 19. Optimus Laptops and Multi GPU Desktop Systems

This section is specifically for desktop systems with multiple GPUs and is applicable to the following type of systems:

- Optimus laptops (for example integrated Intel GPU + discrete NVIDIA GPU).
- Desktop workstation with GPUs from different vendors (for example integrated AMD GPU + discrete NVIDIA GPU).
- Desktop workstation with multiple NVIDIA GPUs.

There is not much to configure and since a few years, for most common cases, everything just works out of the box, even when the NVIDIA driver is not in use.

Warning: On Optimus laptops, the usage of any external software that is written with the idea of forcing the NVIDIA GPU to be always on is **strongly discouraged**. This includes bbswitch, Bumblebee, SUSEPrime, envycontrol and nvidia-prime. Most of these projects are actually abandoned and usually create more issues than benefits.

The exception to this is switcheroo-control, which is well integrated with the system. Most of the time it carries a package name of `switcherooctl` and is usually preinstalled by default with the desktop on most distributions.

Let's take into consideration a very common case, a laptop with Intel + NVIDIA GPU (Dell Precision 5680, NVIDIA Optimus). The GPUs appear as follows:

```
$ lspci | grep -i vga
00:02.0 VGA compatible controller: Intel Corporation Raptor Lake-P [Iris Xe Graphics]
↪(rev 04)
01:00.0 VGA compatible controller: NVIDIA Corporation AD104GLM [RTX 3500 Ada
↪Generation Laptop GPU] (rev a1)
```

We'll be using the output of the `lspci` command to get the `<bus#>:<device#>.<function#>` of each GPU. This will be used throughout this section to differentiate between the Intel GPU (`00:02.0`) and the NVIDIA GPU (`01:00.0`).

19.1. Power Management

The system boots with the graphical output driven by the integrated Intel GPU and the NVIDIA GPU is off:

```
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
suspended
```

A simple command that touches the PCI card like `lspci` or `nvidia-settings` is enough to wake up the NVIDIA GPU for probing:

```
$ lspci > /dev/null
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
active
```

A few seconds after, the GPU is again in suspended:

```
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
suspended
```

The transition is always fast if no program is using the GPU; it usually takes just 4 or 5 seconds for the GPU to turn off. For example, after exiting a game, you can immediately hear the fan shutting down when the GPU goes off.

This is the simplest way to check the power state of the GPU when using the open source DRM drivers or the NVIDIA driver.

19.1.1. VGA Switcheroo (DRM Drivers Only)

If you are using an open source driver, the VGA Switcheroo files appear as soon as two GPU drivers and one handler have registered with `vga_switcheroo`. Since multiple GPUs are using a common framework, `vga_switcheroo` is enabled and we we can manipulate the state of the devices. The default GPU is marked with a * symbol:

```
$ sudo cat /sys/kernel/debug/vgaswitcheroo/switch
0:IGD:+:Pwr:0000:00:02.0
1:DIS-Audio: :DynOff:0000:01:00.1
2:DIS: :DynOff:0000:01:00.0
```

After firing up the GPU for a workload, we can see the state reflected into the virtual file:

```
$ sudo cat /sys/kernel/debug/vgaswitcheroo/switch
0:IGD:+:Pwr:0000:00:02.0
1:DIS-Audio: :DynOff:0000:01:00.1
2:DIS: :DynPwr:0000:01:00.0
```

The `DIS-Audio` device (Discrete Audio) is the actual HDA sound card on the GPU that is used to send audio to an external output (for example, to a TV connected to an HDMI port). That is also controlled by the dynamic control of the devices via VGA Switcheroo.

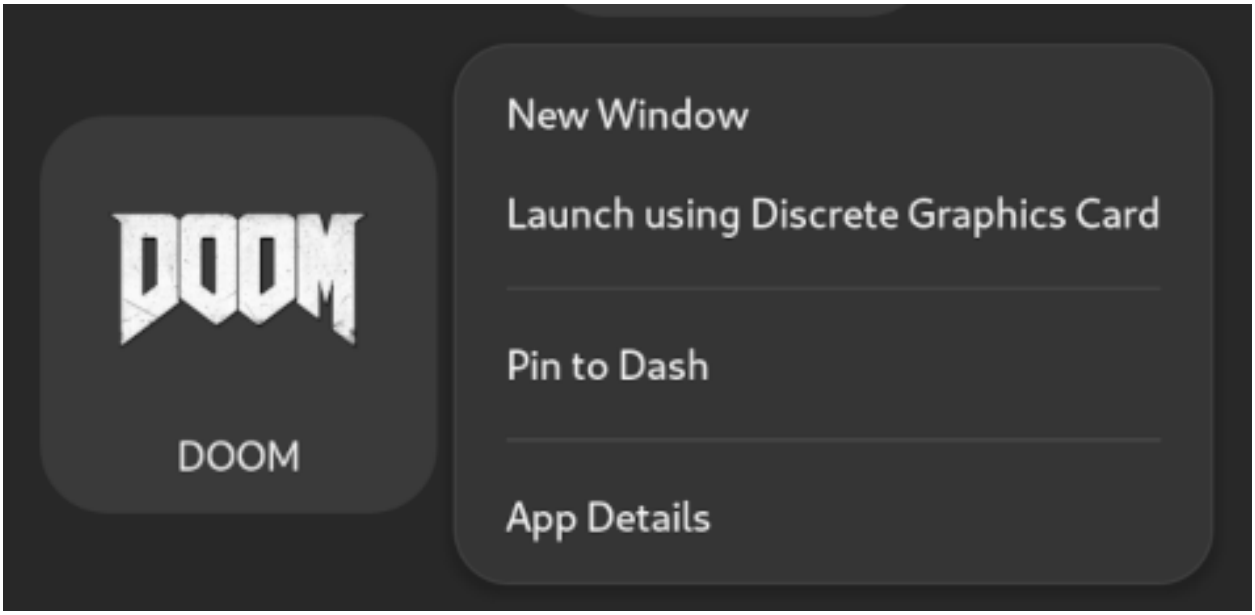
The configuration is flexible, so for example you could have two or more discrete GPUs and one extra audio controller for an eventual HDMI port.

You can also do some really low-level stuff. This is an example command to switch the display output to the discrete GPU, if you have an old system with disconnected GPUs that uses a MUX to switch the display output:

```
$ sudo echo MDIS > /sys/kernel/debug/vgaswitcheroo/switch
```

19.2. Selecting the GPU to Use when Running a Program from the Desktop

If running on Gnome or KDE, **any application can be selected to run on the discrete GPU directly from the desktop by right clicking on the application icon**:

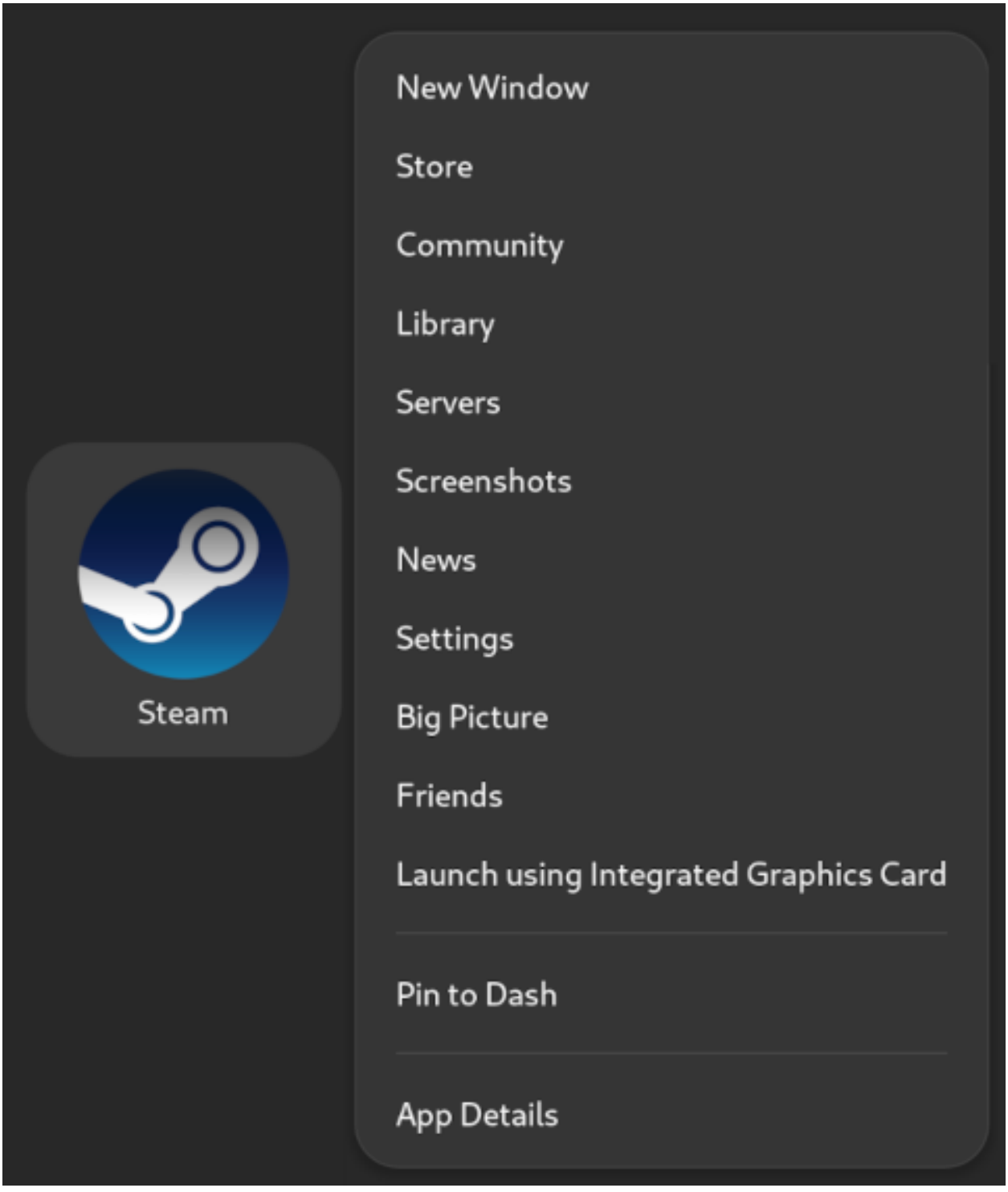


This is supported both in the case of multiple DRI/DRM devices and or a combination with NVIDIA proprietary drivers. There is no visible difference between the two.

Both Gnome and KDE feature an extra setting that can be added to desktop menus to prefer the integrated GPU. For example Steam provides this by default:

```
$ cat /usr/share/applications/steam.desktop | grep -i GPU
PrefersNonDefaultGPU=true
X-KDE-RunOnDiscreteGpu=true
```

Applications bearing those entries receive the **opposite** treatment, **they run on the discete GPU by default. You can right click on the Steam application icon and select the internal GPU**:


New Window
Store
Community
Library
Servers
Screenshots
News
Settings
Big Picture
Friends
Launch using Integrated Graphics Card
Pin to Dash
App Details
Steam

19.3. Selecting the GPU to Use with `switcherooctl`

The system comes with a userspace utility to manipulate the GPUs and that also prints the variables you can use to address a specific GPU. This is the Prime / VGA Swicheroo case:

```
$ switcherooctl list
Device: 0
  Name:        Intel Corporation Raptor Lake-P [Iris Xe Graphics]
  Default:     yes
  Environment: DRI_PRIME=pci-0000_00_02_0

Device: 1
  Name:        NVIDIA Corporation AD104GLM [RTX 3500 Ada Generation Laptop GPU]
  Default:     no
  Environment: DRI_PRIME=pci-0000_01_00_0
```

Note: The `DRI_PRIME` variable is never set by default and it's assumed to be at 0 (so main integrated GPU in most cases) if nothing else sets it.

In the case of NVIDIA drivers, the tool is smart enough to set the appropriate NVIDIA variables to achieve the same result:

```
$ switcherooctl list
Device: 0
  Name:        Intel Corporation Raptor Lake-P [Iris Xe Graphics]
  Default:     yes
  Environment: DRI_PRIME=pci-0000_00_02_0

Device: 1
  Name:        NVIDIA Corporation AD104GLM [RTX 3500 Ada Generation Laptop GPU]
  Default:     no
  Environment: __GLX_VENDOR_LIBRARY_NAME=nvidia __NV_PRIME_RENDER_OFFLOAD=1 __VK_
↪LAYER_NV_optimus=NVIDIA_only
```

Think of `switcherooctl` as a replacement for setting up variables. For example, if your system has 4 GPUs (0-3) and you want to target the 4th GPU, these commands are equivalent:

```
$ switcherooctl launch -g 3 <command>
$ DRI_PRIME=3 <command>
$ DRI_PRIME=pci-0000_03_00_0 <command>
```

19.4. Selecting the GPU to Use with Environment Variables

19.4.1. OpenGL Context

OpenGL came in before this multiple GPU / multiple GPU vendor topic existed. By default, with GLVND that offers multiple GPU support for OpenGL, the first GPU is the one used to run OpenGL applications in the main display and the second GPU is left powered off:

```
$ glxinfo -B | grep string
OpenGL vendor string: Intel
OpenGL renderer string: Mesa Intel(R) Graphics (RPL-P)
OpenGL core profile version string: 4.6 (Core Profile) Mesa 24.0.8
OpenGL core profile shading language version string: 4.60
OpenGL version string: 4.6 (Compatibility Profile) Mesa 24.0.8
OpenGL shading language version string: 4.60
OpenGL ES profile version string: OpenGL ES 3.2 Mesa 24.0.8
OpenGL ES profile shading language version string: OpenGL ES GLSL ES 3.20
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
suspended
```

In the case of the Intel + NVIDIA drivers, we can use the NVIDIA variables consumed by the driver to select the GPU and let the system power on the extra GPU:

```
$ __NV_PRIME_RENDER_OFFLOAD=1 __GLX_VENDOR_LIBRARY_NAME=nvidia glxinfo -B | grep
↪string
OpenGL vendor string: NVIDIA Corporation
OpenGL renderer string: NVIDIA RTX 3500 Ada Generation Laptop GPU/PCIe/SSE2
OpenGL core profile version string: 4.6.0 NVIDIA 555.42.02
OpenGL core profile shading language version string: 4.60 NVIDIA
OpenGL version string: 4.6.0 NVIDIA 555.42.02
OpenGL shading language version string: 4.60 NVIDIA
OpenGL ES profile version string: OpenGL ES 3.2 NVIDIA 555.42.02
OpenGL ES profile shading language version string: OpenGL ES GLSL ES 3.20
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
active
```

If we are using open source drivers for both GPUs, we can use the Mesa DRI variables to select the GPU:

```
$ DRI_PRIME=1 glxinfo -B | grep string
OpenGL vendor string: Mesa
OpenGL renderer string: NV194
OpenGL core profile version string: 4.3 (Core Profile) Mesa 24.0.8
OpenGL core profile shading language version string: 4.30
OpenGL version string: 4.3 (Compatibility Profile) Mesa 24.0.8
OpenGL shading language version string: 4.30
OpenGL ES profile version string: OpenGL ES 3.2 Mesa 24.0.8
OpenGL ES profile shading language version string: OpenGL ES GLSL ES 3.20
```

We can either use the PCI or VGA Switcheroo devices to check the power state of the GPUs:

```
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
active
$ sudo cat /sys/kernel/debug/vgaswitcheroo/switch
0:IGD:+:Pwr:0000:00:02.0
1:DIS-Audio: :DynOff:0000:01:00.1
2:DIS: :DynPwr:0000:01:00.0
```

19.4.2. VA-API (Video Acceleration API) Context

The Video Acceleration API can also be used to display GPU information by using the `vainfo` command:

```
$ vainfo | grep version
libva info: VA-API version 1.21.0
libva info: Trying to open /usr/lib64/dri/iHD_drv_video.so
libva info: Found init function __vaDriverInit_1_21
libva info: va_openDriver() returns 0
vainfo: VA-API version: 1.21 (libva 2.21.0)
vainfo: Driver version: Intel iHD driver for Intel(R) Gen Graphics - 24.2.3 (Full
↪Feature Build)
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
suspended
```

VA-API has its own set of variables for selecting which driver to use in the case of Intel + NVIDIA drivers:

```
$ LIBVA_DRIVER_NAME=nvidia vainfo | grep version
libva info: VA-API version 1.21.0
libva info: User environment variable requested driver 'nvidia'
libva info: Trying to open /usr/lib64/dri/nvidia_drv_video.so
libva info: Found init function __vaDriverInit_1_0
libva info: va_openDriver() returns 0
vainfo: VA-API version: 1.21 (libva 2.21.0)
vainfo: Driver version: VA-API NVDEC driver [direct backend]
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
active
```

Again, with the open source stack, also setting the DRI variable or `switcherooctl` are required to run the command on the correct GPU:

```
$ DRI_PRIME=1 LIBVA_DRIVER_NAME=nouveau vainfo | grep version
libva info: VA-API version 1.21.0
libva info: User environment variable requested driver 'nouveau'
libva info: Trying to open /usr/lib64/dri/nouveau_drv_video.so
libva info: Found init function __vaDriverInit_1_21
libva info: va_openDriver() returns 0
vainfo: VA-API version: 1.21 (libva 2.21.0)
vainfo: Driver version: Mesa Gallium driver 24.0.8 for NV194
$ cat /sys/bus/pci/devices/0000:{00:02.0,01:00.0}/power/runtime_status
active
active
```

19.4.3. VDPAU Context

There is no support for Optimus/Prime laptops in VDPAU and no support for Wayland.

19.4.4. Vulkan or EGL Context

Vulkan and EGL were thought with this use case in mind and the selection of the GPU to use ties into the extensions, so usually the correct one is already considered by the program using the appropriate API. The program can query a particular extension to get an ordered list of GPUs or with some other mechanism. This is usually performed by the program itself, so there is not really a way to "force" one specific GPU.

For example, `vkcube` allows us to select the GPU:

```
$ vkcube --gpu_number 0 --c 20
Selected GPU 0: Intel(R) Graphics (RPL-P), type: IntegratedGpu
$ vkcube --gpu_number 1 --c 20
Selected GPU 1: NVIDIA RTX 3500 Ada Generation Laptop GPU, type: DiscreteGpu
```

Contrary to the OpenGL context, you can use with the following commands to display a list of GPUs to use, not just for a single GPU:

```
$ eglinfo -B
$ __NV_PRIME_RENDER_OFFLOAD=1 eglinfo -B
$ vulkaninfo --summary
```

There are some variables or programs that can be used to influence the extensions used for querying the GPUs, but it's not really a supported path. The application decides based on the information provided by the drivers and some predefined criteria.

19.4.5. Forcing the Usage of X on a Specific GPU in a Wayland Context

Everything described so far is applied as well to Wayland. On top of that, Xwayland is started whenever an application that does not support Wayland yet is started in a Wayland desktop.

If you want to force the use of Xwayland for a program that supports both Wayland and X, then you just need to set an additional variable. For example, depending on the context (DRI, NVIDIA, etc), these are all equivalent:

```
$ XDG_SESSION_TYPE=X11 __NV_PRIME_RENDER_OFFLOAD=1 __GLX_VENDOR_LIBRARY_NAME=nvidia
↪glxgears
$ XDG_SESSION_TYPE=X11 DRI_PRIME=1 glxgears
$ XDG_SESSION_TYPE=X11 switcherooctl launch -g 1 glxgears
```

Chapter 20. Advanced Options

This section contains information on some advanced setup scenarios which are not covered in the basic instructions above.

20.1. Switching between Driver Module Flavors

Use the following steps to switch between the NVIDIA driver proprietary and open module flavors on your system.

Note: Replace *XXX* with the NVIDIA driver branch number such as 575.

Amazon Linux 2023, KylinOS 10, Red Hat Enterprise Linux 8/9, Rocky Linux 8/9, Oracle Linux 8/9

To switch between proprietary and open kernel modules:

```
# dnf -y module switch-to nvidia-driver:<stream> --allowerasing
```

Fedora 41

To switch from proprietary to open:

```
# dnf install --allowerasing nvidia-open
```

To switch from open to proprietary:

```
# dnf install --allowerasing cuda-drivers
```

If you have done a desktop or compute-only installation, you can just switch the kernel module package. For example to go from proprietary to open:

```
# dnf install --allowerasing kmod-nvidia-open-dkms
```

Azure Linux 2/3

Only the open kernel modules are supported, no switching is possible.

Debian 12

To switch from proprietary to open:

```
# apt install --autoremove --purge nvidia-open
```

To switch from open to proprietary:

```
# apt install --autoremove --purge cuda-drivers
```

Ubuntu 20.04/22.04/24.04

To switch from proprietary to open:

```
# apt install --autoremove --purge nvidia-open-575 nvidia-driver-575-open
```

To switch from open to proprietary:

```
# apt install --autoremove --purge cuda-drivers-575 nvidia-driver-575
```

OpenSUSE Leap 15, SUSE Linux Enterprise Server 15

To switch from proprietary to open:

```
# zypper install --details --force-resolution nvidia-open
```

To switch from open to proprietary:

```
# zypper install --details --force-resolution cuda-drivers
```

20.2. Meta Packages

Meta packages are `rpm`/`deb` 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. Not all of the packages listed below are available on every distribution. Sometime they are provided by other packages, sometimes they are not needed as there are other mechanisms offered by the distribution to provide branch segregation (ex. DNF modules).

Table 6: Kernel Modules Meta Packages

Meta Package	Purpose
`nvidia-open`	Installs all NVIDIA Open GPU kernel modules Driver packages. Handles upgrading to the next version of the driver packages when they are released.
`nvidia-open-575`	Installs all NVIDIA Open GPU kernel modules Driver packages. Will not upgrade beyond the 575.xxx branch drivers.
`cuda-drivers`	Installs all NVIDIA proprietary kernel modules Driver packages. Handles upgrading to the next version of the Driver packages when they are released.
`cuda-drivers-575`	Installs all NVIDIA proprietary kernel modules Driver packages. Will not upgrade beyond the 575.xx branch drivers.

20.3. Package Upgrades

When a new version is available, a normal package update command specific for the distribution should suffice in upgrading the driver. The various differences in distributions would take care of obsolency and package switching when performing upgrades to a different branch.

The following section shows some examples of the commands that can be performed.

20.3.1. Red Hat Enterprise Linux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, KylinOS 10, Amazon Linux 2023

When upgrading the driver to the **same** stream:

```
# dnf update
```

When upgrading the driver to a **different** stream:

```
# dnf module reset nvidia-driver
# dnf module enable nvidia-driver:<stream>
# dnf update --allowerasing
```

Or when switching from proprietary modules to open modules:

```
# dnf module reset nvidia-driver
# dnf module enable nvidia-driver:<stream>
# dnf install nvidia-open --allowerasing
```

20.3.2. Fedora 41

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the DNF 4/5 paragraph of the *Version locking* section for more information.

When upgrading the driver, whether configured to a version locked branch or the latest available, the command to execute is always the same; what matters is the DNF version lock configuration:

```
# dnf update
```

When switching from proprietary modules to open modules:

```
# dnf install nvidia-open --allowerasing
```

This will remove any package which would have dependencies removed.

20.3.3. Azure Linux 2/3

When upgrading the driver to the **same** stream:

```
# tdnf update
```

When upgrading the driver to a **different** stream:

```
# tdnf install --allowerasing nvidia-open-575
```

This will remove any package which would have dependencies removed.

20.3.4. SUSE Enterprise Linux Server 15, OpenSUSE Leap 15

When upgrading the driver to the **same** stream:

```
# zypper update --details
```

When upgrading the driver to a **different** stream:

```
# zypper install --details --force-resolution nvidia-open-575
```

This will remove any package which would prevent you from reaching the target.

20.3.5. Debian 12

If APT pinning is configured on the sytem, please adjust the configuration or remove the pinning entirely. Please refer to the APT paragraph of the *Version locking* section for more information.

When upgrading the driver, whether configured to a pinned branch or the latest available, the command to execute is always the same; what matters is the APT pinning configuration:

```
# dnf update
```

Or a bit more aggressive, to take into consideration held back packages that might have changed betweend driver releases, for example to upgrade from `nvidia-open-565` to `nvidia-open-575`:

```
# apt dist-upgrade --autoremove --purge
```

This will remove any package which would have dependencies removed.

20.3.6. Ubuntu 20.04/22.04/24.04

When updating packages of the same driver branch or without a specific tracking branch:

```
# apt update
```

Or a bit more aggressive, to take into consideration held back packages that might have changed betweend driver releases, for example to upgrade from `nvidia-open-565` to `nvidia-open-575`:

```
# apt dist-upgrade --autoremove --purge
```

This will remove any package which would have dependencies removed.

20.4. Red Hat Enterprise Linux 8/9 Precompiled Streams

Precompiled streams offer an optional method of streamlining the installation process. The advantages of precompiled streams:

- Signed by Red Hat: allows Secure Boot and kernel signature validation to be completely enabled
- Precompiled: faster boot up after driver and/or kernel updates
- Pre-tested: kernel and driver combination has been validated
- Removes compiler and associated dependencies: no compiler installation required
- Removes DKMS dependency: enabling EPEL repository not required
- Removes kernel-devel and kernel-headers dependencies: no black screen if matching packages are missing

When using precompiled drivers, a plugin for the DNF package manager is enabled that prevents system breakages by preventing upgrades to a kernel for which no precompiled driver yet exists. This can delay the application of kernel updates, but ensures that a tested kernel and driver combination is always used. A warning is displayed by `dnf` during that upgrade situation:

```
NVIDIA driver: some kernel packages have been filtered due to missing precompiled
↪modules.
Please run "dnf nvidia-plugin" as a command to see a report on the filter being
↪applied.
```

Additional information is shown for all kernels and precompiled modules that are available for the system by running the plugin as a standlone command. For example:

```
# dnf nvidia-plugin
Last metadata expiration check: 0:27:16 ago on Wed 20 Nov 2024 08:09:53 PM CET.

Installed kernel(s):
  kernel-core-5.14.0-503.14.1.el9_5.x86_64
  kernel-core-5.14.0-427.42.1.el9_4.x86_64

Available kernel(s):
  kernel-core-5.14.0-503.11.1.el9_5.x86_64
```

(continues on next page)

(continued from previous page)

```
  kernel-core-5.14.0-503.14.1.el9_5.x86_64

Available driver(s):
  nvidia-driver-3:565.57.01-1.el9.x86_64
  nvidia-driver-cuda-3:565.57.01-1.el9.x86_64

Available kmod(s):
  kmod-nvidia-3:565.57.01-2.el9.x86_64
```

Packaging templates and instructions are provided on GitHub to allow you to maintain your own precompiled kernel module packages for custom kernels and derivative Linux distros: NVIDIA/yum-packaging-precompiled-kmod

To use the new driver packages on RHEL 8 or RHEL 9:

1. First, ensure that the necessary Red Hat repositories are enabled.

   Compared to the normal DKMS installation, this requires less repositories to be enabled on the system.

   Red Hat Enterprise Linux 8:

   ```
   # subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms
   # subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
   ```

   Red Hat Enterprise Linux 9:

   ```
   # subscription-manager repos --enable=rhel-9-for-x86_64-appstream-rpms
   # subscription-manager repos --enable=rhel-9-for-x86_64-baseos-rpms
   ```

2. Choose one of the options below depending on the desired driver:

   - `latest` always updates to the highest versioned driver (precompiled):

     ```
     # dnf module enable nvidia-driver:latest
     ```

   - Locks the driver updates to the specified driver branch (precompiled). Replace `<id>` with the appropriate driver branch streams, for example 570, 560, 550, etc.:

     ```
     # dnf module enable nvidia-driver:<id>
     ```

3. Install the preferred driver combination:

   For example:

   ```
   # dnf install nvidia-open
   # dnf install nvidia-driver-cuda
   # dnf install nvidia-driver nvidia-settings
   # dnf install nvidia-driver nvidia-driver-cuda
   ```

Please refer to the precompiled folder in the driver repositories for more information.

20.4.1. Precompiled Streams Support Matrix

This table shows an example of the supported precompiled, legacy, and open DKMS streams for each driver.

Table 7: Streams Support Matrix

NVIDIA Driver	Precompiled Stream	Legacy DKMS Stream	Open DKMS Stream
Highest version	latest	latest-dkms	open-dkms
Locked at 575.x	575	575-dkms	575-open
Locked at 570.x	570	570-dkms	570-open
Locked at 565.x	565	565-dkms	565-open
Locked at 560.x	560	560-dkms	560-open
Locked at 550.x	550	550-dkms	550-open

Prior to switching between module streams, first reset the DNF module:

```
# dnf module reset nvidia-driver
```

Or as an alternative:

```
# dnf module switch-to nvidia-driver:<stream>
```

20.5. Modularity Profiles

Modularity profiles work with any supported modularity stream and allow for additional use cases. These modularity profiles are available on Amazon Linux 2023, KylinOS 10, and Red Hat Enterprise Linux 8/9.

Table 8: Modularity Profiles

Stream	Profile	Use Case
Default	`/default`	Installs all the driver packages in a stream.
Kickstart	`/ks`	Performs unattended Linux OS installation using a config file.
NVSwitch Fabric	`/fm`	Installs all the driver packages plus components required for boot-strapping an NVSwitch system (including the Fabric Manager and NSCQ telemetry).

For example:

```
# dnf module install nvidia-driver:<stream>/default
# dnf module install nvidia-driver:<stream>/ks
# dnf module install nvidia-driver:<stream>/fm
```

You can install multiple modularity profiles using BASH curly brace expansion, for example:

```
# dnf module install nvidia-driver:latest/{default,fm}
```

Please refer to the Developer Blog for more information.

20.6. Red Hat Enterprise Linux 8/9 Kickstart Installation

Edit the Kickstart configuration file as follows.

Red Hat Enterprise Linux 8, Rocky Linux 8, Oracle Linux 8

1. Enable the EPEL repository:

```
repo --name=epel --baseurl=http://download.fedoraproject.org/pub/epel/8/
↪Everything/x86_64/
```

2. Enable the CUDA repository:

```
repo --name=cuda-rhel8 --baseurl=https://developer.download.nvidia.com/compute/
↪cuda/repos/rhel8/x86_64/
```

3. If you are not using driver 570 or later, in the `packages` section of the Kickstart file, make sure you are using the */ks* profile:

```
@nvidia-driver:open-dkms/ks
```

This is not needed from 570 and above, you can skip the */ks* profile.

1. Perform the *Post-installation Actions*.

Red Hat Enterprise Linux 9, Rocky Linux 9, Oracle Linux 9

1. Enable the EPEL repository:

```
repo --name=epel --baseurl=http://download.fedoraproject.org/pub/epel/9/
↪Everything/x86_64/
```

2. Enable the CUDA repository:

```
repo --name=cuda-rhel9 --baseurl=https://developer.download.nvidia.com/compute/
↪cuda/repos/rhel9/x86_64/
```

3. If you are not using driver 570 or later, in the `packages` section of the Kickstart file, make sure you are using the **/ks** profile:

```
@nvidia-driver:open-dkms/ks
```

4. Perform the *Post-installation Actions*.

20.7. Version locking

For distributions without mechanisms to clearly segregate branches inside the same repository, the normal distribution version locking mechanisms can be used. Here are some examples on how to achieve locking onto a specific driver branch.

The examples below all refer to configuring version lock for branch 575. Make sure every package that you want to lock onto a specific branch has the appropriate line / code block in the configuration files.

20.7.1. DNF 4

DNF version 4 is the package manager of the following distributions:

- Red Hat Enterprise Linux 8 / Rocky Linux 8 / Oracle Linux 8
- Red Hat Enterprise Linux 9 / Rocky Linux 9 / Oracle Linux 9
- Kylin 10
- Amazon Linux 2023

Make sure the `python3-dnf-plugin-versionlock` package is installed to use it. Just run the `dnf versionlock` command to automatically populate the file `/etc/dnf/plugins/versionlock.list` and lock a specific driver version in place:

```
# dnf4 versionlock \*nvidia\*575\*
Adding versionlock on: kmod-nvidia-open-dkms-3:575.51-1.fc41.*
Adding versionlock on: nvidia-kmod-common-3:575.51-1.fc41.*
Adding versionlock on: nvidia-driver-cuda-libs-3:575.51-1.fc41.*
Adding versionlock on: nvidia-open-3:575.51-1.fc41.*
Adding versionlock on: nvidia-xconfig-3:575.51-1.fc41.*
Adding versionlock on: xorg-x11-nvidia-3:575.51-1.fc41.*
Adding versionlock on: kmod-nvidia-latest-dkms-3:575.51-1.fc41.*
Adding versionlock on: libnvidia-cfg-3:575.51-1.fc41.*
Adding versionlock on: nvidia-driver-cuda-3:575.51-1.fc41.*
Adding versionlock on: nvidia-driver-3:575.51-1.fc41.*
Adding versionlock on: libnvidia-fbc-3:575.51-1.fc41.*
Adding versionlock on: nvidia-libXNVCtrl-devel-3:575.51-1.fc41.*
Adding versionlock on: nvidia-libXNVCtrl-3:575.51-1.fc41.*
Adding versionlock on: libnvidia-ml-3:575.51-1.fc41.*
Adding versionlock on: nvidia-modprobe-3:575.51-1.fc41.*
Adding versionlock on: nvidia-settings-3:575.51-1.fc41.*
Adding versionlock on: nvidia-driver-libs-3:575.51-1.fc41.*
Adding versionlock on: nvidia-persistenced-3:575.51-1.fc41.*

# cat /etc/dnf/plugins/versionlock.list
kmod-nvidia-open-dkms-3:575.51-1.fc41.*
nvidia-kmod-common-3:575.51-1.fc41.*
nvidia-driver-cuda-libs-3:575.51-1.fc41.*
nvidia-open-3:575.51-1.fc41.*
nvidia-xconfig-3:575.51-1.fc41.*
xorg-x11-nvidia-3:575.51-1.fc41.*
kmod-nvidia-latest-dkms-3:575.51-1.fc41.*
libnvidia-cfg-3:575.51-1.fc41.*
nvidia-driver-cuda-3:575.51-1.fc41.*
```

(continues on next page)

(continued from previous page)

```
nvidia-driver-3:575.51-1.fc41.*
libnvidia-fbc-3:575.51-1.fc41.*
nvidia-libXNVCtrl-devel-3:575.51-1.fc41.*
nvidia-libXNVCtrl-3:575.51-1.fc41.*
libnvidia-ml-3:575.51-1.fc41.*
nvidia-modprobe-3:575.51-1.fc41.*
nvidia-settings-3:575.51-1.fc41.*
nvidia-driver-libs-3:575.51-1.fc41.*
nvidia-persistenced-3:575.51-1.fc41.*
```

An alternative is to manually edit the configuration file `/etc/dnf/plugins/versionlock.list` and populate it with the following content to stick the driver to a particular branch, and no longer to a specific version:

```
kmod-nvidia*3:575*
libnvidia*3:575*
nvidia-driver*3:575*
nvidia-kmod-common-3:575*
nvidia-libXNVCtrl*3:575*
nvidia-modprobe-3:575*
nvidia-open-3:575*
nvidia-persistenced-3:575*
nvidia-settings-3:575*
nvidia-xconfig-3:575*
xorg-x11-nvidia-3:575*
```

Please refer to the `dnf4-versionlock(8)` manual page for more information and configuration options.

20.7.2. DNF 5

DNF version 5 is the package manager of the following distributions:

- Fedora 41

Just run the `dnf versionlock add` command to populate the file `/etc/dnf/versionlock.toml` (which can then be edited or left as is) to lock a specific driver version:

```
# dnf versionlock add \*nvidia\*575\*
Updating and loading repositories:
Repositories loaded.
Adding versionlock on "dkms-nvidia = 3:575.51.03-1.fc41".
Adding versionlock on "libnvidia-cfg = 3:575.51.03-1.fc41".
Adding versionlock on "libnvidia-fbc = 3:575.51.03-1.fc41".
Adding versionlock on "libnvidia-gpucomp = 3:575.51.03-1.fc41".
Adding versionlock on "libnvidia-ml = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-driver = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-driver-cuda = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-driver-cuda-libs = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-driver-libs = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-kmod-common = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-libXNVCtrl = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-modprobe = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-persistenced = 3:575.51.03-1.fc41".
Adding versionlock on "nvidia-settings = 3:575.51.03-1.fc41".
```

Alternatively, DNF 5 can be configured to specify a range in the configuration file, but this is very verbose. Configure the file `/etc/dnf/versionlock.toml` with the following content:

```
version = "1.0"

[[packages]]
name = "kmod-nvidia*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "libnvidia*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-driver*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-kmod-common*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-libXNVCtrl*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
```

(continues on next page)

(continued from previous page)

```
value = "3:580"

[[packages]]
name = "nvidia-modprobe*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-open*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-persistenced*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-settings*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"

[[packages]]
name = "nvidia-xconfig*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"
```

(continues on next page)

(continued from previous page)

```
[[packages]]
name = "xorg-x11-nvidia*"
[[packages.conditions]]
key = "evr"
comparator = ">="
value = "3:575"
[[packages.conditions]]
key = "evr"
comparator = "<"
value = "3:580"
```

Please refer to the `dnf5-versionlock(8)` manual page for more information and configuration options.

20.7.3. APT

APT is the package manager of the following distributions:

- Debian 12
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

Create a configuration file like `/etc/apt/preferences.d/nvidia` and populate it with the following content to pin the driver to a particular branch and/or version:

```
Package: src:*nvidia*:any src:cuda-drivers:any src:cuda-compat:any
Pin: version 570.148.05-1
Pin-Priority: 1000
```

A priority of >= 1000 allows APT to also consider downgrades with a higher priority. Please refer to the `apt_preferences(5)` manual page for more information and configuration options.

20.8. SUSE Vendor Change

Starting with driver version 575, all the driver packages for openSUSE Leap and SUSE Enterprise Linux Server have a vendor of “NVIDIA”. The system will prevent the update of packages hosted in the other repositories unless there is an override for the vendor.

More information on the topic is available in the Vendor Change page of openSUSE.

This can be triggered on the command line:

```
# zypper update --allow-vendor-change
Loading repository data...
Reading installed packages...

The following package is going to be upgraded:
  dkms
```

(continues on next page)

(continued from previous page)

```
The following package is going to change vendor:
  dkms  openSUSE -> NVIDIA

1 package to upgrade, 1 to change vendor.

Package download size:    58.0 KiB

Package install size change:
              |     148.7 KiB  required by packages that will be installed
     5.4 KiB  |  -  143.3 KiB  released by packages that will be removed

Backend:  classic_rpmtrans
Continue? [y/n/v/...? shows all options] (y):
```

Or by adding the vendor equivalence to a Zypper configuration file. For example, in the case of openSUSE, just add "NVIDIA" at the end of this configuration file:

```
$ cat /etc/zypp/vendors.d/00-openSUSE.conf
[main]
vendors=openSUSE,SUSE,SUSE LLC <https://www.suse.com/>,NVIDIA
```

And then the upgrade can happen allowing the vendor change:

```
# zypper update --details
Loading repository data...
Reading installed packages...

The following package is going to be upgraded:
  dkms  3.0.11-bp156.1.2 -> 3.1.5-1  noarch  test  NVIDIA

1 package to upgrade.

Package download size:    58.0 KiB

Package install size change:
              |     148.7 KiB  required by packages that will be installed
     5.4 KiB  |  -  143.3 KiB  released by packages that will be removed

Backend:  classic_rpmtrans
Continue? [y/n/v/...? shows all options] (y):
```

20.9. Restrict APT to Look for Specific Architectures

Modify Ubuntu's apt package manager to query specific architectures for specific repositories. This is useful when a foreign architecture has been added, causing "404 Not Found" errors to appear when the repository meta-data is updated.

Each repository you wish to restrict to specific architectures must have its `sources.list` entry modified. This is done by modifying the `/etc/apt/sources.list` file and any files containing repositories you wish to restrict under the `/etc/apt/sources.list.d/` directory. Normally, it is sufficient

to modify only the entries in `/etc/apt/sources.list`.

An architecture-restricted repository entry looks like:

```
deb [arch=<arch1>,<arch2>] <url>
```

For example, if you wanted to restrict a repository to only the amd64 and i386 architectures, it would look like:

```
deb [arch=amd64,i386] <url>
```

It is not necessary to restrict the `deb-src` repositories, as these repositories don't provide architecture-specific packages.

For more details, see the `sources.list` manpage.

20.10. APT Repository File not Found

In case of the error: `E: Failed to fetch file:/var/cuda-repo File not found` on Debian and Ubuntu systems.

This can occur when installing CUDA after uninstalling a different version. Use the following command before installation:

```
# rm -v /var/lib/apt/lists/*cuda* /var/lib/apt/lists/*nvidia*
```

20.11. Verbose Versions when Using APT

Use the `--verbose-versions` flag, for example:

```
# apt install --verbose-versions nvidia-open
```

Chapter 21. Optional Components

21.1. 32 bit (i686) packages for Linux x86_64

These packages provide 32-bit (i686) driver libraries needed for things such as Steam (popular game app store/launcher), older video games, and some compute applications.

21.1.1. Debian 12

```
# dpkg --add-architecture i386
# apt update
# apt install nvidia-driver-libs:i386
```

21.1.2. Ubuntu 20.04/22.04/24.04

```
# dpkg --add-architecture i386
# apt update
# apt install \
   libnvidia-compute-<branch>:i386 libnvidia-decode-<branch>:i386 \
   libnvidia-encode-<branch>:i386 libnvidia-extra-<branch>:i386 \
   libnvidia-fbc1-<branch>:i386 libnvidia-gl-<branch>:i386
```

Where <branch> is the driver version, for example 575.

21.1.3. Red Hat Enterprise Linux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, Fedora 41

```
# dnf install nvidia-driver-cuda-libs.i686 nvidia-driver-libs.i686 libnvidia-ml.i686
↪libnvidia-fbc.i686
```

21.1.4. SUSE Enterprise Linux Server 15, OpenSUSE Leap 15

```
# zypper install nvidia-compute-G06-32bit nvidia-gl-G06-32bit nvidia-video-G06-32bit
```

21.2. GPUDirect Storage

Install NVIDIA Filesystem.

Red Hat Enterprise Linux 8/9

```
# dnf install nvidia-fs
```

Note: The GPUDirect storage module is shipped only in DKMS format; this means it requires the module source to be available. **Starting with version 570 of the drivers, a mix of Precompiled streams and DKMS modules is no longer supported.**

Having a mix of precompiled and DKMS modules makes every single benefit of the precompiled modules disappear completely:

- Development packages and headers required to compile modules are still required.
- Secure Boot can not be used without a custom MOK.
- `nvidia-peermem` can not be recompiled to levarage the OFED installations as it's missing from the precompiled packages.
- It requires two extra packages containing the source of the precompiled modules to be installed along with the binary modules, making the installation more error-prone and complicated.

Until driver version 565, the source profile can be installed with this command to install the addditional packages containing the source of the precompiled modules:

```
# dnf module install nvidia-driver:$stream/src
```

Ubuntu 20.04/22.04/24.04

```
# apt install nvidia-fs
```

21.3. NVSwitch

To install Fabric Manager, NSCQ, NVSDM, IMEX:

Red Hat Enterprise Linux 8/9

```
# dnf module install nvidia-driver:$stream/fm
```

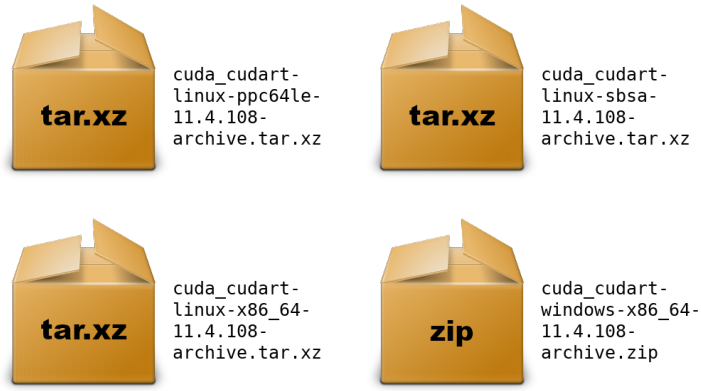
Ubuntu 20.04/22.04/24.04

```
# apt install -V nvidia-fabric-manager-<branch> libnvidia-nscq-<branch> libnvsdm-
↪<branch> nvidia-imex-<branch>
```

Chapter 22. Tarballs 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/nvidia-driver/redist/.



These component `.tar.xz` and `.zip` binary archives do not replace existing packages such as `.deb`, `.rpm`, `.run`, 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_<version>.json`, which corresponds to the Datacenter Driver release label 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.

Chapter 23. Post-installation Actions

The post-installation actions must be manually performed.

23.1. Persistence Daemon

NVIDIA is providing a user-space daemon on Linux to support persistence of driver state across CUDA job runs. The daemon approach provides a more elegant and robust solution to this problem than persistence mode. For more details on the NVIDIA Persistence Daemon, see the documentation here.

All the distribution packages contain a `systemd` preset to enable it automatically if installed as a dependency of other driver components. It can be restarted manually by running:

```
# systemctl restart persistenced
```

23.2. Verify the Driver Version

If you installed the driver, verify that the correct version of it is loaded. When the driver is loaded, the driver version can be found by executing the following command:

```
$ cat /proc/driver/nvidia/version
```

23.3. Local Repository Removal

Removal of the local repository installer is recommended after installation of the driver.

Amazon Linux 2023, Fedora 41, KylinOS 10, Red Hat Enterprise Linux 8/9, Rocky Linux 8/9, Oracle Linux 8/9

```
# dnf remove nvidia-driver-local-repo-$distro*
```

Azure Linux 2/3

```
# tdnf remove nvidia-driver-local-repo-$distro*
```

Ubuntu 20.04/22.04/24.04, Debian 12

```
# apt remove --purge nvidia-driver-local-repo-$distro*
```

OpenSUSE Leap 15, SUSE Linux Enterprise Server 15

```
# zypper remove nvidia-driver-local-repo-$distro*
```

Chapter 24. Removing the Driver

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

Amazon Linux 2023, KylinOS 10, Red Hat Enterprise Linux 8/9, Rocky Linux 8/9, Oracle Linux 8/9

To remove NVIDIA driver:

```
# dnf module remove --all nvidia-driver
# dnf module reset nvidia-driver
```

Fedora 41

To remove NVIDIA driver:

```
# dnf remove nvidia-driver\*
```

Azure Linux 2/3

To remove NVIDIA driver:

```
# tdnf remove nvidia-driver-cuda
# tdnf autoremove
```

OpenSUSE and SLES

To remove NVIDIA driver:

```
# zypper remove \*nvidia\*
```

Ubuntu 20.04/22.04/24.04, Debian 12

To remove NVIDIA driver:

```
# apt remove --autoremove --purge -V nvidia-driver\* libxnvctrl\*
```

Chapter 25. GPG Keys Used to Sign the Packages

The GPG public keys used for the CUDA repository packages are the following:

- `rpm` based distributions: `d42d0685`
- `deb` based distributions: `3bf863cc`

Chapter 26. Additional Considerations

Now that you have CUDA-capable hardware and the NVIDIA driver installed, you can install the CUDA Toolkit. Consult the CUDA Installation Guide, located in https://docs.nvidia.com/cuda/cuda-installation-guide-linux/.

For technical support on installation questions, consult and participate in the developer forums at https://forums.developer.nvidia.com/c/gpu-graphics/linux/148.

Chapter 27. Frequently Asked Questions

27.1. 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 *Compute-only and Desktop Installation* section for details on how to modify your `sources.list` file to prevent these errors.

27.2. 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 *Compute-only and Desktop Installation* section.

If you only have a single NVIDIA GPU in the system that you want to use for compute only (such as in an Optmus laptop), you can perform a compute-only installation, which skips the graphical components of the driver.

27.3. What do I do if the display does not load, or CUDA does not work, after performing a system update?

System updates may include an updated Linux kernel. In many cases, a new Linux kernel will be installed without properly updating the required Linux kernel headers and development packages. To ensure the CUDA driver continues to work when performing a system update, rerun the commands in the *Verify the System has the Correct Kernel Packages Installed* section.

Additionally, when updating kernel components without rebooting, the DKMS framework will sometimes fail to correctly rebuild the NVIDIA kernel module packages when a new Linux kernel is installed. When this happens, it is usually sufficient to invoke DKMS manually by running the appropriate commands in a virtual console, and then rebooting. For example:

```
# dkms status
nvidia/565.57.01: added
```

```
# dkms -m nvidia/565.57.01 -k 6.11.7-300.fc41.x86_64 build

Sign command: /lib/modules/6.11.7-300.fc41.x86_64/build/scripts/sign-file
Signing key: /var/lib/dkms/mok.key
Public certificate (MOK): /var/lib/dkms/mok.pub

Cleaning build area... done.
Building module(s)..... done.
Signing module /var/lib/dkms/nvidia/565.57.01/build/kernel-open/nvidia.ko
Signing module /var/lib/dkms/nvidia/565.57.01/build/kernel-open/nvidia-modeset.ko
Signing module /var/lib/dkms/nvidia/565.57.01/build/kernel-open/nvidia-drm.ko
Signing module /var/lib/dkms/nvidia/565.57.01/build/kernel-open/nvidia-uvm.ko
Signing module /var/lib/dkms/nvidia/565.57.01/build/kernel-open/nvidia-peermem.ko
Cleaning build area... done.
```

```
# dkms -m nvidia/565.57.01 -k 6.11.7-300.fc41.x86_64 install

Module nvidia-565.57.01 for kernel 6.11.7-300.fc41.x86_64 (x86_64):
Before uninstall, this module version was ACTIVE on this kernel.
Deleting /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia.ko.xz
Deleting /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-modeset.ko.xz
Deleting /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-drm.ko.xz
Deleting /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-uvm.ko.xz
Deleting /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-peermem.ko.xz
Running depmod.... done.

Installing /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia.ko.xz
Installing /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-modeset.ko.xz
Installing /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-drm.ko.xz
Installing /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-uvm.ko.xz
Installing /lib/modules/6.11.7-300.fc41.x86_64/extra/nvidia-peermem.ko.xz
Running depmod... done.
```

You can reach a virtual console by hitting CTRL+ALT+F2 at the same time.

27.4. How do I handle “Errors were encountered while processing: glx-diversions”?

This sometimes occurs when trying to install and uninstall Debian packages before 565. Run the following commands:

```
# apt remove --purge glx-diversions nvidia-alternative
```

27.5. Unknown symbols in the kernel modules

The `nvidia.ko` kernel module fails to load, saying some symbols are unknown. For example:

```
nvidia: Unknown symbol drm_open (err 0)
```

Check to see if there are any optionally installable modules that might provide these symbols which are not currently installed.

For the example of the `drm_open` symbol, check to see if there are any packages which provide drm_open and are not already installed. For instance, on Ubuntu, the `linux-image-extra` package provides the DRM kernel module (which provides `drm_open`). This package is optional even though the kernel headers reflect the availability of DRM regardless of whether this package is installed or not.

Then re-run the commands from *Removing the Driver* section.

27.6. Third-party packages

- Canonical provides signed `-server` packages that correspond with NVIDIA Datacenter Driver releases.
- SUSE provides signed kmp packages.

Chapter 28. Notices

28.1. Notice

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

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

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

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

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or

services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

28.2. OpenCL

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

28.3. Trademarks

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

Copyright