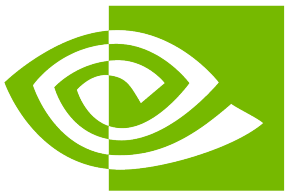


NVIDIA Driver Installation Guide

Release 615

NVIDIA Corporation

Sep 09, 2026

Contents

This guide provides installation instructions for the NVIDIA driver (branch 615).

The **NVIDIA Driver Installation Guide** provides administrators and users with comprehensive instructions and best practices for installing and configuring NVIDIA® GPU drivers on supported Linux distributions. It covers all stages of driver deployment, from system preparation and dependency setup to installation, verification, and maintenance.

The guide explains:

- Preparing systems for driver installation (kernel headers, module support, and OS compatibility).
- Choosing between distribution-specific packages and the generic NVIDIA installer.
- Verifying installation by checking module loading, device enumeration, and feature activation.
- Managing optional components such as NVIDIA Fabric Manager and file-system modules.
- Performing driver updates and aligning versions with CUDA releases and soperating system updates.

With detailed sections on prerequisites, OS support policies, kernel module handling, packaging and installation paths, and verification steps, the guide ensures a stable and optimal driver setup across enterprise, HPC, and datacenter environments.

Chapter 1. Introduction

This section describes the operating system requirements for installing and running the NVIDIA driver and CUDA Toolkit. It lists supported Linux and Windows versions, explains how distribution codenames and architectures are used in commands, and summarizes the validated kernel, compiler, and GLIBC configurations.

1.1. Linux System Requirements

The following table lists the supported Linux distributions.

Note

The values in the “Codename” and “Architecture” columns are used to substitute the `<distribution>` and `<arch>` placeholders across this document.

Table 1: Supported Linux Distributions

Distribution	Codename	Architecture
amd64 systems (x86_64)		
Red Hat Enterprise Linux 10	rhel10	x86_64
Red Hat Enterprise Linux 9	rhel9	x86_64
Red Hat Enterprise Linux 8	rhel8	x86_64
AlmaLinux 10	rhel10	x86_64
AlmaLinux 9	rhel9	x86_64
AlmaLinux 8	rhel8	x86_64
openSUSE Leap 15 SP6	opensuse15	x86_64
openSUSE Leap 16	suse16	x86_64
Rocky Linux 10	rhel10	x86_64
Rocky Linux 9	rhel9	x86_64
Rocky Linux 8	rhel8	x86_64

continues on next page

Table 1 – continued from previous page

Distribution	Codename	Architecture
SUSE Linux Enterprise Server 15 SP6+	sles15	x86_64
SUSE Linux Enterprise Server 16	suse16	x86_64
Ubuntu 26.04 LTS	ubuntu2604	amd64
Ubuntu 24.04 LTS	ubuntu2404	amd64
Ubuntu 22.04 LTS	ubuntu2204	amd64
Debian 12	debian12	amd64
Debian 13	debian13	amd64
Fedora 44	fedora44	x86_64
KylinOS V11 2503	kylin11	x86_64
Azure Linux 3.0	azl3	x86_64
Amazon Linux 2023	amzn2023	x86_64
Oracle Linux 10	rhel10	x86_64
Oracle Linux 9	rhel9	x86_64
Oracle Linux 8	rhel8	x86_64
arm64 systems (SBSA)		
Red Hat Enterprise Linux 10	rhel10	aarch64
Red Hat Enterprise Linux 9	rhel9	aarch64
Red Hat Enterprise Linux 8	rhel8	aarch64
SUSE Linux Enterprise Server 15 SP6+	sles15	aarch64
SUSE Linux Enterprise Server 16	suse16	aarch64
KylinOS V11 2503	kylin11	aarch64
Ubuntu 26.04 LTS	ubuntu2604	arm64
Ubuntu 24.04 LTS	ubuntu2404	arm64
Ubuntu 22.04 LTS	ubuntu2204	arm64
Azure Linux 3.0	azl3	aarch64
Amazon Linux 2023	amzn2023	aarch64

1.2. Red Hat Enterprise Linux compatible distributions

This software supports **Red Hat Enterprise Linux compatible distributions**, including but not limited to **derivatives, rebuilds, and variations** that maintain binary and behavioral compatibility with RHEL.

Please note that **support for these compatible distributions is conditional**. Reported issues are ac-

cepted **only if they can be reliably reproduced on Red Hat Enterprise Linux itself**. Issues that occur exclusively on a compatible distribution and cannot be reproduced on RHEL are considered out of scope for support.

1.3. Windows System Requirements

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

- NVIDIA GPU
- A supported version of Windows

The table below summarizes driver support for GPUs across major Windows OS versions.

Table 2: Supported Windows versions

OS Name	OS Version
Windows 11	25H2
Windows 11	24H2 (SV4) (Germanium)
Windows 11	23H2
Windows 11	22H2 (SV2)
Windows 10	22H2
Windows Server 2022	21H2
Windows Server 2025	24H2

1.4. 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.

1.5. Validated Linux Configurations

The following table lists the same supported Linux distributions together with the specific operating system, kernel, compiler, and GLIBC versions that were validated by QA. Use this table to confirm that your system matches a tested configuration. See the footnotes after the table for vendor kernel support details.

Table 3: Native Linux Distribution Support and Validated OS Versions for CUDA 13.3

Distribution	OS Version	Kernel[1]	Default GCC	GLIBC
x86_64				
RHEL 10	10.1	6.12.0-124	14.3.1	2.39
RHEL 9	9.7	5.14.0-611.5	11.5.0	2.34
RHEL 8	8.10	4.18.0-553	8.5.0	2.28
Rocky Linux 10	10.1	6.12.0-124	14.3.1	2.39
Rocky Linux 9	9.7	5.14.0-611.5	11.5.0	2.34
Rocky Linux 8	8.10	4.18.0-553	8.5.0	2.28
Oracle Linux 9	9	5.14.0-427	11.4.1	2.34
Oracle Linux 8	8	4.18.0-553	8.5.0	2.28
SUSE SLES 16	16.0	6.12.0-160000.5	15.1.1	2.40
SUSE SLES 15	15.7	6.4.0-150600.21	7.5.0	2.38
Ubuntu 24.04 LTS	24.04.4	6.17.0-19	14.3.0	2.39
Ubuntu 22.04 LTS	22.04.5	6.5.0-45	12.3.0	2.35
Debian 13	13.3	6.12.73-1	14.2.0	2.41
Debian 12	12.13	6.1.159	12.2.0	2.36
OpenSUSE Leap 15	15.6	6.4.0-150600.21	7.5.0	2.38
Fedora 43	43	6.17.1-300	15.2.1	2.42
KylinOS V11	V11	6.6.0-32.7	12.3.1	2.28
Amazon Linux 2023	AL2023	6.1.82-99.168	11.4.1	2.34
MSFT Azure Linux	3.0	6.6.64.2-9.azl3	13.2.0	2.38-8
Generic arm64 systems (sbsa)				
RHEL 10	10.1	6.12.0	14.3.1	2.39
RHEL 9	9.7	5.14.0-611.16.1	11.5.0	2.34
RHEL 8	8.10	4.18.0-553	8.5.0	2.28
Ubuntu 22.04 LTS	22.04.5	6.5.0-1019	11.4.0	2.35
Ubuntu 24.04 LTS	24.04.4	6.8.0-87	13.4.0	2.39
SUSE SLES 15	15.7	6.4.0-150700.51	7.5.0	2.38
KylinOS V11	V11	6.6.0	12.3.1	2.38
GRACE only arm64 systems (sbsa)				
Amazon Linux 2023	AL2023	6.12.16-18	11.4.1	2.34
MSFT Azure Linux	3.0	6.6.64.2-9.azl3	13.2.0	2.38-8
Ubuntu 22.04 LTS	22.04.5	6.8.0-1030-nvidia-64k	11.4.0	2.35

continues on next page

Table 3 – continued from previous page

Distribution	OS Version	Kernel[1]	Default GCC	GLIBC
Ubuntu 24.04 LTS	24.04.4	6.8.0-1049-nvidia-64k	13.3.0	2.39
RHEL 10	10.1	6.12.0-124.21.1	14.3.1	2.39
RHEL 9	9.7	5.14.0-611.16.1	11.5.0	2.34
SUSE SLES 15	15.7	6.4.0-150700.51	7.5.0	2.38
Debian 13	13.3	6.12.69	14.2.0	2.41
Debian 12	12.13	6.1.162	12.2.0	2.36
arm64 sbsa Jetson (dGPU + iGPU with OpenRM)				
Ubuntu 24.04 LTS Rel38 (JP7.x) native	24.04	6.14.0-33-generic	13.3.0	2.39
Ubuntu 24.04 LTS Rel38 (JP7.x) cross	24.04	6.8.12-tegra	13.3.0	2.39

Additional information on specific kernel versions supported:

- Red Hat Enterprise Linux (RHEL): https://access.redhat.com/articles/3078
- SUSE Linux Enterprise Server (SLES): https://www.suse.com/support/kb/doc/?id=000019587
- Oracle Linux: https://blogs.oracle.com/scoter/oracle-linux-and-unbreakable-enterprise-kernel-uek-releases

1.6. OS Support Policy

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

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

Chapter 2. 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.
- Select a driver version to install.
- Handle conflicting installation methods.

2.1. Verify You Have a Supported Distribution of Linux

The NVIDIA driver packages are supported on some specific distributions of Linux. Please see the table at *Linux System Requirements*.

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

```
$ hostnamectl
```

2.2. 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 as 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. For more advanced cases, such as custom kernel branches, you should ensure that your kernel headers and sources match the kernel build you 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.

2.2.1. Select a driver version

The Datacenter driver documentation lists all the driver versions available for installation that have been certified for Datacenter usage.

The list also contains release notes for each driver version.

Chapter 3. Choose an Installation Method

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

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 the NVIDIA driver:

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

Additional package manager capabilities are described in the Advanced Options section.

Note

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

Chapter 4. Recent Updates

4.1. Version locking packages for Ubuntu / Debian

Driver pinning packages have been introduced for Debian and Ubuntu to track specific driver branches and versions across the various releases.

For more information please refer to the *version locking section for APT*.

Hint

For the best experience when locking a branch, we suggest to install the pinning package prior to installing the driver.

This has been introduced mainly for these new cases:

- Lock systems into tracking specific driver branches
- Lock systems to specific driver versions
- Allow easy upgrades/downgrades/switching between releases and branches
- Consistency in the documentation (always the same instructions regardless of driver release)

4.1.1. Ubuntu 590 and later packages

Starting with branch 590, the Ubuntu packages will be renamed by removing the branch designation from the package name. Switching branches, installing specific driver versions and specific upgrade / downgrade requirements will be supported through these version locking (pinning) packages.

From 590 onwards, the default experience will be to upgrade the system to the latest drivers; like any other packages in the distribution; while the branch tracking will become an “opt in” choice.

Packages will be renamed as follows, and from 590 onwards, they will keep the same name:

Table 4: Ubuntu 590 package renames

Original name	New name
cuda-drivers-580	*will be removed*
cuda-drivers	*no change*
libnvidia-cfg1-580	libnvidia-cfg1
libnvidia-common-580	libnvidia-common
libnvidia-compute-580	libnvidia-compute
libnvidia-decode-580	libnvidia-decode
libnvidia-encode-580	libnvidia-encode
libnvidia-extra-580	libnvidia-extra
libnvidia-fbc1-580	libnvidia-fbc1
libnvidia-gl-580	libnvidia-gl
libnvidia-gpucomp-580	libnvidia-gpucomp
libnvidia-nscq	*no change*
libnvsdm	*no change*
libxnvctrl0	*no change*
libxnvctrl-dev	*no change*
nvidia-dkms-580	nvidia-dkms
nvidia-dkms-580-open	nvidia-dkms-open
nvidia-driver-580	nvidia-driver
nvidia-fabricmanager	*no change*
nvidia-firmware-580	nvidia-firmware
nvidia-headless-580	nvidia-headless
nvidia-headless-580-open	nvidia-headless-open
nvidia-headless-no-dkms-580	nvidia-headless-no-dkms
nvidia-headless-no-dkms-580-open	nvidia-headless-no-dkms-open
nvidia-imex	*no change*
nvidia-kernel-common-580	nvidia-kernel-common
nvidia-kernel-source-580	nvidia-kernel-source
nvidia-kernel-source-580-open	nvidia-kernel-source-open
nvidia-modprobe	*no change*
nvidia-open-580	*will be removed*
nvidia-open	*no change*
nvidia-persistenced	*no change*

continues on next page

Table 4 – continued from previous page

Original name	New name
nvidia-settings	*no change*
nvidia-xconfig	*no change*
xserver-xorg-video-nvidia-580	xserver-xorg-video-nvidia

Chapter 5. 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 and is required for older GPUs from the Maxwell, Pascal, or Volta architectures. The open-source GPU kernel modules are not compatible with these platforms.
- **Open-source** - These are 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 and suggested 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
```

See also

Version locking
More information on locking specific branches for each Linux distribution.

5.1. 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 5: Open GPU kernel module installation

Distribution	Install the latest	Install a specific branch
Red Hat Enterprise Linux 8/9 AlmaLinux 8/9 Oracle Linux 8/9 KylinOS 11 Amazon Linux 2023 Rocky Linux 8/9	`# dnf module enable nvidia-driver:open-dkms` `# dnf install nvidia-open`	`# dnf module enable nvidia-driver:580-open` `# dnf install nvidia-open`
Red Hat Enterprise Linux 10 AlmaLinux 10 Rocky Linux 10 Oracle Linux 10	`# dnf install nvidia-open`	`# dnf versionlock \*nvidia\*580\*` `# dnf install nvidia-open`
Fedora 44	`# dnf install nvidia-open`	`# dnf versionlock add \*nvidia\*580\*` `# dnf install nvidia-open`
Azure Linux 3	`# dnf install nvidia-open`	`tdnf` does not support any sort of locking.
openSUSE Leap 15/16 SUSE Linux Enterprise Server 15/16	`# zypper install nvidia-open`	`# zypper addlock "*nvidia* >= 590"` `# zypper install nvidia-open`
Debian 12/13	`# apt install nvidia-open`	`# apt install nvidia-driver-pinning-580` `# apt install nvidia-open`
Ubuntu 22.04/24.04/26.04	`# apt install nvidia-open`	`# apt install nvidia-driver-pinning-580` `# apt install nvidia-open`

5.2. 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 6: Proprietary GPU kernel module installation

Distribution	Install the latest	Install a specific branch
Red Hat Enterprise Linux 8/9 AlmaLinux 8/9 Oracle Linux 8/9 KylinOS 11 Amazon Linux 2023 Rocky Linux 8/9	`# dnf module enable nvidia-driver:latest-dkms` `# dnf install cuda-drivers`	`# dnf module enable nvidia-driver:580-dkms` `# dnf install cuda-drivers`
Red Hat Enterprise Linux 10 AlmaLinux 10 Rocky Linux 10 Oracle Linux 10	`# dnf install cuda-drivers`	`# dnf versionlock \*nvidia\*580\*` `# dnf install cuda-drivers`
Fedora 44	`# dnf install cuda-drivers`	`# dnf versionlock add \*nvidia\*580\*` `# dnf install cuda-drivers`
Azure Linux 3	Only the open kernel modules are supported	Only the open kernel modules are supported
openSUSE Leap 15/16 SUSE Linux Enterprise Server 15/16	`# zypper install cuda-drivers`	`# zypper addlock "*nvidia* >= 590"` `# zypper install cuda-drivers`
Debian 12/13	`# apt install cuda-drivers`	`# apt install nvidia-driver-pinning-580` `# apt install cuda-drivers`
Ubuntu 22.04/24.04/26.04	`# apt install cuda-drivers`	`# apt install nvidia-driver-pinning-580` `# apt install cuda-drivers`

Chapter 6. Amazon Linux

This section covers:

- Amazon Linux 2023

6.1. 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*.

6.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

6.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

For `aarch64`:

```
# dnf config-manager --add-repo https://developer.download.nvidia.com/
↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
```

2. Clean DNF repository cache:

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

6.4. 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: `580-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: `580-dkms` or `latest-dkms`

6.5. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

6.6. 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.

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
```

Chapter 7. AlmaLinux

This section covers:

- AlmaLinux 8
- AlmaLinux 9
- AlmaLinux 10

Note

The next step describe the standard installation with DKMS modules as provided by NVIDIA.

An alternative to the standard installation, is the installation with Precompiled Open Kernel modules. For more information on this type of installation, plese refer to the section below *Precompiled Open Kernel Modules*.

7.1. Preparation

1. Perform the *Pre-installation Actions*.
2. Kernel headers and development packages for the currently running kernel can be installed with:
   - **AlmaLinux 9/10:**

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

     If using the *64k* kernel variant on `aarch64`:

     ```
     # dnf install kernel-64k-devel-matched kernel-headers
     ```

   - **AlmaLinux 8:**

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

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.

   - **AlmaLinux 9/10**

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

   - **AlmaLinux 8**

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

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

7.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
 ↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
 ↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

7.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

   For `aarch64`:

   ```
   # dnf config-manager --add-repo https://developer.download.nvidia.com/
    ↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
   ```

2. Clean DNF repository cache:

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

7.4. 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:

- AlmaLinux 8
- AlmaLinux 9

Open Kernel Modules

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

- Example DKMS streams: `580-open` or `open-dkms`

Proprietary Kernel Modules

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

- Example DKMS streams: `580-dkms` or `latest-dkms`
- Example precompiled streams: `580` or `latest`

7.5. 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
```

7.6. 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.

7.6.1. 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
```

7.6.2. 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
```

7.7. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

7.8. 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.

7.8.1. AlmaLinux 8/9

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 kernel modules to open kernel modules:

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

7.8.2. AlmaLinux 10

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the *DNF 4* section of the *Version locking* chapter 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 kernel modules to open kernel modules:

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

This will remove any package which would have dependencies removed.

7.9. Precompiled Open Kernel Modules

These alternative installation instructions are valid only for the following distributions:

- AlmaLinux 9
- AlmaLinux 10

Note

This solution is not provided by NVIDIA but is provided by AlmaLinux. It is, nevertheless, the **recommended way of installing the drivers in Secure Boot required scenarios**.

For more information, please refer to the AlmaLinux wiki page.

Precompiled open kernel modules offer an optional method of streamlining the installation process. The advantages of precompiled modules:

- Secure Boot: the kernel modules are signed by trusted keys in the AlmaLinux kernel
- Simpler dependencies: removes *kernel-devel*, *kernel-headers*, DKMS and GCC compiler requirements
- Simpler updates: the kernel modules are updated and installed like any other package
- Simpler setup: all the required repositories are enabled with one command

The precompiled modules have a much simpler setup, the steps described in the *Preparation* section are not needed.

To proceed with this installation method, proceed directly to enable all the required repositories with one command.

```
# dnf install almalinux-release-nvidia-driver
```

This will enable the following repositories all at once:

- AlmaLinux NVIDIA driver repository.
- AlmaLinux CRB repository.
- NVIDIA's CUDA repository.
- EPEL repository.

Note

The local installer option is not supported, as the AlmaLinux configuration automatically enables the NVIDIA online repository.

Then proceed to install the driver as normal. For example, for a desktop installation along with compute components for CUDA:

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

Same command but with 32 bit support on AlmaLinux 9:

```
# dnf install nvidia-open-kmod nvidia-driver nvidia-driver-cuda nvidia-driver-
↪libs.i686
```

7.9.1. kABI Modules Information

The format of the kernel module packages is *kABI* (kernel ABI), which means there is no direct one to one mapping with kernel versions and kernel module binaries, but instead the dependency between the kernel modules and kernel packages is done through versioned symbol dependencies that are part of the ABI stable list.

An example of matching symbols dependencies:

```
$ rpm -qp --requires kmod-nvidia-open-590.44.01-1.el10_1.x86_64.rpm | grep
↪drm_modeset
kernel(drm_modeset_acquire_fini) = 0x3d1dada1
kernel(drm_modeset_acquire_init) = 0x091eaa38
kernel(drm_modeset_backoff) = 0x9a62789e
kernel(drm_modeset_drop_locks) = 0xed88ec0a
kernel(drm_modeset_lock_all) = 0xd3e9c304
kernel(drm_modeset_lock_all_ctx) = 0xfb8a89da
kernel(drm_modeset_unlock_all) = 0x79fa2207
```

```
$ rpm -qp --provides kernel-core-6.12.0-124.8.1.el10_1.x86_64.rpm | grep drm_
↪modeset
kernel(drm_modeset_acquire_fini) = 0x3d1dada1
kernel(drm_modeset_acquire_init) = 0x091eaa38
kernel(drm_modeset_backoff) = 0x9a62789e
kernel(drm_modeset_drop_locks) = 0xed88ec0a
kernel(drm_modeset_lock) = 0x1511267f
kernel(drm_modeset_lock_all) = 0xd3e9c304
kernel(drm_modeset_lock_all_ctx) = 0xfb8a89da
kernel(drm_modeset_lock_init) = 0x263f5b58
kernel(drm_modeset_lock_single_interruptible) = 0xb6428d3d
kernel(drm_modeset_unlock) = 0xebe4dddc
kernel(drm_modeset_unlock_all) = 0x79fa2207
```

The RHEL kABI stable list is guaranteed to remain constant, with modifications documented if a change is really required due to kernel backports. This way, if a kernel module compiled during the lifetime of the distribution is using symbols from the stable ABI, are guaranteed to remain compatible, installable and usable for the lifetime of the distribution.

Due to this, there is no need for any particular enforcing mechanism for matching pre-built modules to kernels. The result is that kernel module packages can have a different cadence than the kernel and still be usable for a long time even with different kernels.

Chapter 8. Azure Linux

This section covers:

- Azure Linux 3

8.1. 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-$(uname -r) \
    kernel-drivers-gpu-$(uname -r)
```

3. Enable the extended repository:

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

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

8.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

8.3. Network Repository Installation

1. Enable the network repository. For x86_64:

```
# wget https://developer.download.nvidia.com/compute/cuda/repos/
↪<distribution>/<arch>/cuda-<distribution>.repo -O /etc/yum.repos.d/cuda-
↪<distribution>.repo
```

   For sbsa:

```
# wget https://developer.download.nvidia.com/compute/cuda/repos/
↪<distribution>/sbsa/cuda-<distribution>.repo -O /etc/yum.repos.d/cuda-
↪<distribution>.repo
```

2. Clean DNF repository cache:

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

8.4. Driver Installation

These instructions apply to both local and network installations.

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

8.5. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

8.6. 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.

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

```
# tdnf update
```

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

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

This will remove any package which would have dependencies removed.

Chapter 9. Debian

This section covers:

- Debian 12
- Debian 13

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:

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

3. Enable the contrib repository:

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

4. Choose an installation method: *Local Repository Enablement* or *Network Repository Enablement (amd64)*.

9.2. Local Repository Enablement

1. Download the NVIDIA driver repository:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-
↪<distribution>-<driver-version>_1.0-1_<arch>.deb
```

   where `<driver-version>` is the NVIDIA driver version.

2. Install local repository on file system:

```
# dpkg -i nvidia-driver-local-repo-<distribution>-<driver-version>_1.0-1_
↪<arch>.deb
# apt update
```

3. Enroll ephemeral public GPG key:

```
# cp /var/nvidia-driver-local-repo-<distribution>-<driver-version>/nvidia-
↪driver-*-keyring.gpg /usr/share/keyrings/
```

9.3. Network Repository Enablement (amd64)

1. Install the new cuda-keyring package:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/
↪<distribution>/x86_64/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/
↪<distribution>/x86_64/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/<distribution>/
↪x86_64/ /" \
    | tee /etc/apt/sources.list.d/cuda-<distribution>-<arch>.list
```

9.4. Network Repository Enablement (arm64)

1. Install the new cuda-keyring package:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/
↪<distribution>/sbsa/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/
↪<distribution>/sbsa/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/<distribution>/
 ↪sbsa/ /" \
    | tee /etc/apt/sources.list.d/cuda-<distribution>-<arch>.list
```

9.5. Selecting a Branch or a Specific Driver version

To confine and pin the system to track a specific branch or driver version, install the appropriate pinning package, for example:

```
# apt install nvidia-driver-pinning-<branch>
```

or

```
# apt install nvidia-driver-pinning-<version>
```

Hint

For the best experience when locking a branch, we suggest installing the pinning package prior to installing the driver.

More information on the use and scope of these packages can be found in the recent updates section on *Version locking packages for Ubuntu / Debian* and explained in detail at *Version locking*.

9.6. 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
```

9.7. 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.

9.7.1. 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
```

9.7.2. 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
```

9.8. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

9.9. 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.

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:

```
# apt dist-upgrade
```

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

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

This will remove any package which would have dependencies removed.

Chapter 10. Fedora

This section covers:

- Fedora 44

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

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

10.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

For aarch64:

```
# dnf config-manager addrepo --from-repofile=https://developer.download.
↪nvidia.com/compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.
↪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*.

10.7. 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.

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.

Chapter 11. KylinOS

This section covers:

- KylinOS 11

11.1. 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*.

11.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

11.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

For aarch64:

```
# dnf config-manager --add-repo https://developer.download.nvidia.com/
↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
```

2. Clean DNF repository cache:

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

11.4. 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
```

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

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

Proprietary Kernel Modules

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

11.5.2. 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
```

11.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

11.7. Package Upgrades

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the *DNF 4* section of the *Version locking* chapter 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 kernel modules to open kernel modules:

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

This will remove any package which would have dependencies removed.

Chapter 12. Oracle Linux

This section covers:

- Oracle Linux 8
- Oracle Linux 9
- Oracle Linux 10

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:
   - **Oracle Linux 9/10:**

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

     If using the *64k* kernel variant on *aarch64*:

     ```
     # dnf install kernel-64k-devel-matched kernel-headers
     ```

   - **Oracle Linux 8:**

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

   - **Oracle Linux 10 using the UEK kernel variant:**

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

   - **Oracle Linux 8/9 using the UEK kernel variant:**

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

The UEK kernel on these distributions is compiled with a different compiler than the default system one; so the development package of the kernel will pull in the correct GCC toolset as a dependency. Once the package are installed, DKMS must be instructed to build all the modules that need to be used with the UEK kernel with that specific toolset.

The system compiler:

```
$ gcc --version | grep GCC
gcc (GCC) 11.5.0 20240719 (Red Hat 11.5.0-5.0.1)
```

With the UEK 8 kernel, we can clearly see that the kernel package is built with a different compiler:

```
$ cat /usr/src/kernels/6.12.0-100.28.2.2.el9uek.x86_64/.config | grep
 ↪CONFIG_CC_VERSION_TEXT
CONFIG_CC_VERSION_TEXT="gcc (GCC) 14.2.1 20250110 (Red Hat 14.2.1-7)"
```

This compiler belongs to the GCC toolset 14 Software Collection that was just installed as a dependency:

```
$ . /opt/rh/gcc-toolset-14/enable
$ gcc --version | grep GCC
gcc (GCC) 14.2.1 20250110 (Red Hat 14.2.1-7)
```

DKMS must be configured to use the same compiler to build the modules. From DKMS 3.3.0 there is support for configuring the environment for all modules:

```
# echo build_environment=/opt/rh/gcc-toolset-14/enable > /etc/dkms/
 ↪framework.conf.d/uek.conf
```

After this, the modules can be built like any other DKMS module.

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.

   - **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*.

12.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

12.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

   For `aarch64`:

   ```
   # dnf config-manager --add-repo https://developer.download.nvidia.com/
   ↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
   ```

2. Clean DNF repository cache:

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

12.4. 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.

Open Kernel Modules

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

- Example DKMS streams: `580-open` or `open-dkms`

Proprietary Kernel Modules

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

- Example DKMS streams: `580-dkms` or `latest-dkms`
- Example precompiled streams: `580` or `latest`

12.5. 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
```

12.6. 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.

12.6.1. 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
```

12.6.2. 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
```

12.7. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

12.8. 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.

12.8.1. Oracle Linux 8/9

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 kernel modules to open kernel modules:

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

12.8.2. Oracle Linux 10

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the *DNF 4* section of the *Version locking* chapter 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 kernel modules to open kernel modules:

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

This will remove any package which would have dependencies removed.

Chapter 13. Red Hat Enterprise Linux

This section covers:

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

13.1. Preparation

1. Perform the *Pre-installation Actions*.
2. Precompiled streams **do not** require kernel headers or DKMS. For information on precompiled streams, plese refer to the section *Precompiled Kernel Modules*.

   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 9/10:**

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

     If using the *64k* kernel variant on *aarch64*:

     ```
     # dnf install kernel-64k-devel-matched kernel-headers
     ```

   - **Red Hat Enterprise Linux 8:**

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

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 10**

```
# subscription-manager repos --enable=rhel-10-for-<arch>-appstream-
 ↪rpms
# subscription-manager repos --enable=rhel-10-for-<arch>-baseos-rpms
# subscription-manager repos --enable=codeready-builder-for-rhel-10-
 ↪<arch>-rpms
# dnf install https://dl.fedoraproject.org/pub/epel/epel-release-
 ↪latest-10.noarch.rpm
```

- **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
```

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

13.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
 ↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
 ↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

13.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

For `aarch64`:

```
# dnf config-manager --add-repo https://developer.download.nvidia.com/
 ↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
```

2. Clean DNF repository cache:

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

13.4. 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:

- Red Hat Enterprise Linux 8
- Red Hat Enterprise Linux 9

Open Kernel Modules

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

- Example DKMS streams: `580-open` or `open-dkms`

Proprietary Kernel Modules

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

- Example DKMS streams: `580-dkms` or `latest-dkms`
- Example precompiled streams: `580` or `latest`

For information on precompiled streams, plese refer to the section *Precompiled Kernel Modules*.

13.5. 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
```

13.6. 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.

13.6.1. 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
```

13.6.2. 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
```

13.7. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

13.8. 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.

13.8.1. Red Hat Enterprise Linux 8/9

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 kernel modules to open kernel modules:

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

13.8.2. Red Hat Enterprise Linux 10

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the *DNF 4* section of the *Version locking* chapter 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 kernel modules to open kernel modules:

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

This will remove any package which would have dependencies removed.

13.9. Precompiled Kernel Modules

13.9.1. Open kernel modules

This section applies to:

- Red Hat Enterprise Linux 10

Precompiled streams of open source modules offer an optional method of streamlining the installation process using Red Hat signed kernel modules. The advantages of precompiled streams:

- UEFI Secure Boot works out of the box via trusted keys already installed in the kernel keyring by Red Hat.
- Simpler dependencies: removes *kernel-devel*, *kernel-headers*, DKMS and GCC compiler requirements.
- Fewer repositories: the EPEL repository is no longer needed for DKMS or for the extra EGL libraries.

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.
```

(continues on next page)

(continued from previous page)

```
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
```

To use the precompiled kernel module packages provided by Red Hat (examples for x86_64):

1. Enable the *Extensions* repository:

```
# subscription-manager repos --enable rhel-10-for-x86_64-extensions-rpms
Repository 'rhel-10-for-x86_64-extensions-rpms' is enabled for this
 ↪system.
```

2. Install the preferred driver combination, selecting the **precompiled kernel module packages** as part of the transaction:

```
# dnf install kmod-nvidia-open nvidia-driver nvidia-driver-cuda
Updating Subscription Management repositories.
Dependencies resolved.
================================================================================
 Package                                          Architecture Version
 ↪                  Repository                                Size
================================================================================
Installing:
 kmod-nvidia-open-590.48.01-6.12.0-124.31.1       x86_64       3:590.48.
 ↪01-3.el10_1          rhel-10-for-x86_64-extensions-rpms        22 M
 nvidia-driver                                    x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        4.7 M
 nvidia-driver-cuda                               x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        534 k
Installing dependencies:
 egl-gbm                                          x86_64       2:1.1.2.1-
 ↪1.el10_1           rhel-10-for-x86_64-extensions-rpms         21 k
 egl-wayland                                      x86_64       1.1.20-1.
 ↪el10_1              rhel-10-for-x86_64-extensions-rpms        45 k
 egl-x11                                          x86_64       1.0.3-1.
 ↪el10_1               rhel-10-for-x86_64-extensions-rpms       55 k
 libnvidia-cfg                                    x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        151 k
 libnvidia-gpucomp                                x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        24 M
 libnvidia-ml                                     x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        676 k
 libvdpau                                         x86_64       1.5-8.el10
 ↪                   rhel-10-for-x86_64-appstream-rpms          20 k
 nvidia-driver-cuda-libs                          x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        85 M
 nvidia-driver-libs                               x86_64       3:590.48.
 ↪01-1.el10            cuda-rhel10-x86_64                        104 M
 nvidia-kmod-common                               noarch       3:590.48.
```

(continues on next page)

(continued from previous page)

```
↪01-1.el10            cuda-rhel10-x86_64                              72 M
 nvidia-modprobe                                    x86_64       3:590.48.
↪01-1.el10            cuda-rhel10-x86_64                              30 k
 nvidia-persistenced                                x86_64       3:590.48.
↪01-1.el10            cuda-rhel10-x86_64                              34 k
 ocl-icd                                            x86_64       2.3.2-8.
↪el10                  rhel-10-for-x86_64-baseos-rpms                 69 k
 opencl-filesystem                                  noarch       1.0-22.
↪el10                   rhel-10-for-x86_64-appstream-rpms             9.0 k
Installing weak dependencies:
 dnf-plugin-nvidia                                  noarch       2.3-1.el10
↪                   cuda-rhel10-x86_64                              17 k

Transaction Summary
================================================================================
Install  18 Packages

Total download size: 313 M
Installed size: 1.0 G
Is this ok [y/N]:
```

13.9.2. Proprietary kernel modules

This section applies to:

- Red Hat Enterprise Linux 8 on x86_64
- Red Hat Enterprise Linux 9 on x86_64

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

- UEFI Secure Boot can be enabled by importing trusted keys from NVIDIA.
- Simpler dependencies: removes *kernel-devel*, *kernel-headers*, DKMS and GCC compiler requirements.
- Fewer repositories: the EPEL repository is no longer needed for DKMS.

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
```

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 precompiled kernel module packages:

1. 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>
     ```

2. 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.

13.9.2.1 Importing the MOK key for Secure Boot

If running on an UEFI system with Secure Boot enabled, import the Machine Owner Key in the firmware.

First of all, download the appropriate key:

- **Red Hat Enterprise Linux 9**

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/rhel9/x86_64/
↪NVIDIA2019-public_key.der
```

- **Red Hat Enterprise Linux 8**

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/
↪NVIDIA2019-public_key.der
```

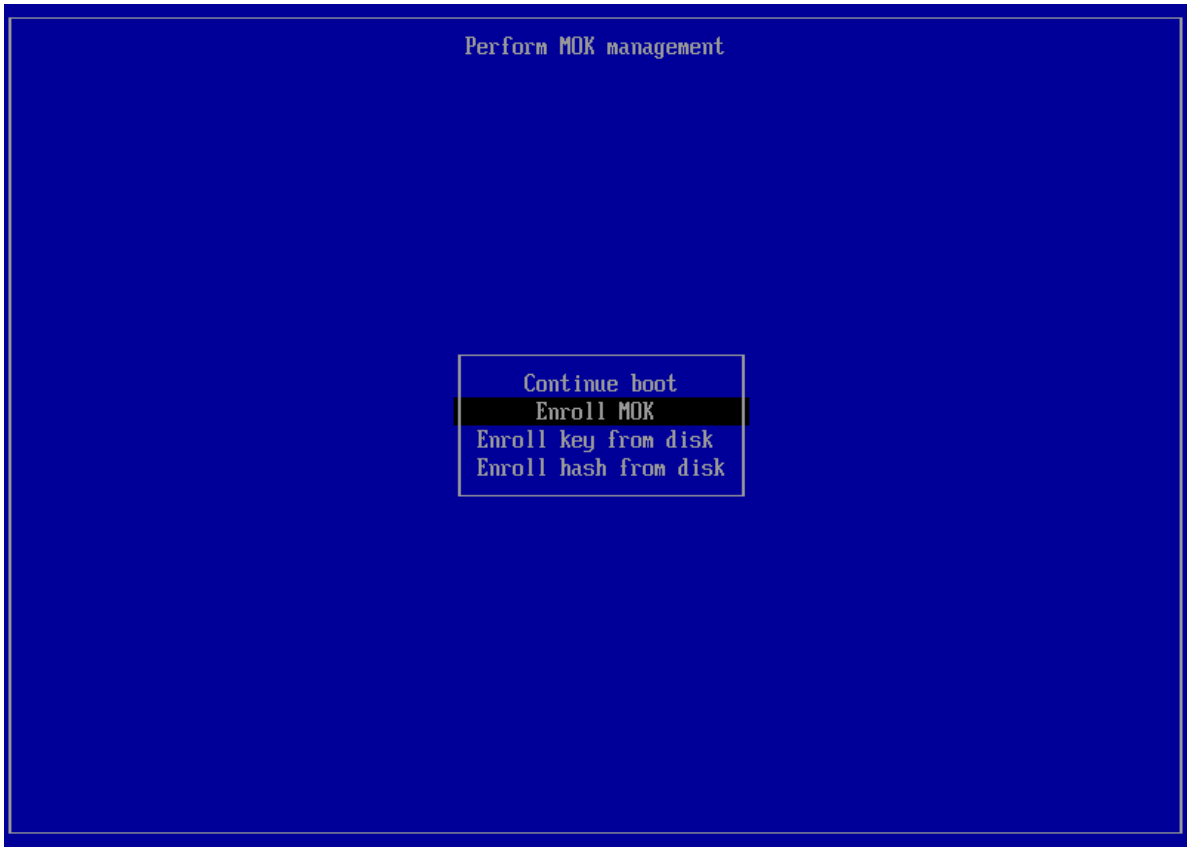
Then import the key into the firmware, selecting a temporary password:

```
# mokutil --import *-public_key.der
input password: <hidden>
input password again: <hidden>
# mokutil --list-new --short
d56f7e6ace NVIDIA
# systemctl reboot
```

During the reboot you'll be presented with the `mokutil` tool:

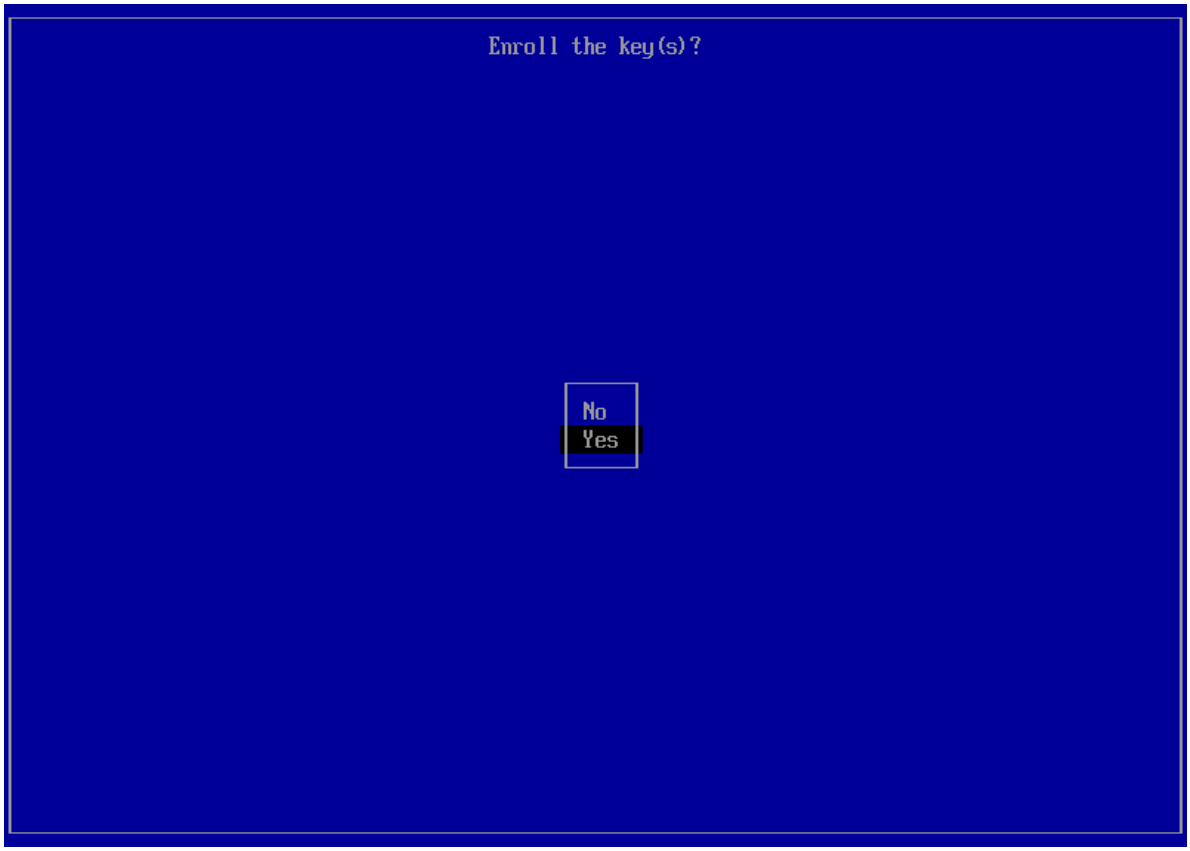
1. Press any key to begin.

2. Select *Enroll MOK*:



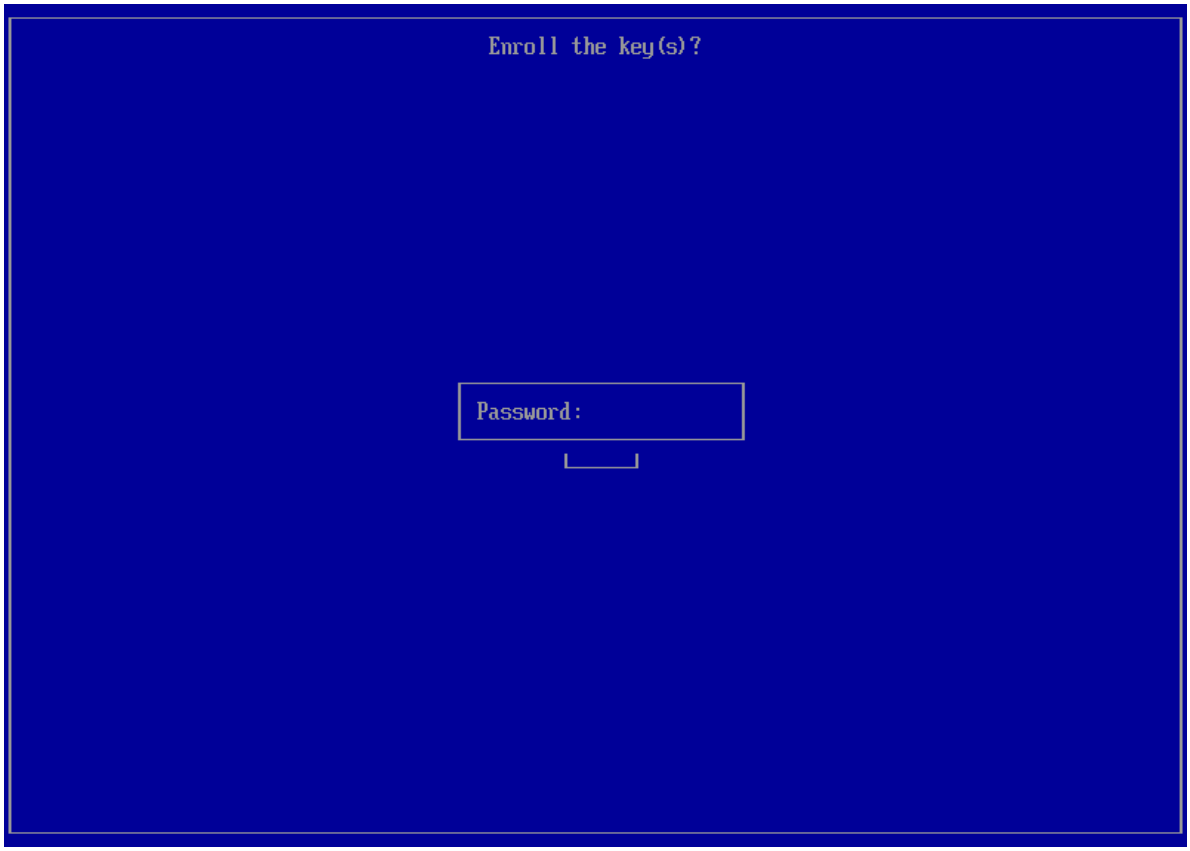
3. Select *Continue* to proceed:

4. Select *Yes* to enroll the key.

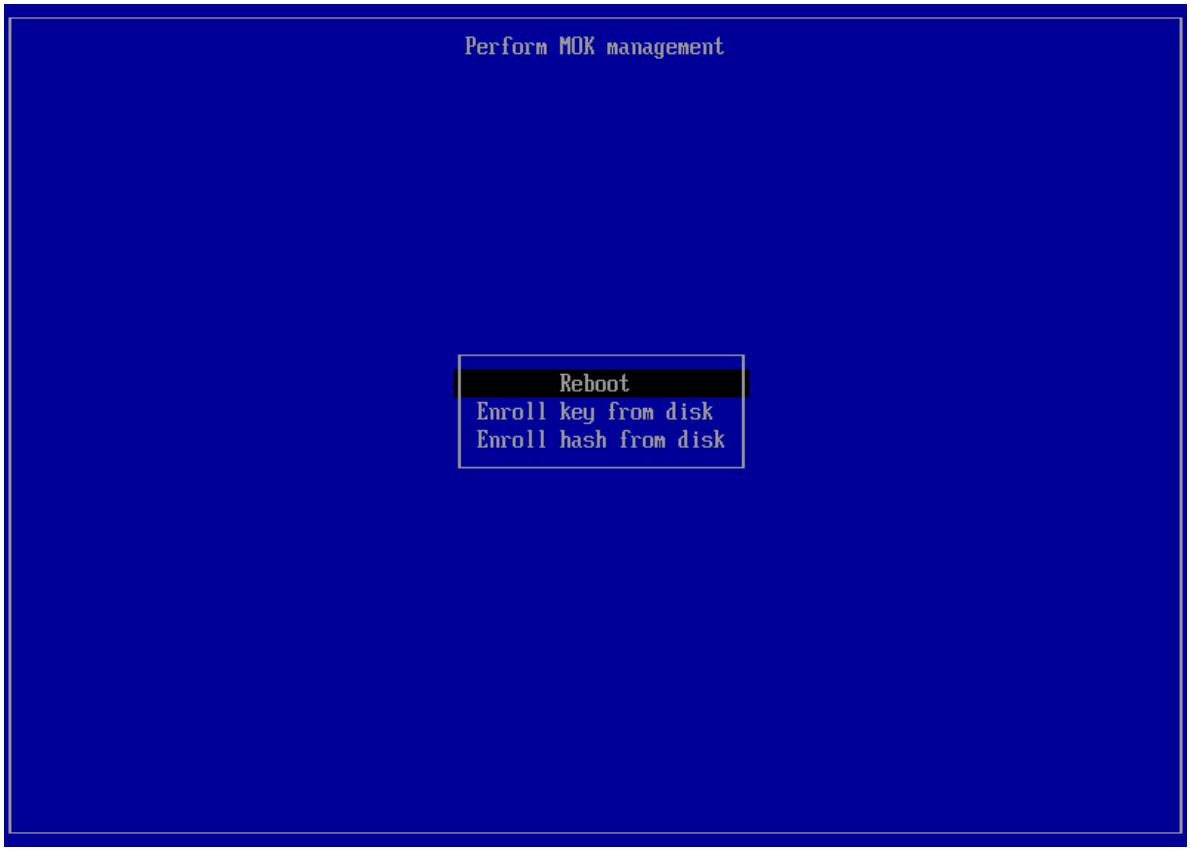


5. Type the MOK password you created for the key during installation.

> **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.



13.9.3. 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 590.x	590	590-dkms	590-open
Locked at 580.x	580	580-dkms	580-open
Locked at 570.x	570	570-dkms	570-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>
```

Chapter 14. Rocky Linux

This section covers:

- Rocky Linux 8
- Rocky Linux 9
- Rocky Linux 10

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:
   - **Rocky Linux 9/10:**

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

     If using the *64k* kernel variant on *aarch64*:

     ```
     # dnf install kernel-64k-devel-matched kernel-headers
     ```

   - **Rocky Linux 8:**

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

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.

   - **Rocky Linux 9/10**

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

- **Rocky Linux 8**

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

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

14.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# dnf install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

14.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

For `aarch64`:

```
# dnf config-manager --add-repo https://developer.download.nvidia.com/
↪compute/cuda/repos/<distribution>/sbsa/cuda-<distribution>.repo
```

2. Clean DNF repository cache:

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

14.4. 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:

- Rocky Linux 8
- Rocky Linux 9

Open Kernel Modules

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

- Example DKMS streams: `580-open` or `open-dkms`

Proprietary Kernel Modules

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

- Example DKMS streams: `580-dkms` or `latest-dkms`
- Example precompiled streams: `580` or `latest`

14.5. 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
```

14.6. 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.

14.6.1. 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
```

14.6.2. 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
```

14.7. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

14.8. 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.

14.8.1. Rocky Linux 8/9

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 kernel modules to open kernel modules:

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

14.8.2. Rocky Linux 10

If DNF locking is configured on the sytem, please adjust the configuration or remove the lock entirely. Please refer to the *DNF 4* section of the *Version locking* chapter 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 kernel modules to open kernel modules:

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

This will remove any package which would have dependencies removed.

Chapter 15. SUSE

This section covers:

- SUSE Linux Enterprise 15
- SUSE Linux Enterprise 16
- OpenSUSE Leap 15
- OpenSUSE Leap 16

15.1. 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*.

15.2. Local Repository Installation

Install the NVIDIA driver repository:

```
# zypper install https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-<distribution>-
↪<driver-version>-1.0-1.<arch>.rpm
```

where `<driver-version>` is the NVIDIA driver version

15.3. Network Repository Installation

1. Enable the network repository. For `x86_64`:

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

   For `aarch64`:

   ```
   # zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   ↪<distribution>/sbsa/cuda-<distribution>.repo
   ```

2. Refresh Zypper repository cache:

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

15.4. 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
```

15.5. 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.

15.5.1. Compute-only System

Open Kernel Modules

```
# zypper -v install nvidia-compute-G07 nvidia-open-driver-G07
```

Proprietary Kernel Modules

```
# zypper -v install nvidia-compute-G07 nvidia-driver-G07
```

15.5.2. Desktop-only System

Open Kernel Modules

```
# zypper -v install nvidia-video-G07 nvidia-open-driver-G07
```

Proprietary Kernel Modules

```
# zypper -v install nvidia-video-G07 nvidia-driver-G07
```

15.6. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

15.7. 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.

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
```

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

Chapter 16. Ubuntu

This section covers:

- Ubuntu 22.04
- Ubuntu 24.04
- Ubuntu 26.04

16.1. 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 Enablement* or *Network Repository Enablement (amd64)*.

16.2. Local Repository Enablement

1. Download the NVIDIA driver repository:

```
$ wget https://developer.download.nvidia.com/compute/nvidia-driver/
↪<driver-version>/local_installers/nvidia-driver-local-repo-
↪<distribution>-<driver-version>_1.0-1_<arch>.deb
```

   where `<driver-version>` is the NVIDIA driver version.

2. Install local repository on file system:

```
# dpkg -i nvidia-driver-local-repo-<distribution>-<driver-version>_1.0-1_
 ↪<arch>.deb
# apt update
```

3. Enroll ephemeral public GPG key:

```
# cp /var/nvidia-driver-local-repo-<distribution>-<driver-version>/nvidia-
 ↪driver-*-keyring.gpg /usr/share/keyrings/
```

16.3. Network Repository Enablement (amd64)

Install the new cuda-keyring package.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/<distribution>
 ↪/x86_64/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/
 ↪<distribution>/x86_64/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/<distribution>/
 ↪x86_64/ /" \
    | tee /etc/apt/sources.list.d/cuda-<distribution>-<arch>.list
```

16.4. Network Repository Enablement (arm64)

Install the new cuda-keyring package.

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/<distribution>
 ↪/sbsa/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/
 ↪<distribution>/sbsa/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/<distribution>/
 ↪sbsa/ /" \
    | tee /etc/apt/sources.list.d/cuda-<distribution>-<arch>.list
```

16.5. Selecting a Branch or a Specific Driver version

To confine and pin the system to track a specific branch or driver version, install the appropriate pinning package, for example:

```
# apt install nvidia-driver-pinning-<branch>
```

or

```
# apt install nvidia-driver-pinning-<version>
```

Hint

For the best experience when locking a branch, we suggest installing the pinning package prior to installing the driver.

More information on the use and scope of these packages can be found in the recent updates section on *Version locking packages for Ubuntu / Debian* and explained in detail at *Version locking*.

Warning

Starting with branch 590, the Ubuntu packages have been renamed by removing the branch designation from the package name. Switching branches, installing specific driver versions and specific upgrade / downgrade requirements will be supported through these version locking (pinning) packages.

Please refer to the *Ubuntu 590 and later packages* section of the recent updates for more information.

16.6. 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
```

16.7. 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.

16.7.1. Compute-only System

Open Kernel Modules

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

Proprietary Kernel Modules

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

16.7.2. Desktop-only System

Open Kernel Modules

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

Proprietary Kernel Modules

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

16.8. Reboot the System

```
# reboot
```

Perform the *Post-installation Actions*.

16.9. 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.

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:

```
# apt dist-upgrade
```

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

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

This will remove any package which would have dependencies removed.

Chapter 17. Windows

The NVIDIA Display Driver for Windows can be installed or uninstalled silently using command-line options.

17.1. Silent Installation

To install the display driver silently, use the following command:

```
setup.exe -s -n Display.Driver
```

This suppresses the installer UI and installs only the display driver.

17.2. Enabling Logs

To enable logging during installation, add the following options:

```
setup.exe -s -n Display.Driver -log:c:\logs -loglevel:6
```

This command saves installation logs to the specified directory (`c:\logs`) with a log level of 6.

17.3. Silent Uninstallation

To silently uninstall the display driver, run:

```
"C:\Windows\SysWOW64\RunDll32.EXE" "C:\Program Files\NVIDIA Corporation\
↪Installer2\InstallerCore\NVI2.DLL",UninstallPackage Display.Driver -silent -
↪n
```

The `-n` option suppresses the reboot prompt.

17.4. Exit Codes

After installation or uninstallation, check the exit code:

- `0`: Success
- 1: Success, but reboot required

- Any other value: Failure

17.4.1. Exit Code on Silent Uninstallation

To capture the correct exit code when running silent uninstallation, use the following PowerShell command instead of invoking `RunDll32` directly:

```
$process = Start-Process -FilePath "C:\Windows\SysWOW64\RunDll32.EXE" `
  -ArgumentList '"C:\Program Files\NVIDIA Corporation\Installer2\
↪InstallerCore\NVI2.DLL",UninstallPackage Display.Driver', '-silent', '-n' `
  -Wait -PassThru

Write-Output "Exit Code: $($process.ExitCode)"
```

This method ensures the uninstall process runs to completion and returns an accurate exit status.

Chapter 18. Driver Helper Script

A 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 8: 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]`	Linux distribution using the `DISTRO:VERSION` or `DISTRO` pattern.
`--module-flavor [MODULE_FLAVOR]`	Kernel module flavor; `open` and `closed` are accepted values.
`--verbose`	[OPTIONAL] Verbose output.

The following are example command outputs:

Table 9: Command outputs

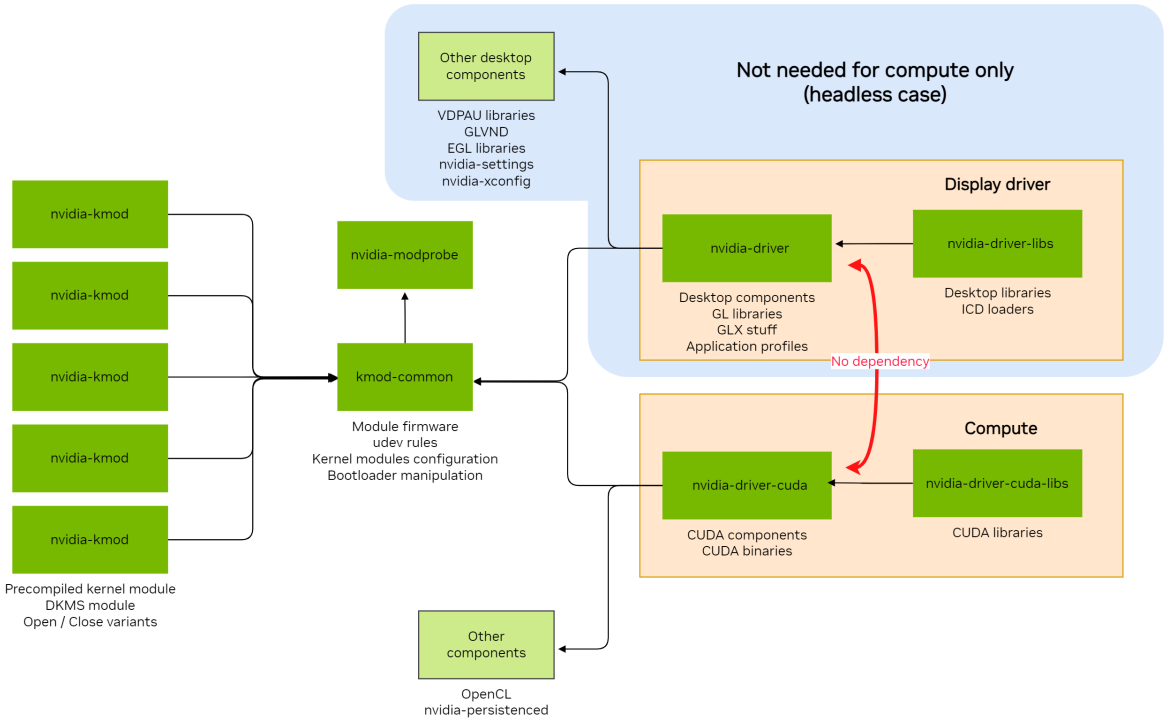
Command	Example Output
`$ nvidia-driver-assistant`	**Detected GPUs:** NVIDIA GeForce RTX 3070 - (pci_id 0x2484) **Detected system:** Ubuntu 24.04 **Please copy and paste the following command to install the open kernel module flavour:** sudo apt install -y nvidia-open
`$ nvidia-driver-assistant --install`	**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

Chapter 19. 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 configurations:

- 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:

- Amazon Linux 2023
- Red Hat Enterprise Linux 8 / AlmaLinux 8 / Rocky Linux 8 / Oracle Linux 8
- Red Hat Enterprise Linux 9 / AlmaLinux 9 / Rocky Linux 9 / Oracle Linux 9
- Red Hat Enterprise Linux 10 / AlmaLinux 10 / Rocky Linux 10 / Oracle Linux 10
- KylinOS 11
- Fedora 44
- OpenSUSE Leap 15
- OpenSUSE Leap 16
- SUSE Linux Enterprise Server 15
- SUSE Linux Enterprise Server 16
- Debian 12
- Debian 13
- Ubuntu 22.04
- Ubuntu 24.04
- Ubuntu 26.04

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

- Azure Linux 3

More information is available in the respective sections.

Chapter 20. Wayland-only Desktop Installation

Most of the current supported distributions use Wayland as the default display server; so it's possible to have a Wayland-only installation with the NVIDIA driver packages.

Red Hat Enterprise Linux 8/9, AlmaLinux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, Fedora 44

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 by the NVIDIA X driver package. So by removing the X.org package, you're removing everything that is not required to run an X.org session.

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

Red Hat Enterprise Linux 10, AlmaLinux 10, Rocky Linux 10, Oracle Linux 10

The distribution already ships without any X.org component, only Wayland is available.

Ubuntu 22.04/24.04/26.04

Since no reverse dependency is possible, the X.org component for the NVIDIA driver must be explicitly installed in Ubuntu.

```
# apt install xserver-xorg-video-nvidia
```

Chapter 21. 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 615. Make sure every package that you want to lock onto a specific branch has the appropriate line / code block in the configuration files.

21.1. DNF 4

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

- Red Hat Enterprise Linux 8 / AlmaLinux 8 / Rocky Linux 8 / Oracle Linux 8
- Red Hat Enterprise Linux 9 / AlmaLinux 9 / Rocky Linux 9 / Oracle Linux 9
- Red Hat Enterprise Linux 10 / AlmaLinux 10 / Rocky Linux 10 / Oracle Linux 10
- Kylin 11
- 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.*
```

(continues on next page)

(continued from previous page)

```
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.*
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.

21.2. DNF 5

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

- Fedora 44

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"
```

(continues on next page)

(continued from previous page)

```
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 = "<"
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 = ">="
```

(continues on next page)

(continued from previous page)

```
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"

[[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.

21.3. APT

APT is the package manager of the following distributions:

- Debian 12
- Debian 13
- Ubuntu 22.04
- Ubuntu 24.04

- Ubuntu 26.04

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.

Ubuntu:

```
# Basic driver packages, includes foreign architectures
Package: src:nvidia-graphics-drivers-580:any src:nvidia-kmod-open:any
Pin: version 580*
Pin-Priority: 1000

# Basic driver packages, only in the native architectures
Package: src:nvidia-modprobe src:nvidia-modprobe src:nvidia-persistenced
 ↪src:nvidia-settings src:nvidia-xconfig
Pin: version 580*
Pin-Priority: 1000

# Meta packages, includes foreign architectures
Package: src:cuda-drivers:any src:nvidia-open:any
Pin: version 580*
Pin-Priority: 1000

# Extra driver packages, only in the native architectures
Package: src:libnvidia-nscq src:libnvsdm src:nvidia-fabricmanager src:nvidia-
 ↪imex src:nvlink5 src:fabricmanager src:imex
Pin: version 580*
Pin-Priority: 1000
```

Debian:

```
# Basic driver packages, includes foreign architectures
Package: src:nvidia-graphics-drivers:any src:nvidia-kmod-open:any
Pin: version 580*
Pin-Priority: 1000

# Basic driver packages, only in the native architectures
Package: src:nvidia-modprobe src:nvidia-modprobe src:nvidia-persistenced
 ↪src:nvidia-settings src:nvidia-xconfig
Pin: version 580*
Pin-Priority: 1000

# Meta packages, includes foreign architectures
Package: src:cuda-drivers:any src:nvidia-open:any
Pin: version 580*
Pin-Priority: 1000

# Extra driver packages, only in the native architectures
Package: src:libnvidia-nscq src:libnvsdm src:nvidia-fabricmanager src:nvidia-
 ↪imex src:nvlink5 src:fabricmanager src:imex
Pin: version 580*
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.

21.3.1. Pinning packages

To facilitate locking driver packages to branches or to pin specific driver version, packages that contain precompiled pinning files are available. These version locking packages have a few benefits:

- All packages are aligned to the same driver branch, including ones that were never consistent (`nvidia-modprobe`. etc.).
- All packages can have proper versions even if we must declare dependencies as equal or higher (ex. `nvidia-fabricmanager`, etc.) due to APT limitations.
- Users can also downgrade easily, both to versions and specific driver branches, which is not something that can be really achieved for Debian.
- Ubuntu specific benefits:
  - From version 590 onwards, all packages are renamed, simplified and made consistent for all branches. This means consistent documentation and **consistent instructions for every new branch**.
  - Extra packages (for example from Canonical) are no longer pulled down and mixed in a transaction with NVIDIA packages when installing / upgrading.
  - Non DGX users (laptops, workstations, etc.) can just add the repository and get upgraded to the latest drivers without the need to reinstall the packages for every new branch.

Hint

For the best experience when locking a branch, we suggest to install the pinning package prior to installing the driver.

For example, the branch locking packages are called:

```
nvidia-driver-pinning-570
nvidia-driver-pinning-575
nvidia-driver-pinning-580
nvidia-driver-pinning-590
```

On top of that there are version specific pinning packages that are built as part of the driver, and that allow pinning a specific driver version. For example:

```
nvidia-driver-pinning-570.199
nvidia-driver-pinning-570.200
nvidia-driver-pinning-580.102.03
nvidia-driver-pinning-580.103
```

Note

APT flags every file under `/etc` as a configuration file, so the pinning file is marked as configuration and not overwritten / removed unless the user purges the configuration (`--purge`).

Each of these packages have the following conflicts set, so only one can be installed at a time:

```
Provides: nvidia-driver-pinning
Replaces: nvidia-driver-pinning
Conflicts: nvidia-driver-pinning
```

Switching the pinning from one branch/version to another is only a matter of installing a new one. For example:

```
# apt install -V --purge nvidia-driver-pinning-590
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
   nvidia-driver-570 (570-0ubuntu1)
The following NEW packages will be installed:
   nvidia-driver-590 (590-0ubuntu1)
0 upgraded, 1 newly installed, 0 downgraded, 1 to remove and 0 not upgraded.
Need to get 10 KB of archives.
After this operation, 10 KB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Due to the higher piority, **these packages can be used to install, downgrade or upgrade to specific versions**.

Warning

On Ubuntu, up and including to branch 580, the packages still have the branch reference in the package name. The upgrade/downgrade/install of driver packages that would easily switch branches is only supported on 590+.

From 590 driver releases onwards, Ubuntu packages will lose the branch definition in the package names (like Debian packages) to allow easy switching back and forth on branches and versions.

For example, to switch from the generic latest driver to a specific release on Debian:

```
# apt-get install nvidia-driver-pinning-570.199
[..]
# apt-get -V dist-upgrade --autoremove
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be REMOVED:
   libnvidia-present (580.103-1)
The following packages will be DOWNGRADED:
   firmware-nvidia-gsp (580.103-1 => 570.199-1)
   libcuda1 (580.103-1 => 570.199-1)
   libcuda1:i386 (580.103-1 => 570.199-1)
   libcudadebugger1 (580.103-1 => 570.199-1)
   libegl-nvidia0 (580.103-1 => 570.199-1)
   libegl-nvidia0:i386 (580.103-1 => 570.199-1)
   libgles-nvidia1 (580.103-1 => 570.199-1)
   libgles-nvidia1:i386 (580.103-1 => 570.199-1)
   libgles-nvidia2 (580.103-1 => 570.199-1)
```

(continues on next page)

(continued from previous page)

```
   libgles-nvidia2:i386 (580.103-1 => 570.199-1)
   libglx-nvidia0 (580.103-1 => 570.199-1)
   libglx-nvidia0:i386 (580.103-1 => 570.199-1)
   libnvcuvid1 (580.103-1 => 570.199-1)
   libnvidia-allocator1 (580.103-1 => 570.199-1)
   libnvidia-allocator1:i386 (580.103-1 => 570.199-1)
   libnvidia-api1 (580.103-1 => 570.199-1)
   libnvidia-cfg1 (580.103-1 => 570.199-1)
   libnvidia-eglcore (580.103-1 => 570.199-1)
   libnvidia-eglcore:i386 (580.103-1 => 570.199-1)
   libnvidia-encode1 (580.103-1 => 570.199-1)
   libnvidia-fbc1 (580.103-1 => 570.199-1)
   libnvidia-glcore (580.103-1 => 570.199-1)
   libnvidia-glcore:i386 (580.103-1 => 570.199-1)
   libnvidia-glvkspirv (580.103-1 => 570.199-1)
   libnvidia-glvkspirv:i386 (580.103-1 => 570.199-1)
   libnvidia-gpucomp (580.103-1 => 570.199-1)
   libnvidia-gpucomp:i386 (580.103-1 => 570.199-1)
   libnvidia-ml1 (580.103-1 => 570.199-1)
   libnvidia-ml1:i386 (580.103-1 => 570.199-1)
   libnvidia-ngx1 (580.103-1 => 570.199-1)
   libnvidia-nvvm4 (580.103-1 => 570.199-1)
   libnvidia-opticalflow1 (580.103-1 => 570.199-1)
   libnvidia-pkcs11-openssl3 (580.103-1 => 570.199-1)
   libnvidia-ptxjitcompiler1 (580.103-1 => 570.199-1)
   libnvidia-ptxjitcompiler1:i386 (580.103-1 => 570.199-1)
   libnvidia-rtcore (580.103-1 => 570.199-1)
   libnvidia-sandboxutils (580.103-1 => 570.199-1)
   libnvidia-vksc-core (580.103-1 => 570.199-1)
   libnvoptix1 (580.103-1 => 570.199-1)
   libxnvctrl0 (580.103-1 => 570.199-1)
   nvidia-driver (580.103-1 => 570.199-1)
   nvidia-driver-cuda (580.103-1 => 570.199-1)
   nvidia-driver-libs (580.103-1 => 570.199-1)
   nvidia-driver-libs:i386 (580.103-1 => 570.199-1)
   nvidia-egl-icd (580.103-1 => 570.199-1)
   nvidia-kernel-open-dkms (580.103-1 => 570.199-1)
   nvidia-kernel-support (580.103-1 => 570.199-1)
   nvidia-modprobe (580.103-1 => 570.199-1)
   nvidia-open (580.103-1 => 570.199-1)
   nvidia-opencl-icd (580.103-1 => 570.199-1)
   nvidia-persistenced (580.103-1 => 570.199-1)
   nvidia-settings (580.103-1 => 570.199-1)
   nvidia-vdpau-driver (580.103-1 => 570.199-1)
   nvidia-vulkan-icd (580.103-1 => 570.199-1)
   nvidia-vulkan-icd:i386 (580.103-1 => 570.199-1)
   nvidia-xconfig (580.103-1 => 570.199-1)
   xserver-xorg-video-nvidia (580.103-1 => 570.199-1)
0 upgraded, 0 newly installed, 57 downgraded, 1 to remove and 0 not upgraded.
Need to get 0 B/341 MB of archives.
After this operation, 171 MB disk space will be freed.
Do you want to continue? [Y/n]
```

21.4. Zypper

Zypper is the package manager of the following distributions:

- SUSE Enterprise Linux 15
- SUSE Enterprise Linux 16
- openSUSE Leap 15
- openSUSE Leap 16

Just run the `zypper addlock` command to populate the file `/etc/zypp/locks` (which can then be edited or left as is) to lock a specific driver version.

Note

The selection is negative, so the lock specifies the packages that needs to be *excluded*.

```
# zypper addlock "*nvidia* >= 580"
Specified lock has been successfully added.
# cat /etc/zypp/locks

type: package
version: >= 580
match_type: glob
case_sensitive: on
solvable_name: *nvidia*
```

This will make sure that anything above and including branch 580 is excluded. For example, with 580 already in the CUDA repository, we can see it only allows anything lower than 580:

```
# zypper search --details nvidia-compute-G06
Loading repository data...
Reading installed packages...

S  | Name                       | Type    | Version      | Arch   | Repository
---+----------------------------+---------+--------------+--------+-------------
 ↪----------
 l | nvidia-compute-G06         | package | 580.82.07-1  | x86_64 | cuda-
 ↪opensuse15-x86_64
 l | nvidia-compute-G06         | package | 580.65.06-1  | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 575.57.08-1  | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 575.51.03-1  | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 570.172.08-1 | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 570.158.01-1 | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 570.148.08-1 | x86_64 | cuda-
 ↪opensuse15-x86_64
   | nvidia-compute-G06         | package | 570.133.20-1 | x86_64 | cuda-
```

(continues on next page)

(continued from previous page)

```
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 570.124.06-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 570.86.15-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 570.86.10-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 565.57.01-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 560.35.05-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 560.35.03-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 560.28.03-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 555.42.06-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 555.42.02-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.163.01-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.144.03-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.127.08-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.127.05-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.90.12-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.90.07-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.54.15-0  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06       | package | 550.54.14-0  | x86_64 | cuda-
↪opensuse15-x86_64
l | nvidia-compute-G06-32bit | package | 580.82.07-1  | x86_64 | cuda-
↪opensuse15-x86_64
l | nvidia-compute-G06-32bit | package | 580.65.06-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 575.57.08-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 575.51.03-1  | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 570.172.08-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 570.158.01-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 570.148.08-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 570.133.20-1 | x86_64 | cuda-
↪opensuse15-x86_64
   | nvidia-compute-G06-32bit | package | 570.124.06-1 | x86_64 | cuda-
```

(continues on next page)

(continued from previous page)

```
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 570.86.15-1  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 570.86.10-1  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 565.57.01-1  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 560.35.05-1  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 560.35.03-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 560.28.03-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 555.42.06-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 555.42.02-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.163.01-1 | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.144.03-1 | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.127.08-1 | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.127.05-1 | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.90.12-1  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.90.07-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.54.15-0  | x86_64 | cuda-
↪opensuse15-x86_64
  | nvidia-compute-G06-32bit | package | 550.54.14-0  | x86_64 | cuda-
↪opensuse15-x86_64
```

Please refer to the `locks(5)` manual page for more information and configuration options.

Chapter 22. 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.

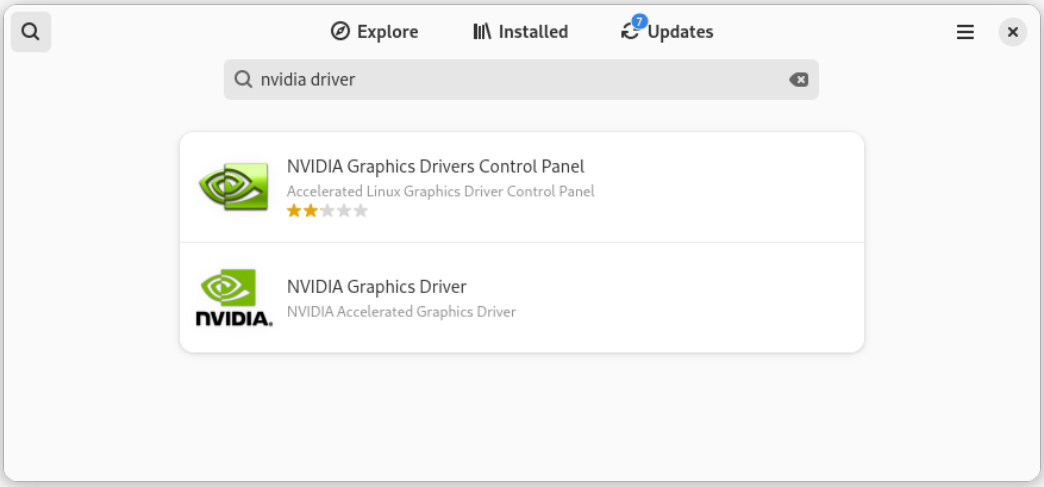
At the moment this process is enabled for the following distributions:

- Red Hat Enterprise Linux 10 / AlmaLinux 10 / Rocky Linux 10
- Fedora 44

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

22.1. Driver Installation

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

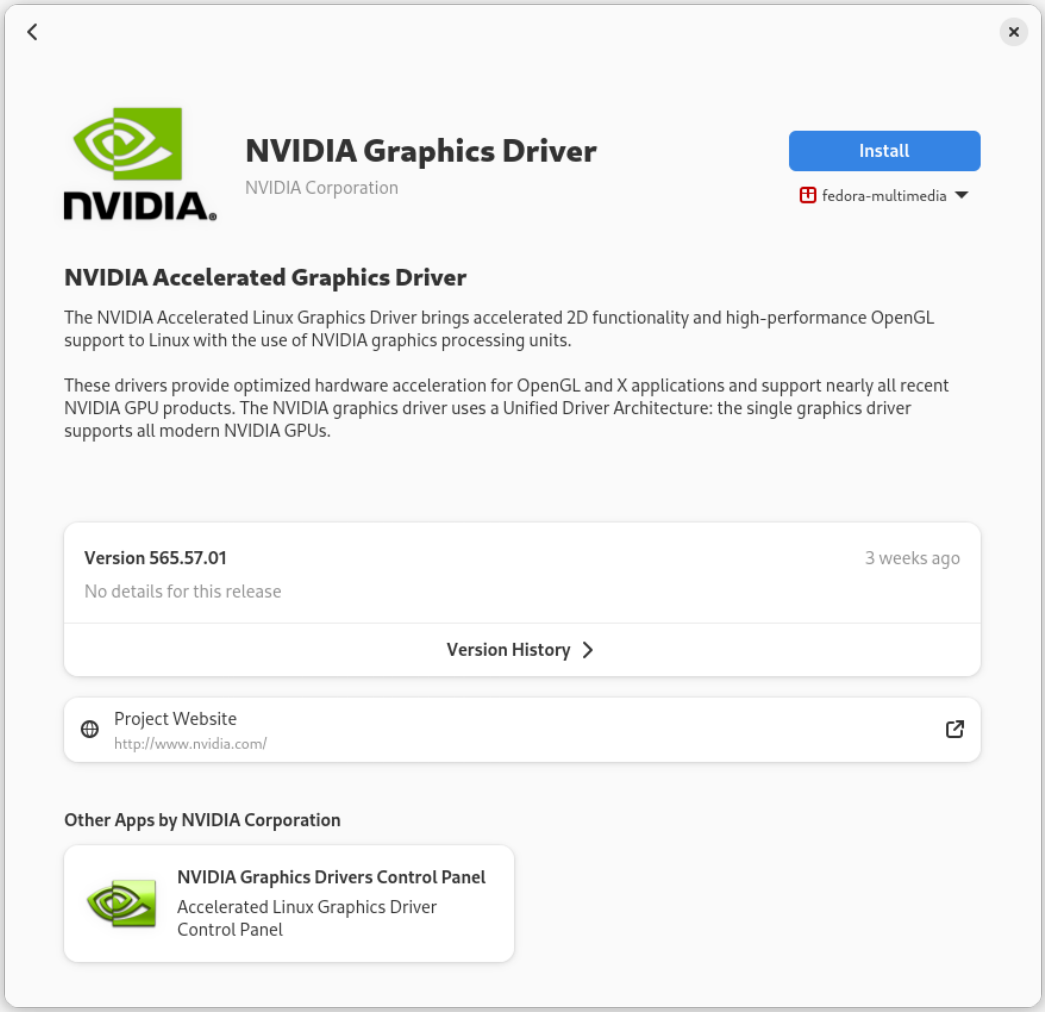


   If nothing is found, then run the following command to force a PackageKit metadata refresh out of its normal regular schedule:

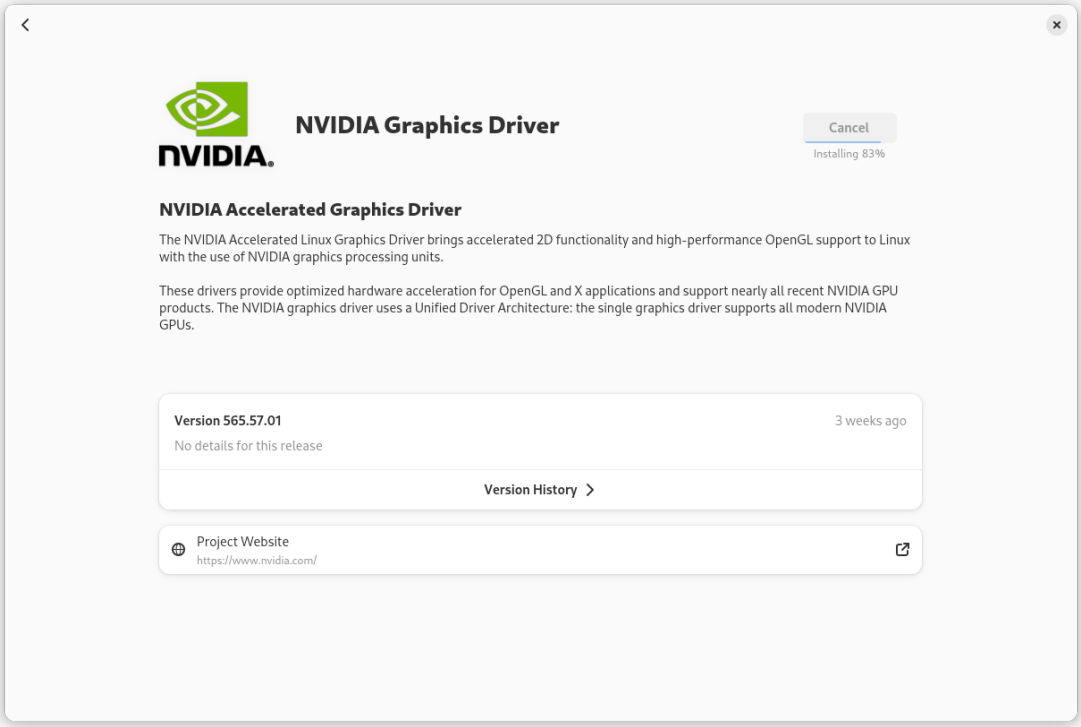
```
# pkcon refresh force
```

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.

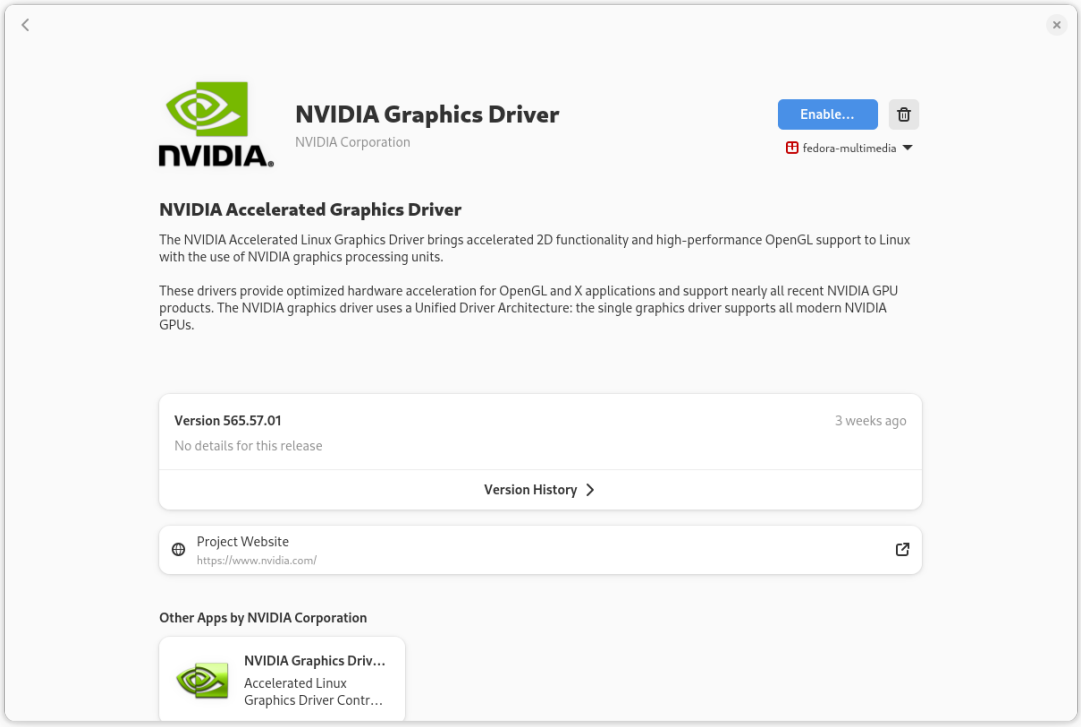


4. The installation will now begin. You can check the progress below the greyed out button.

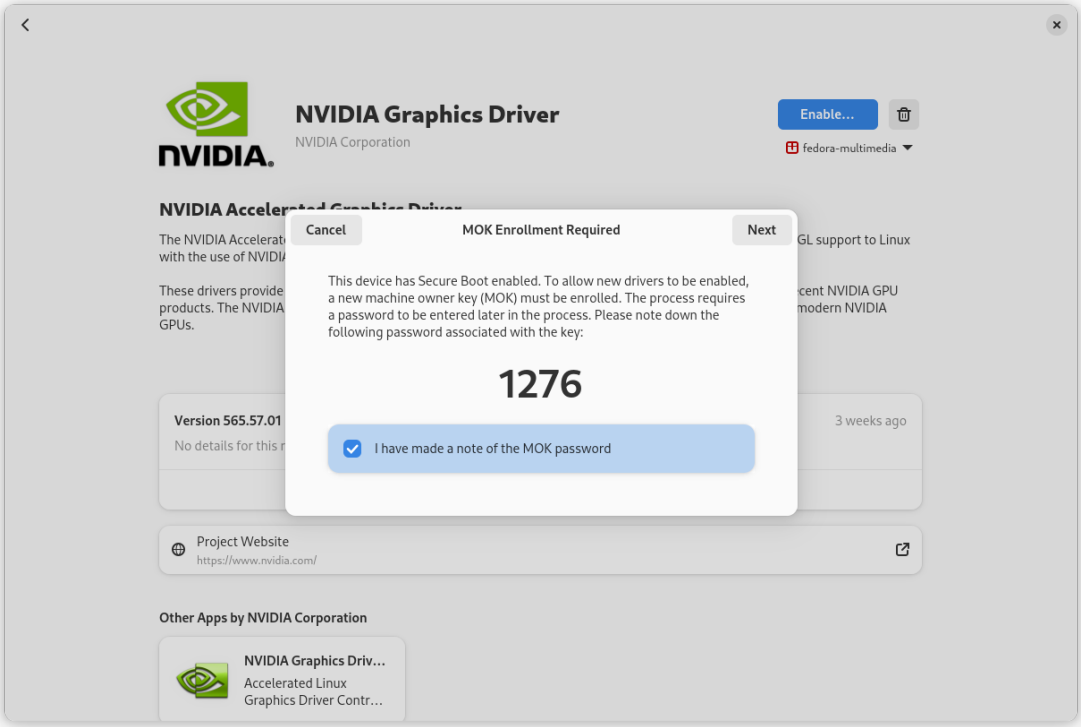


22.2. Secure Boot Preparation

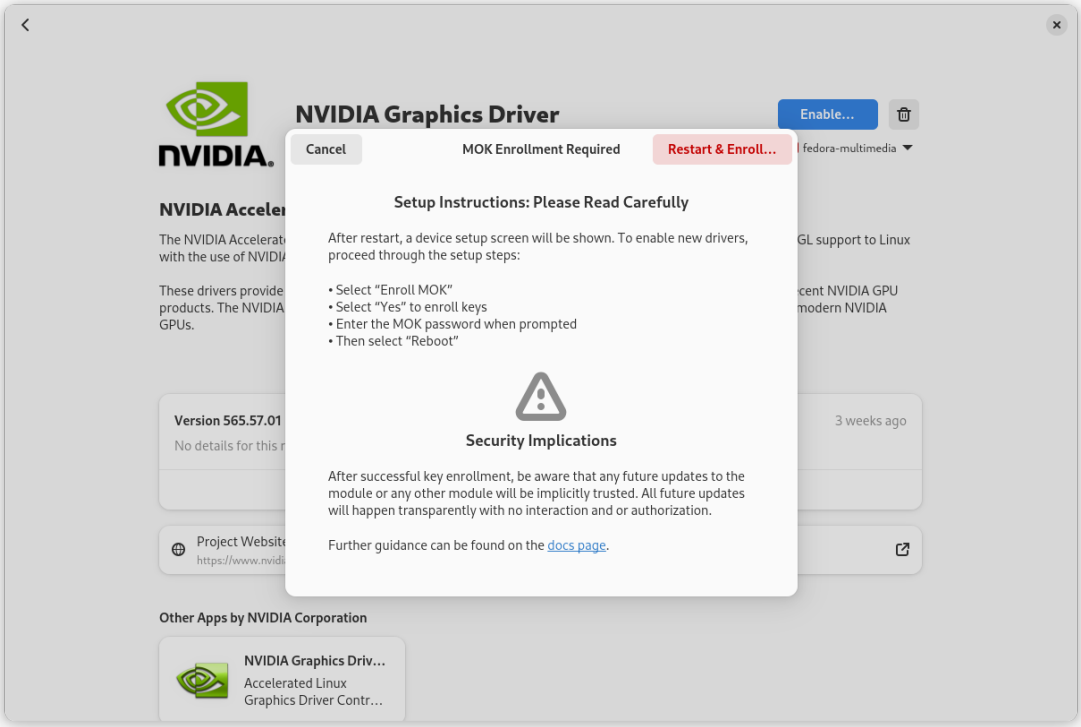
1. Once the driver is installed, the button will switch to *Enable...*. Click on the button.



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.

22.3. Machine Owner Key Enrollment

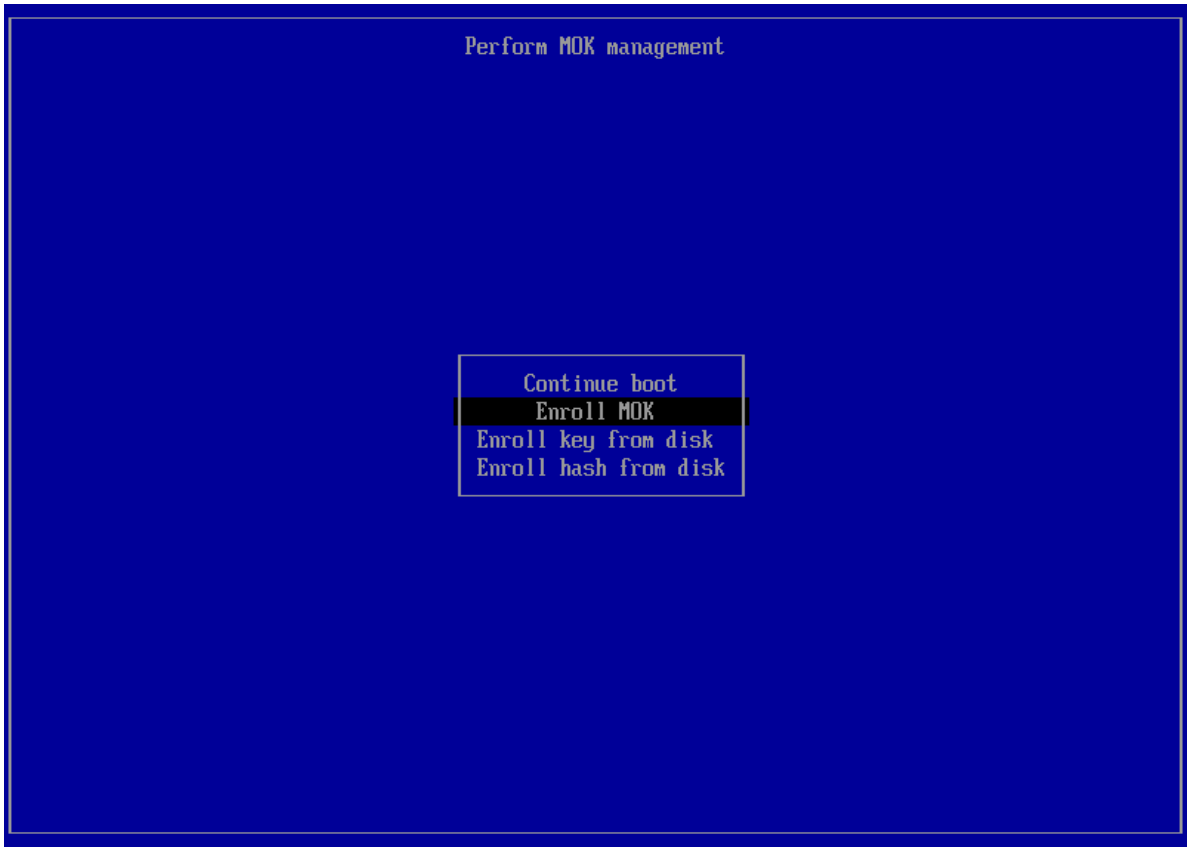
Note

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:

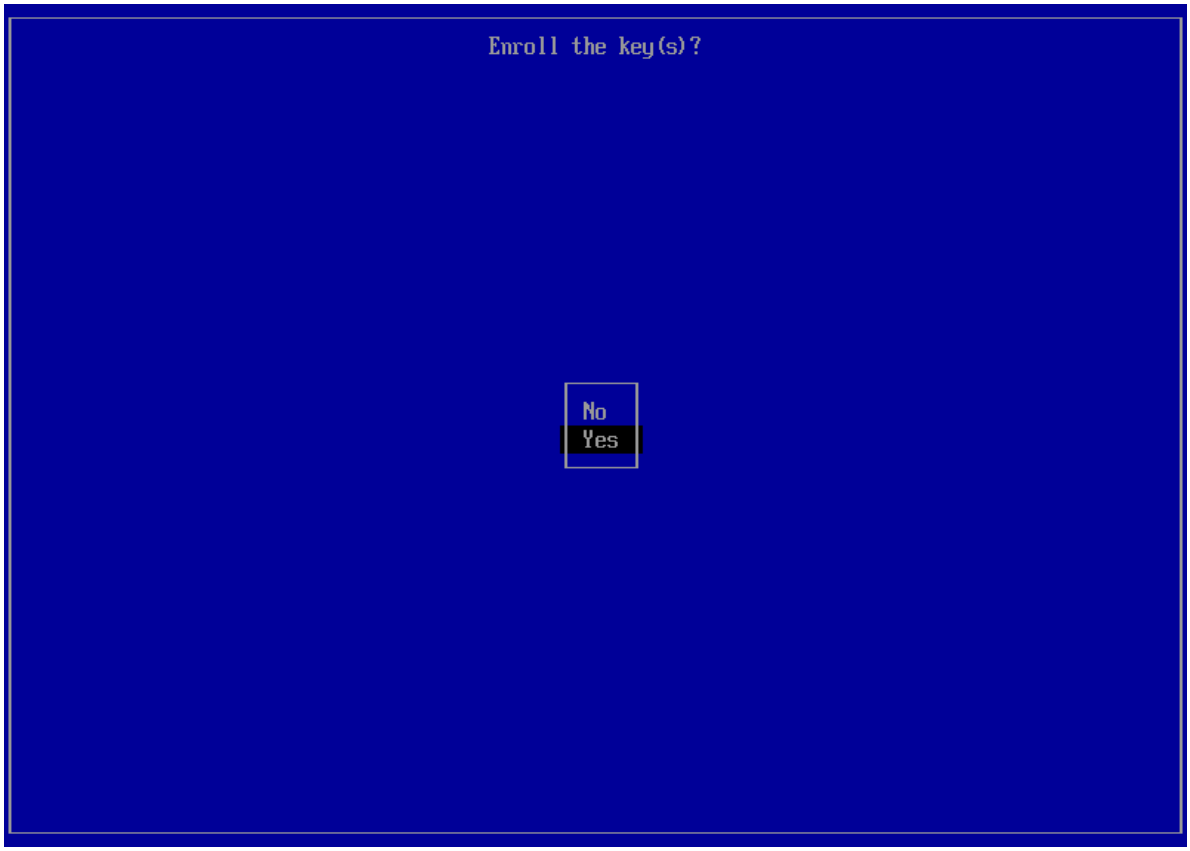
1. Press any key to begin.

2. Select *Enroll MOK*:



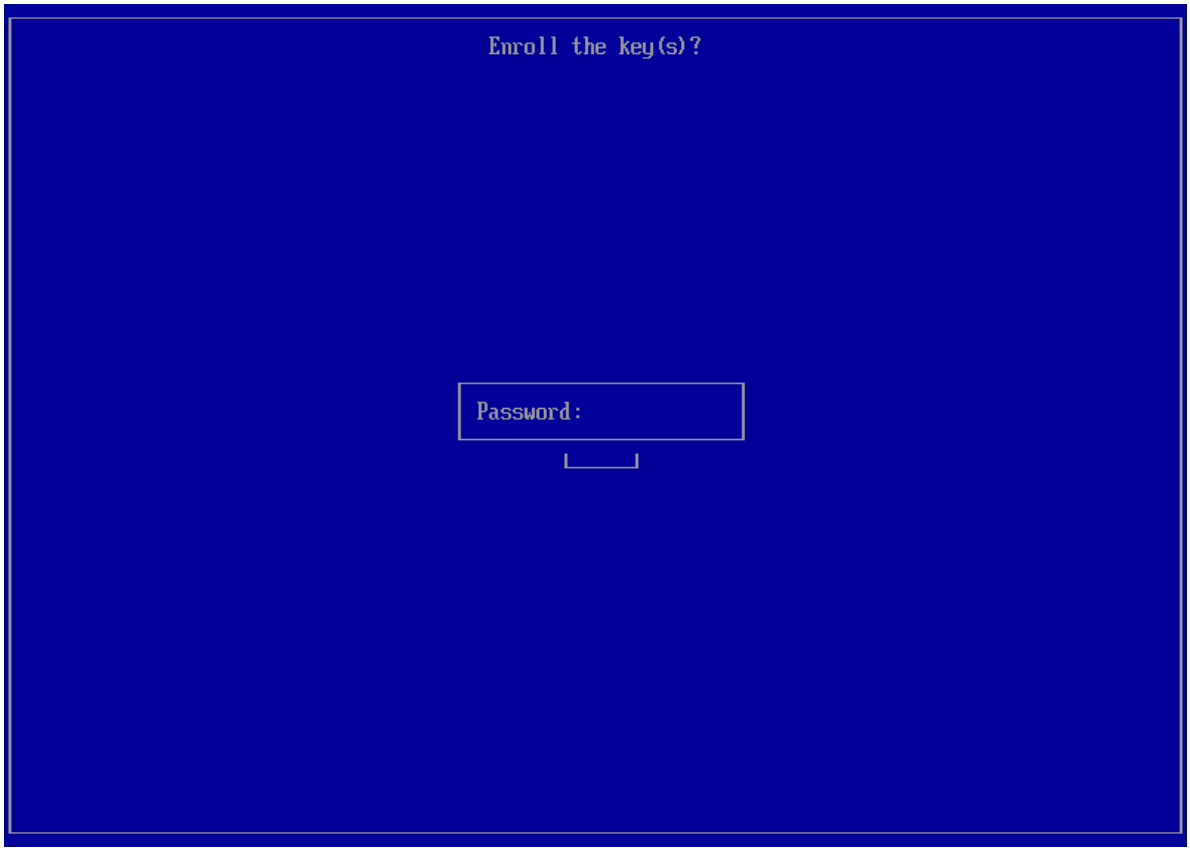
3. Select *Continue* to proceed:

4. Select *Yes* to enroll the key.

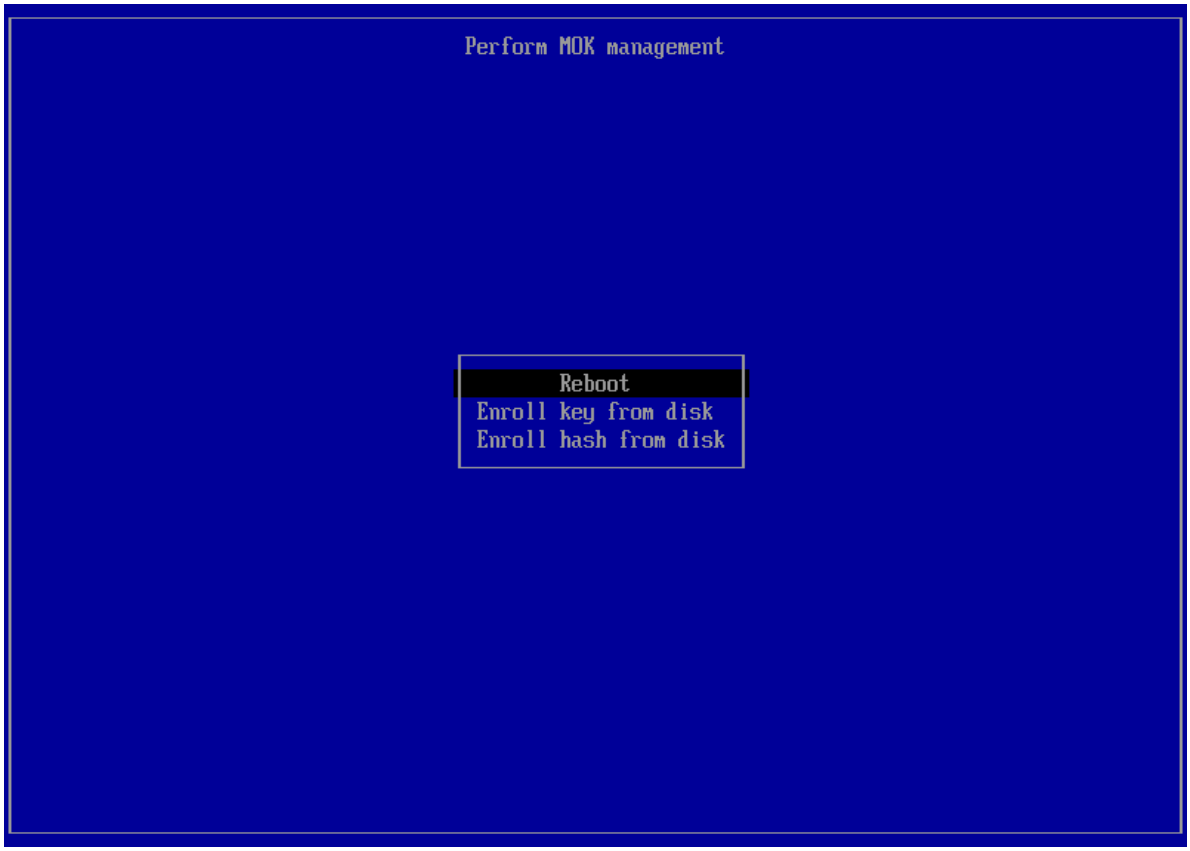


5. Type the MOK password you created for the key during installation.

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

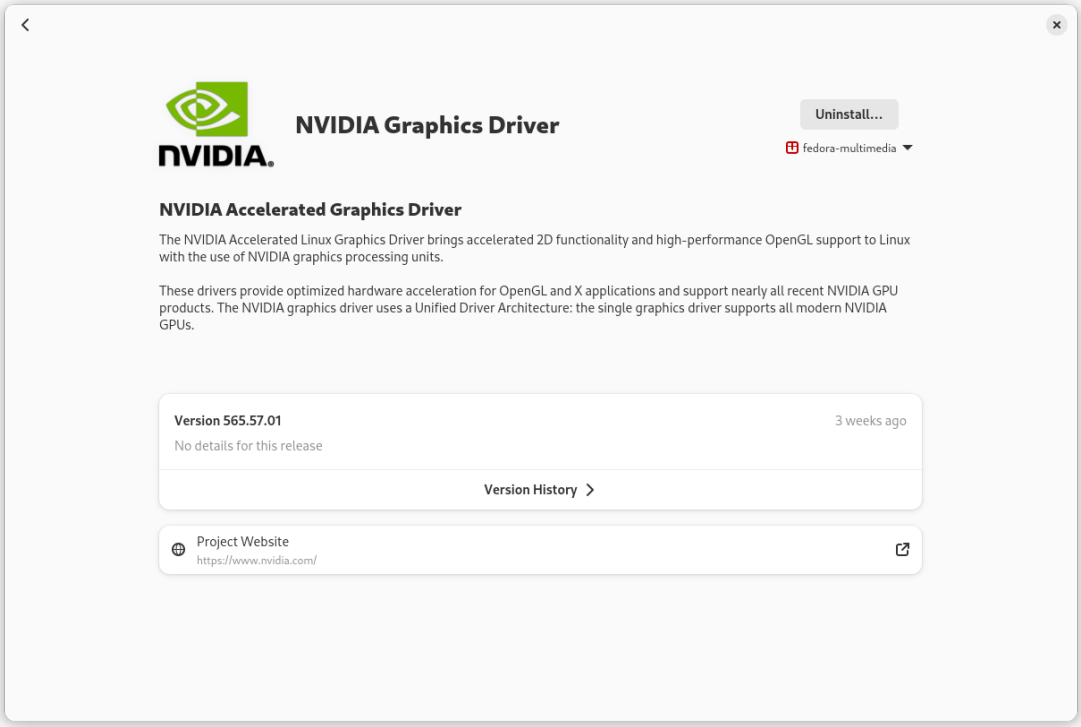


6. Select *Reboot* to reboot into the operating system with the NVIDIA drivers and Secure Boot enabled.

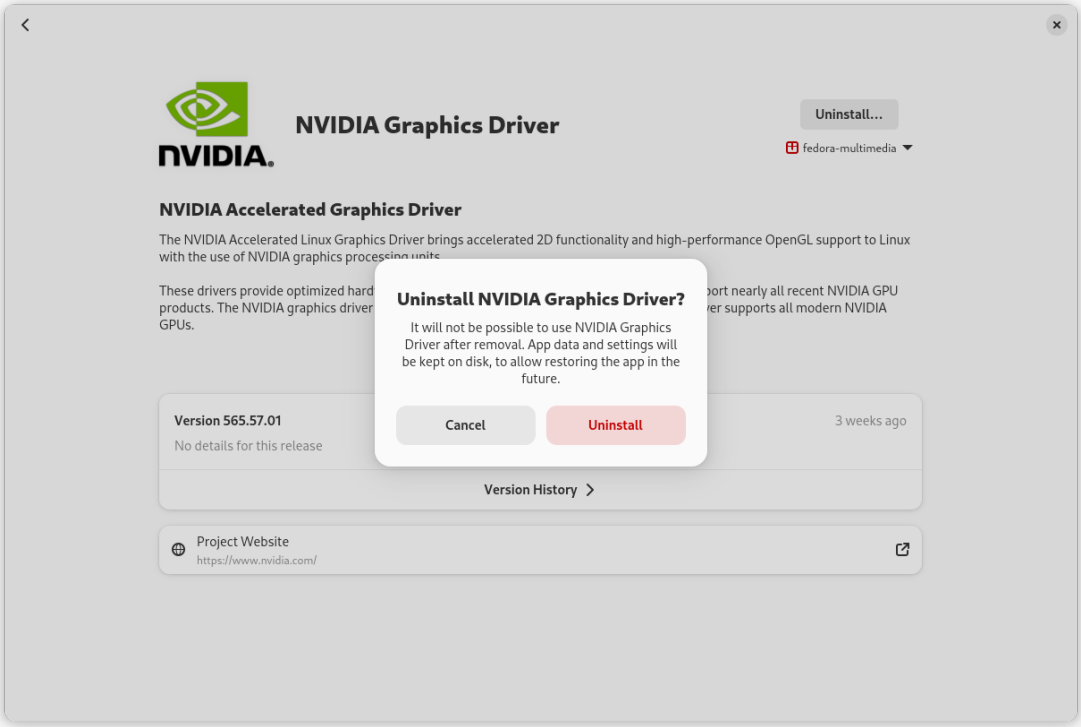


22.4. Uninstallation

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 23. 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 for 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`).

23.1. 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.

23.1.1. 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 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.

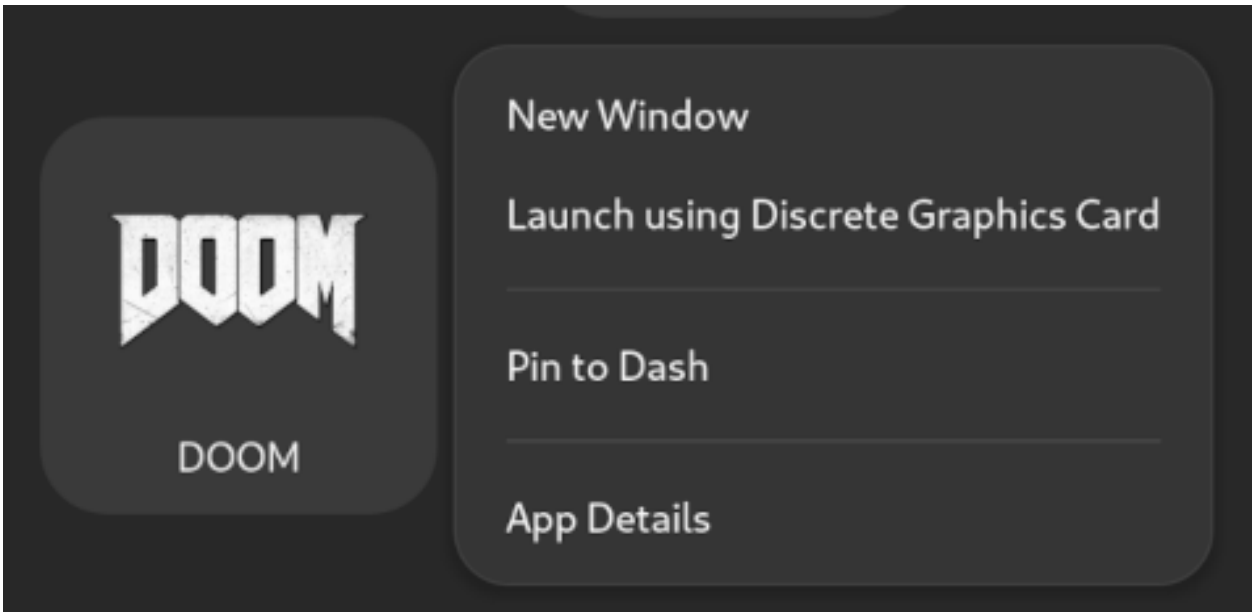
The configuration is flexible, so for example you could have two or more discrete GPUs and one extra audio controller for an eventual HDMI port.

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
```

23.2. 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**:

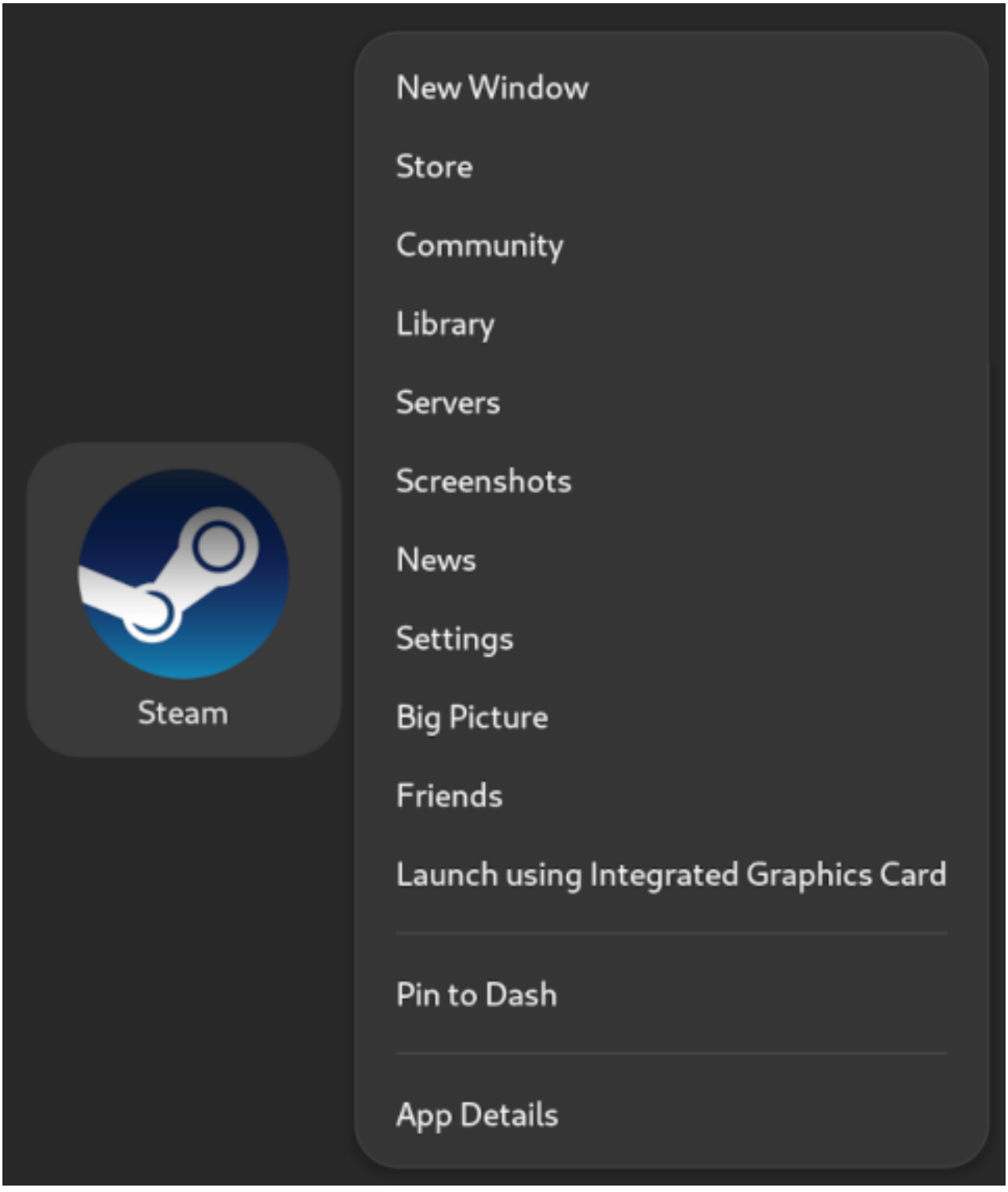


This is supported both in the case of multiple DRI/DRM devices and or a combination with NVIDIA proprietary drivers. There is no visible difference between the two.

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 discrete 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

23.3. 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 Switcheroo 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>
```

23.4. Selecting the GPU to Use with Environment Variables

23.4.1. 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
```

23.4.2. 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
```

(continues on next page)

(continued from previous page)

```
active
active
```

23.4.3. VDPAU Context

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

23.4.4. 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.

23.4.5. 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 24. Advanced Options

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

24.1. 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 615.

Red Hat Enterprise Linux 8/9, AlmaLinux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, Amazon Linux 2023, KylinOS 11

To switch between proprietary and open kernel modules:

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

Red Hat Enterprise Linux 10, AlmaLinux 10, Rocky Linux 10, Oracle Linux 10, Fedora 44

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 3

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

Debian 12/13

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 22.04/24.04/26.04

To switch from proprietary to open:

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

To switch from open to proprietary:

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

SUSE Linux Enterprise Server 15/16, OpenSUSE Leap 15/16

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
```

24.2. 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 10: Kernel Modules Meta Packages

Meta Package	Purpose
`nvidia-open`	Installs all NVIDIA Open GPU kernel modules Driver packages.
`cuda-drivers`	Installs all NVIDIA proprietary kernel modules Driver packages.

24.3. 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 and Red Hat Enterprise Linux 8/9.

Table 11: Modularity Profiles

Stream	Profile	Use Case
Default	`/default`	Installs all the driver packages in a stream.
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>/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.

24.4. Kickstart Installation

Edit the Kickstart configuration file to add the necessary configuration. Replace the placeholder with the appropriate architecture.

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

1. Enable the EPEL repository:

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

2. Enable the CUDA repository:

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

3. Make sure the bootloader configuration is updated with the necessary parameters (ex. disabling `nouveau`, etc.) by adding the following line in the `%post` section:

   ```
   nvidia-boot-update post
   ```

   This is the same command that is executed on a live system when installing the packages.

4. Perform the *Post-installation Actions*.

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

1. Enable the EPEL repository:

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

2. Enable the CUDA repository:

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

3. Make sure the bootloader configuration is updated with the necessary parameters (ex. disabling nouveau, etc.) by adding the following line in the `%post` section:

```
nvidia-boot-update post
```

This is the same command that is executed on a live system when installing the packages.

Red Hat Enterprise Linux 10, AlmaLinux 10, Rocky Linux 10, Oracle Linux 10

1. Enable the EPEL repository:

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

2. Enable the CUDA repository:

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

3. Make sure the bootloader configuration is updated with the necessary parameters (ex. disabling nouveau, etc.) by adding the following line in the `%post` section:

```
nvidia-boot-update post
```

This is the same command that is executed on a live system when installing the packages.

KylinOS 11

1. Enable the CUDA repository:

```
repo --name=cuda-rhel9 --baseurl=https://developer.download.nvidia.com/
↪compute/cuda/repos/kylin11/<arch>/
```

2. Make sure the bootloader configuration is updated with the necessary parameters (ex. disabling nouveau, etc.) by adding the following line in the `%post` section:

```
nvidia-boot-update post
```

This is the same command that is executed on a live system when installing the packages.

3. Perform the *Post-installation Actions*.

24.5. 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

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):
```

24.6. 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.

24.7. 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*
```

24.8. Verbose Versions when Using APT

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

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

Chapter 25. Optional Components

25.1. 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.

Debian 12/13

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

Ubuntu 22.04/24.04/26.04

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

Red Hat Enterprise Linux 8/9, AlmaLinux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, Fedora 44

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

SUSE Enterprise Linux Server 15, OpenSUSE Leap 15

```
# zypper install \
   nvidia-compute-G07-32bit \
   nvidia-gl-G07-32bit \
   nvidia-video-G07-32bit
```

25.2. GPUDirect Storage

The GPU direct storage requires the CUDA network repository to be available. The packages are **not** provided in the local repository installers. With the repository configured, to install the 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** on Red Hat based platforms.

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 leverage 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 additional packages containing the source of the precompiled modules:

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

Ubuntu 22.04/24.04/26.04, Debian 12/13

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

25.3. Using `nvidia-peermem`

The NVIDIA driver packages provide a kernel module, `nvidia-peermem`, which provides NVIDIA InfiniBand based HCAs (Host Channel Adapters) direct peer-to-peer read and write access to the NVIDIA GPU's video memory. It allows GPUDirect RDMA-based applications to use GPU computing power with the RDMA interconnect without needing to copy data to host memory.

This capability is supported with NVIDIA ConnectX®-3 VPI or newer adapters. It works with both InfiniBand and RoCE (RDMA over Converged Ethernet) technologies.

NVIDIA OFED (Open Fabrics Enterprise Distribution), or MLNX_OFED, introduces an API between the InfiniBand Core and peer memory clients such as NVIDIA GPUs. The nvidia-peermem module registers the NVIDIA GPU with the InfiniBand subsystem by using peer-to-peer APIs provided by the NVIDIA GPU driver.

The kernel must have the required support for RDMA peer memory either through additional patches to the kernel or via MLNX_OFED as a prerequisite for loading and using `nvidia-peermem`. So unless the MLNX_OFED distribution is installed prior to installing the NVIDIA driver, the `nvidia-peermem` module will simply be built as a stub. Also whenever the OFED distribution changes, the module must be rebuilt to match the updated OFED kernel modules.

25.3.1. Automatic rebuild of `nvidia-peermem`

DKMS has been extended to support automatic rebuilding of dependent kernel modules explicitly for this use case, so a specific configuration can be added to the system which requires `nvidia-peermem` with OFED support. It's sufficient to add the following snippet to `/etc/dkms/nvidia.conf` prior or after installing the GPU driver or the OFED components:

```
BUILD_DEPENDS[4]="mlnx-ofed-kernel"
BUILD_DEPENDS_REBUILD="yes"
```

The first line refers to the `nvidia-peermem` module as defined in the `/usr/src/nvidia-*/dkms.conf` file; the number between square brackets must match with the module ID in that file. The second part is the DKMS module (so the DKMS "module" as a whole) on which it depends on.

The second line instructs DKMS to make sure the module ID declared in the first line changes if DKMS tracks a dependency change (update, rebuild for a new kernel, etc).

If the OFED distribution has been installed after the driver, a simple DKMS rebuild should suffice. An example with driver 590 on Fedora.

First, we remove the installed modules, so in case of a reinstall, these are not backed up:

```
# dkms status -m nvidia
nvidia/590.44.01, 6.17.11-300.fc42.x86_64, x86_64: installed
# dkms uninstall -m nvidia/590.44.01
Module nvidia/590.44.01 for kernel 6.17.11-300.fc42.x86_64 (x86_64):
Before uninstall, this module version was ACTIVE on this kernel.
Deleting /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia.ko.xz
Deleting /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-modeset.ko.xz
Deleting /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-drm.ko.xz
Deleting /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-uvm.ko.xz
Deleting /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-peermem.ko.xz
Running depmod..... done.
Executing post-transaction command.......... done.
# dkms status -m nvidia
nvidia/590.44.01, 6.17.11-300.fc42.x86_64, x86_64: built
```

Then, we force a rebuild, so `nvidia-peermem` is rebuilt and correctly linked to OFED:

```
# sudo dkms build -m nvidia/590.44.01 --force
Sign command: /lib/modules/6.17.11-300.fc42.x86_64/build/scripts/sign-file
Signing key: /var/lib/dkms/mok.key
Public certificate (MOK): /var/lib/dkms/mok.pub

Building module(s)..................... done.
Signing module /var/lib/dkms/nvidia/590.44.01/build/kernel-open/nvidia.ko
Signing module /var/lib/dkms/nvidia/590.44.01/build/kernel-open/nvidia-
↪modeset.ko
```

(continues on next page)

(continued from previous page)

```
Signing module /var/lib/dkms/nvidia/590.44.01/build/kernel-open/nvidia-drm.ko
Signing module /var/lib/dkms/nvidia/590.44.01/build/kernel-open/nvidia-uvm.ko
Signing module /var/lib/dkms/nvidia/590.44.01/build/kernel-open/nvidia-
↪peermem.ko
```

And then we install it again:

```
# dkms install -m nvidia/590.44.01
Installing /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia.ko.xz
Installing /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-modeset.ko.xz
Installing /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-drm.ko.xz
Installing /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-uvm.ko.xz
Installing /lib/modules/6.17.11-300.fc42.x86_64/extra/nvidia-peermem.ko.xz
Running depmod..... done.
Executing post-transaction command..... done.
```

25.4. NVSwitch

To install Fabric Manager, NSCQ, NVSDM, IMEX:

Red Hat Enterprise Linux 8/9, AlmaLinux 8/9, Rocky Linux 8/9, Oracle Linux 8/9, Amazon Linux 2023, Kylin 11

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

Red Hat Enterprise Linux 10, AlmaLinux 10, Rocky Linux 10, Oracle Linux 10, Fedora 44

```
# dnf install nvidia-fabricmanager libnvidia-nscq libnvsdm nvidia-imex
```

SUSE Enterprise Linux Server 15/16, openSUSE Leap 15/16

```
# zypper install nvidia-fabricmanager libnvidia-nscq libnvsdm nvidia-imex
```

Azure Linux 3

```
# tdnf install nvidia-fabricmanager libnvidia-nscq libnvsdm nvidia-imex
```

Warning

tdnf, the package manager used in Azure Linux, **has issues with dependency solving.** It might be that the version that gets installed is not the one expected.

As an example of this issue, at the moment of writing this, a very old virtual provide is used in place of the correct package with a higher version:

```
root [ / ]# tdnf list nvidia-fabricmanager nvidia-fabric-manager
Loaded plugin: tdnfrepogpgcheck
nvidia-fabricmanager.x86_64          580.105.08-1        cuda-azl3-x86_64
nvidia-fabricmanager.x86_64          580.65.06-1         cuda-azl3-x86_64
nvidia-fabricmanager.x86_64          580.82.07-1         cuda-azl3-x86_64
```

(continues on next page)

(continued from previous page)

```
nvidia-fabricmanager.x86_64          580.95.05-1         cuda-azl3-x86_64
nvidia-fabricmanager.x86_64          590.44.01-1.azl3    cuda-azl3-x86_64
nvidia-fabricmanager.x86_64          590.48.01-1.azl3    cuda-azl3-x86_64
nvidia-fabric-manager.x86_64         575.51.03-1         cuda-azl3-x86_64
nvidia-fabric-manager.x86_64         575.57.08-1         cuda-azl3-x86_64
root [ / ]# tdnf install nvidia-fabricmanager
Loaded plugin: tdnfrepogpgcheck

Installing:
nvidia-fabric-manager x86_64         575.57.08-1         cuda-azl3-x86_64  36.
↪56M      10.06M

Total installed size:  36.56M
Total download size:  10.06M
Is this ok [y/N]: ^C
root [ / ]# tdnf install nvidia-fabricmanager --best
Loaded plugin: tdnfrepogpgcheck

Installing:
nvidia-fabric-manager x86_64         575.57.08-1         cuda-azl3-x86_64  36.
↪56M      10.06M

Total installed size:  36.56M
Total download size:  10.06M
Is this ok [y/N]: ^C
```

In this case, specifying the major version is a better choice:

```
# tdnf install nvidia-fabricmanager-590
```

Ubuntu 22.04/24.04/26.04, Debian 12/13

```
# apt install -V nvidia-fabricmanager libnvidia-nscq libnvsdm nvidia-imex
```

If you want to target a specific branch or release, these packages are tracked by the Version Locking packages. For more information please refer to the *version locking section for APT*.

Chapter 26. DLSS / Smooth Motion / Reflex

This is a high level guide, with the most common settings. Links are provided for more details and configuration on each specific component.

Common DLSS features you may see in games to improve performance:

- **DLSS Super Resolution (SR)**: Reconstructs higher-resolution frames from a lower-resolution render.
- **DLSS Frame Generation (FG)**: Generates additional frames.
- **DLSS Ray Reconstruction (RR)**: Improves ray-traced lighting quality by replacing traditional denoising with AI reconstruction.
- **DLSS Anti-Aliasing (DLAA)**: Uses DLSS technology for anti-aliasing at native resolution.

DLSS features that are used and "consumed" independently of the game implementation:

- **Smooth Motion**: Frame generation feature for titles that do not natively support DLSS Frame Generation. It delivers smoother gameplay by inferring an additional frame between two rendered frames.

When a game has native support for DLSS, there are usually settings to enable the various DLSS technologies without anything that needs to be done on the system running the game. The libraries and options are game specific, so the options can vary substantially depending on the title and the age of the title. For games that do not support NVAPI, DLSS options are missing in-game.

This applies to both native Linux games as well to Windows games runnning via Steam Play / Proton. On Linux, Proton must expose NVIDIA's APIs (NVAPI) for DLSS to function. This is done automatically with the included DXVK-NVAPI as part of a standard Proton installations.

Smooth Motion has no requirement, it can be enabled for any title without native DLSS support or with titles with native DLSS support disabled.

Warning

Native DLSS Frame Generation and Smooth Motion are competing technologies and **should not be used together**. Doing so results in lower performance and visual artifacts.

Other technologies:

- **NVIDIA Reflex**: A technology that reduces system latency in games by synchronizing CPU and GPU work so your inputs appear on screen faster.

26.1. Smooth Motion

Smooth Motion is enabled via an environment variable:

```
NVPRESENT_ENABLE_SMOOTH_MOTION=1
```

When enabling this, the Frame Generation is transparent to the game engine and gets applied automatically.

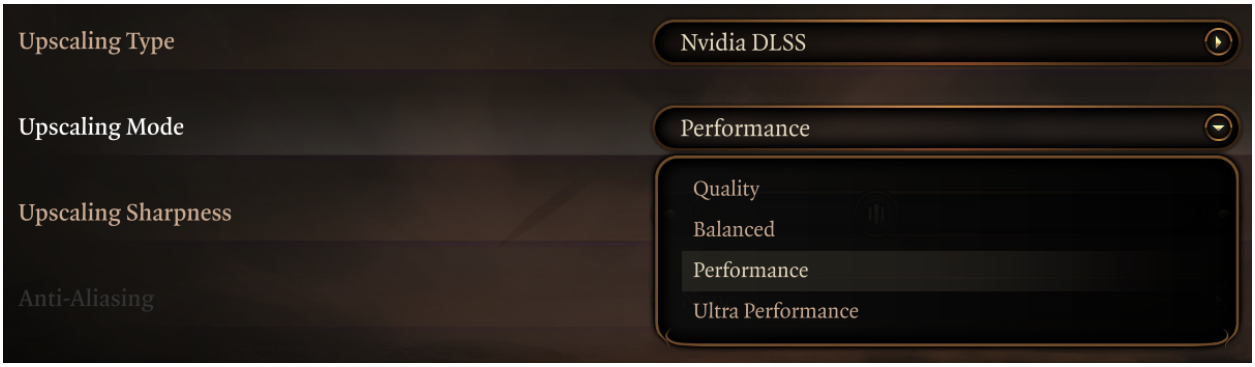
See also

Chapter 39. NVIDIA Smooth Motion
NVIDIA Accelerated Linux Graphics Driver README

26.2. DLSS SR/FG/RR

DLSS support is normally enabled in the game settings, and settings themselves depend on the game engine and update level. For example for *Doom Eternal* on Proton:

And with the Linux native port of *Baldur's Gate 3*:



26.2.1. DLSS Update (Steam Play / Proton)

Again with the example of *Doom Eternal*, we can see that the game originally ships with DLSS 2.3:

```
$ cd ~/.local/share/Steam/steamapps
$ peres -v common/DOOMEternal/nvngx_dlss.dll
File Version:                   65263.1213.1.0
Product Version:                2.3.0.0
```

This can also be verified while the game is running with the *DLSS Indicator (Steam Play / Proton)*:

So DLSS models are limited to the version of the Streamline SDK libraries that the game originally shipped with; but libraries and the settings can also be overridden.

By setting the following environment variable, the DLLs shipped with the game can be overridden:

```
PROTON_ENABLE_NGX_UPDATER=1
```

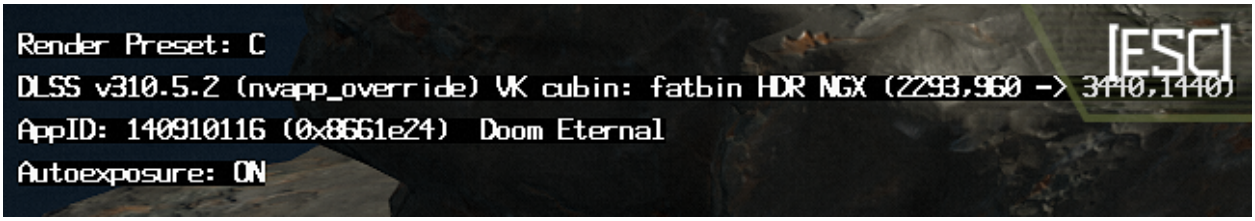
This result in the following files being added to the Proton root folder (the virtual Windows system drive) at runtime:

```
$ grep DOOMEternal appmanifest_*
appmanifest_782330.acf:      "installdir"            "DOOMEternal"
$ cd compatdata/782330/pfx/drive_c/
$ find ProgramData/NVIDIA/ -type f
ProgramData/NVIDIA/NGX/models/config/versions/2/files/nvngx_server_config.txt
ProgramData/NVIDIA/NGX/models/config/versions/1/files/nvngx_mapping.json
ProgramData/NVIDIA/NGX/models/config/versions/1/files/nvngx_deny_list.txt
ProgramData/NVIDIA/NGX/models/dlss/versions/20317442/files/160_E658700.bin
ProgramData/NVIDIA/NGX/models/nvngx_config.txt
```

And by looking at some of those, we can see that the DLSS versions have been overridden:

```
$ cat ProgramData/NVIDIA/NGX/models/nvngx_config.txt
[dlss]
app_E658700 = 310.5.2
$ peres -v ProgramData/NVIDIA/NGX/models/dlss/versions/20317442/files/160_
↪E658700.bin
File Version:                   65263.1213.1.0
Product Version:                310.5.2.0
```

This will only override the libraries, not the game engine settings. Again this can also be verified while the game is running with the *DLSS Indicator (Steam Play / Proton)*:



The amount of libraries downloaded depends on the features implemented by the game engine. For example, for *Cyberpunk 2077*, we can see that three libraries are updated:

```
$ cd compatdata/1091500/pfx/drive_c/ProgramData/NVIDIA/NGX
$ find models -name "*.*"
models/config/versions/2/files/nvngx_server_config.txt
models/config/versions/1/files/nvngx_mapping.json
models/config/versions/1/files/nvngx_deny_list.txt
```

(continues on next page)

(continued from previous page)

```
models/dlss/versions/131844/files/160_B9DB490.bin
models/dlss/versions/20317442/files/160_E658700.bin
models/nvngx_config.txt
models/dlssg/versions/20317442/files/160_E658700.bin
models/dlssd/versions/20317442/files/160_E658700.bin
$ cat nvngx_config.txt
[dlss]
app_B9DB490 = 2.3.4
app_E658700 = 310.5.2

[dlssg]
app_E658700 = 310.5.2

[dlssd]
app_E658700 = 310.5.2
```

`dlss` is for Super Resolution, `dlssg` for Frame Generation and `dlssd` is for Ray Reconstruction.

Note

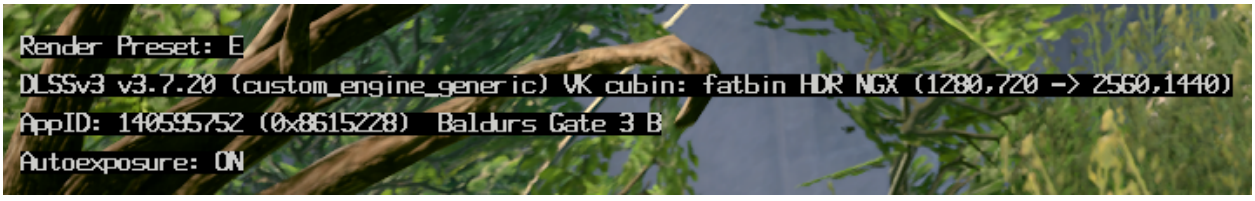
Some titles prevent DLL updates even with `PROTON_ENABLE_NGX_UPDATER=1`, like *Deep Rock Galactic*. These titles require also the *override of presets and snippets* as described in the next section to fully unblock the updated libraries.

26.2.2. DLSS Update (native)

Again with the example of *Baldur's Gate 3*, we can see that the game originally ships with DLSS 3.7:

```
$ cd ~/.local/share/Steam/steamapps/common/Baldurs\ Gate\ 3/bin/
$ ls -1sh *dlss*
42M libnvidia-ngx-dlss.so.3.7.20
```

This can also be verified while the game is running with the *DLSS Indicator (native)*:

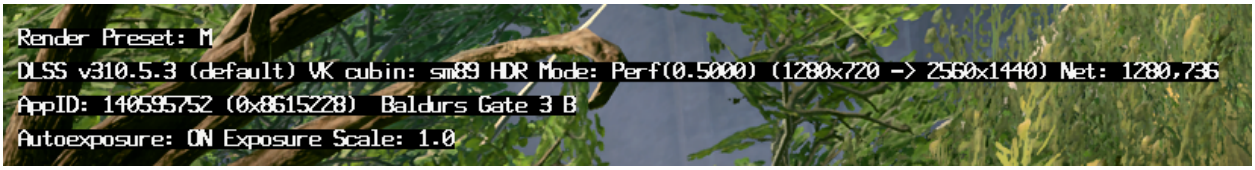


So DLSS models are limited to that version of the libraries that the game originally shipped with, but libraries can also be overridden.

To update the DLSS libraries to a later version:

```
$ wget https://github.com/NVIDIA/DLSS/raw/refs/heads/main/lib/Linux_x86_64/
↪rel/libnvidia-ngx-dlss.so.310.5.3
$ chmod +x libnvidia-ngx-dlss.so.310.5.3
$ ls -1sh *dlss*
53M libnvidia-ngx-dlss.so.310.5.3
42M libnvidia-ngx-dlss.so.3.7.20
```

This will only override the libraries, not the game engine settings. This can also be verified while the game is running with the *DLSS Indicator (native)*:



Note

Overriding presets is currently not supported on Linux.

26.2.3. DLSS Settings Override (Steam Play / Proton)

After updating the libraries (DLLs) to the latest published versions, it's also possible to override game settings. To override settings for Super Resolution, Ray Reconstruction and Frame Generation and as well use the recommended presets available from the Streamline SDK you might also want to add these variables:

```
DXVK_NVAPI_DRS_NGX_DLSS_SR_OVERRIDE=on
DXVK_NVAPI_DRS_NGX_DLSS_RR_OVERRIDE=on
DXVK_NVAPI_DRS_NGX_DLSS_FG_OVERRIDE=on

DXVK_NVAPI_DRS_NGX_DLSS_SR_OVERRIDE_RENDER_PRESET_SELECTION=render_preset_
↪latest
DXVK_NVAPI_DRS_NGX_DLSS_RR_OVERRIDE_RENDER_PRESET_SELECTION=render_preset_
↪latest
```

This will result in a complete override of all settings in the game regardless of what is configured in the game engine settings. As shown by the *DLSS Indicator (Steam Play / Proton)*:

Attention

NVAPI headers defining the presets available are included in DXVK-NVAPI, which is then in turn included in Proton. **Depending on the age of the Proton release not all presets might available.**

In this case, switch to a newer major Proton version or *Proton Experimental* that might contain a more recent version of the DXVK-NVAPI libraries.

See also

NGX snippet updates and preset overrides
DXVK-NVAPI wiki

26.2.4. DLSS Indicator (Steam Play / Proton)

Applied DLSS Settings can be verified in-game through the use of an additional environment variable:

```
DXVK_NVAPI_SET_NGX_DEBUG_OPTIONS=DLSSIndicator=1024,DLSSGIndicator=2
```

This will enable the overlay information (DLSS Indicator) at the bottom of the screen for Super Resolution and at the top for Frame Generation. These are the equivalent values of the Windows Registry settings, as specified in the programming guide linked below.

Note

These indicators are written to the registry inside the `wineprefix` (`~/.local/share/Steam/steamapps/compatdata/<product id>`) of the game. This means that once set, the Indicators are permanent; and they need to be set again with their value at zero at least once to make them disappear:

```
DXVK_NVAPI_SET_NGX_DEBUG_OPTIONS=DLSSIndicator=0,DLSSGIndicator=0
```

See also

20.0 DLSS-FG INDICATOR TEXT
Streamline - DLSS-G

3.18 Current DLSS Settings
NVIDIA DLSS Super Resolution Programming Guide

26.2.5. DLSS Indicator (native)

Applied DLSS Settings can be verified in-game through the use of an additional environment variable:

```
__NGX_SHOW_INDICATOR=1024
```

This will enable the overlay information (DLSS Indicator) at the bottom of the screen for Super Resolution. This is the equivalent of the Windows Registry settings, as specified in the programming guide linked below.

See also

3.18 Current DLSS Settings
NVIDIA DLSS Super Resolution Programming Guide

26.3. Reflex

NVIDIA Reflex is a technology by NVIDIA that reduces system latency by synchronizing the GPU and CPU to deliver faster, more responsive input-to-screen gameplay, especially in competitive games.

26.3.1. Reflex for Vulkan (Steam Play / Proton)

Reflex is supported out of the box in DirectX games via DXVK-NVAPI; but in order to support the Vulkan flavor of NVIDIA Reflex, an additional Vulkan layer is required.

This layer intercepts Vulkan calls made by DXVK-NVAPI, enriches them with extra data, and forwards to a Vulkan driver that supports `VK_NV_low_latency2` device extension. The Vulkan extension gives applications timing suggestions on when to start the recording of new frames to reduce the latency between input sampling and frame presentation.

To install the layer, first of all download the latest DXVK-NVAPI release and then:

```
$ tar -xzf dxvk-nvapi-*.tar.gz --strip-components=2 ./layer
$ sed -i -e 's|./libdxvk|libdxvk|g' VkLayer_DXVK_NVAPI_reflex.json
# install -m0644 -D VkLayer_DXVK_NVAPI_reflex.json \
    /usr/local/share/vulkan/implicit_layer.d/VkLayer_DXVK_NVAPI_reflex.json
```

Then for Debian/Ubuntu systems:

```
# install -m0755 -D libdxvk_nvapi_vkreflex_layer.so \
    /usr/local/lib/x86_64-linux-gnu/libdxvk_nvapi_vkreflex_layer.so
```

For all other RPM based distributions:

```
# install -m0755 -D libdxvk_nvapi_vkreflex_layer.so \
   /usr/local/lib64/libdxvk_nvapi_vkreflex_layer.so
```

Then a simple check will report if the Vulkan layer is available on the system:

```
$ vulkaninfo --summary | grep NVAPI
VK_LAYER_DXVK_NVAPI_reflex         DXVK-NVAPI Vulkan Reflex compatibility
↪layer   1.3.295  version 1
```

Because the layer could interfere with other Vulkan applications that don't use Reflex, it is currently **disabled by default** even when installed. To enable the layer, set the following environment variable:

```
DXVK_NVAPI_VKREFLEX=1
```

See also

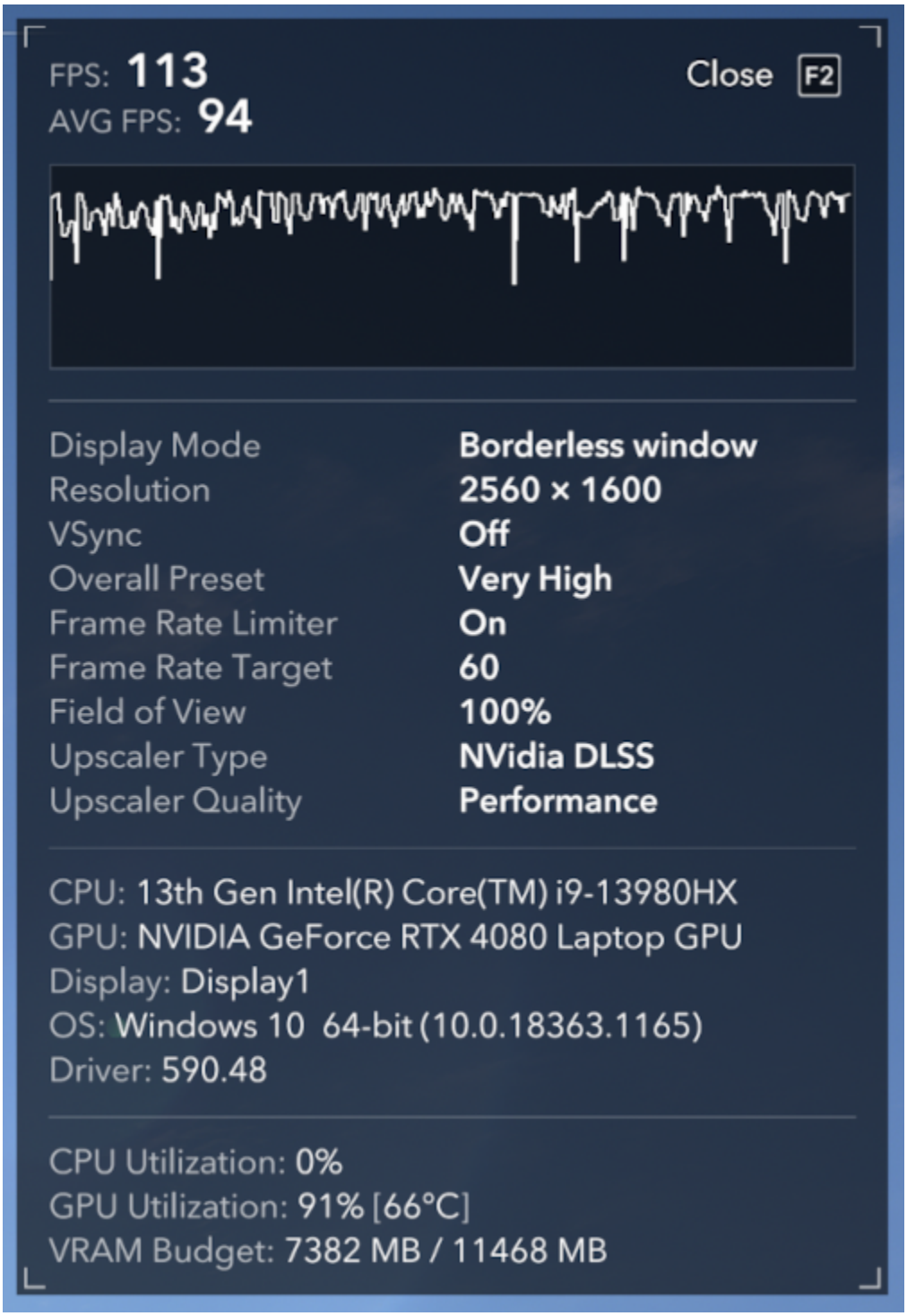
Vulkan Reflex layer
DXVK-NVAPI README.md

26.3.2. Reflex (native)

Reflex is supported, not via the Reflex SDK but directly via the Vulkan extension `VK_NV_low_latency2` in games that natively implement that. The Vulkan extension gives applications timing suggestions on when to start the recording of new frames to reduce the latency between input sampling and frame presentation.

26.4. GPU utilization reporting (Steam Play / Proton)

On the titles that support it, GPU utlization and usage reporting is done through the NVIDIA Management Library (NVML). For example, on *Assassin's Creed Shadows*:



If the NVML library support is missing, the reporting is not correct:

Not being part of Wine/Proton, Wine NVML needs to be manually installed. First of all, download the latest release from the wine-nvml Github project.

Then proceed with the installation. For example, with Proton 10:

```
$ PROTON_LIBS=~/.local/share/Steam/steamapps/common/Proton\ 10.0/files/lib/
↪wine
$ unzip wine-nvml*.zip
$ install -m0555 -p lib64/wine/x86_64-windows/nvml.dll "${PROTON_LIBS}/x86_64-
↪windows/"
$ install -m0555 -p lib64/wine/x86_64-unix/nvml.so "${PROTON_LIBS}/x86_64-
↪unix/"
```

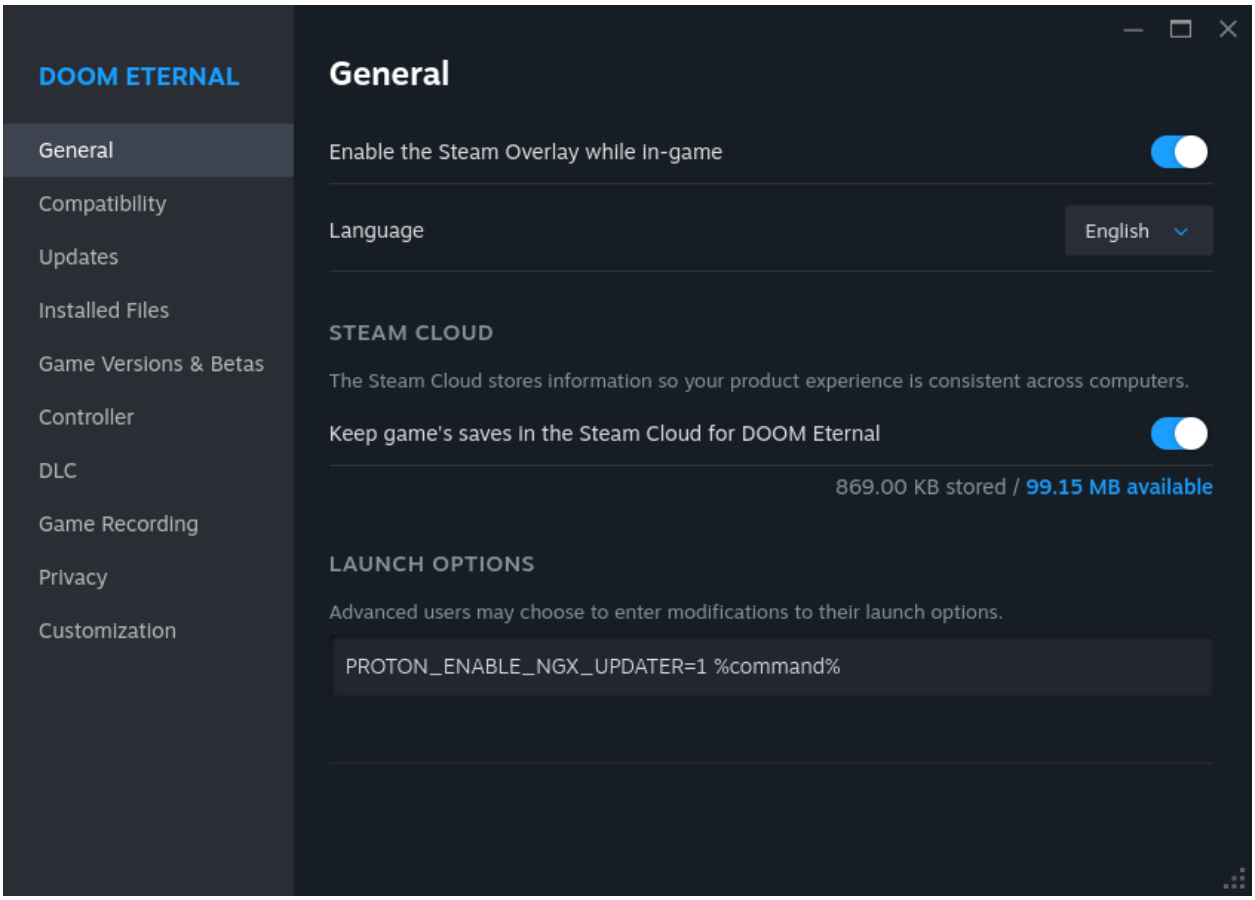
The `wine-nvml` files are not part of a standard Proton installation, so the files are **not overwritten or deleted** in case of updates or manual integrity verification.

See also

wine-nvml project page

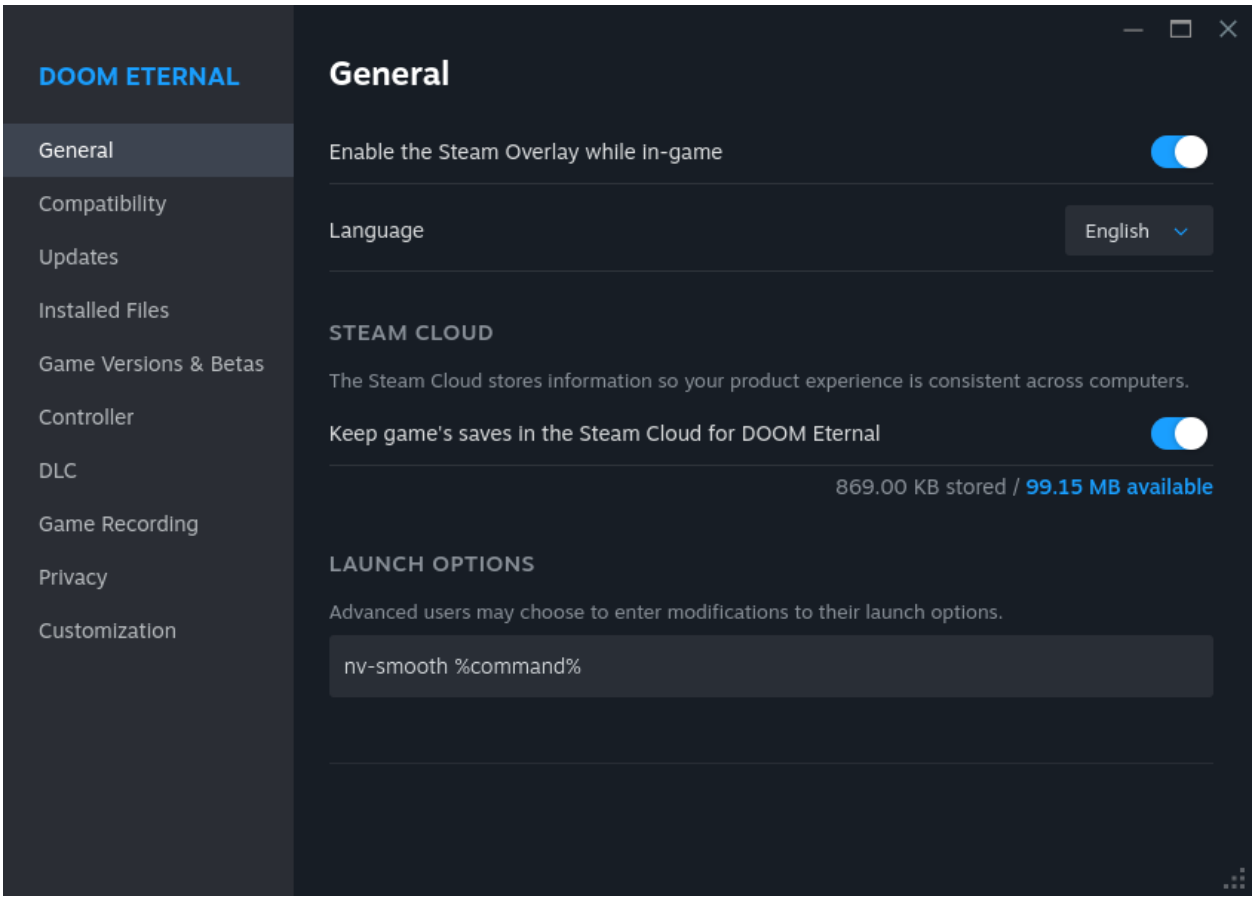
26.5. Applying settings to games

There are multiple ways settings can be applied to a specific game; for example directly via the Steam UI:



Through a script that's applied to the Steam UI:

```
$ cat /usr/local/bin/nv-smooth
#!/usr/bin/bash
export NVPRESENT_ENABLE_SMOOTH_MOTION=1
export DXVK_NVAPI_VKREFLEX=1
exec gamemoderun "$@"
```



Through a specific Proton settings file, which will apply the same setting to **all** titles started with this particular Proton version:

```
$ cat .local/share/Steam/steamapps/common/Proton\ -\ Experimental/user_
↪settings.py
user_settings = {
  "PROTON_ENABLE_NGX_UPDATER": "1",
  "DXVK_NVAPI_DRS_NGX_DLSS_SR_OVERRIDE": "on",
  "DXVK_NVAPI_DRS_NGX_DLSS_RR_OVERRIDE": "on",
  "DXVK_NVAPI_DRS_NGX_DLSS_FG_OVERRIDE": "on",
  "DXVK_NVAPI_DRS_NGX_DLSS_SR_OVERRIDE_RENDER_PRESET_SELECTION": "render_
↪preset_latest",
  "DXVK_NVAPI_DRS_NGX_DLSS_RR_OVERRIDE_RENDER_PRESET_SELECTION": "render_
↪preset_latest",
  "DXVK_NVAPI_VKREFLEX": "1"
}
```

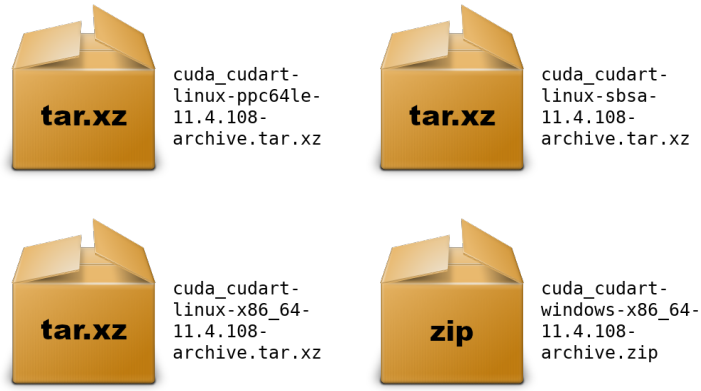
Note

When using a combination of the above, for example using generic Proton settings, a custom script and specific variables in front of the script; **launch options take precedence above the generic proton settings**.

Chapter 27. 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.

Chapter 28. Post-installation Actions

The post-installation actions must be manually performed.

28.1. 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 nvidia-persistenced
```

28.2. 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
```

28.3. Local Repository Removal

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

Amazon Linux 2023, Fedora 44, KylinOS 11, Red Hat Enterprise Linux 8/9/10, AlmaLinux 8/9/10, Rocky Linux 8/9/10, Oracle Linux 8/9/10

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

Azure Linux 3

```
# tdnf remove nvidia-driver-local-repo-\*
```

Ubuntu 22.04/24.04/24.04, Debian 12

```
# apt remove --purge nvidia-driver-local-repo\*
```

SUSE Linux Enterprise Server 15/16, OpenSUSE Leap 15/16

```
# zypper remove nvidia-driver-local-repo\*
```

Chapter 29. 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 11, Red Hat Enterprise Linux 8/9, AlmaLinux 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
```

Red Hat Enterprise Linux 10, AlmaLinux 10, Rocky Linux 10, Oracle Linux 10, Fedora 44

To remove the NVIDIA driver:

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

Azure Linux 3

Unfortuantely `tdnf` does not support wildcards; so to remove the NVIDIA driver:

```
# rpm -qa | grep -iE '(nvidia|cuda)' | xargs -r tdnf remove -y
```

OpenSUSE 15/16 and SUSE Enterprise Linux Server 15/16

To remove the NVIDIA driver:

```
# zypper remove \
   cuda-compat\* \
   cuda-drivers\* \
   libnvidia-\* \
   nvidia-\*
```

Ubuntu 22.04/24.04/26.04

To remove the NVIDIA driver:

```
# apt remove --autoremove --purge -V \
   cuda-compat\* \
   cuda-drivers\*  \
   libnvidia-cfg1\* \
   libnvidia-compute\* \
   libnvidia-decode\* \
   libnvidia-encode\* \
   libnvidia-extra\* \
```

(continues on next page)

(continued from previous page)

```
   libnvidia-fbc1\* \
   libnvidia-gl\* \
   libnvidia-gpucomp\* \
   libnvidia-nscq\* \
   libnvsdm\* \
   libxnvctrl\* \
   nvidia-dkms\* \
   nvidia-driver\* \
   nvidia-fabricmanager\* \
   nvidia-firmware\* \
   nvidia-headless\* \
   nvidia-imex\* \
   nvidia-kernel\* \
   nvidia-modprobe\* \
   nvidia-open\* \
   nvidia-persistenced\* \
   nvidia-settings\* \
   nvidia-xconfig\* \
   xserver-xorg-video-nvidia\*
```

If the 32 bit compatibility libraries are installed along with the main architecture on `amd64`:

```
# apt remove --autoremove --purge -V \
   libnvidia-compute\*:i386 \
   libnvidia-decode\*:i386 \
   libnvidia-encode\*:i386 \
   libnvidia-fbc1\*:i386 \
   libnvidia-gpucomp\*:i386 \
   libnvidia-gl\*:i386
```

Debian 12/13

To remove the NVIDIA driver:

```
# apt remove --autoremove --purge -V \
   cuda-compat\* \
   cuda-drivers\* \
   cuda-mps\* \
   firmware-nvidia-gsp\* \
   libcuda1\* \
   libcudadebugger1\* \
   libegl-nvidia0\* \
   libgles-nvidia1\* \
   libgles-nvidia2\* \
   libglx-nvidia0\* \
   libnvcuvid1\* \
   libnvidia-allocator1\* \
   libnvidia-api1\* \
   libnvidia-cfg1\* \
   libnvidia-eglcore\* \
   libnvidia-encode1\* \
   libnvidia-fbc1\* \
   libnvidia-glcore\* \
```

(continues on next page)

(continued from previous page)

```
   libnvidia-glvkspirv\* \
   libnvidia-gpucomp\* \
   libnvidia-ml1\* \
   libnvidia-ngx1\* \
   libnvidia-nscq\* \
   libnvidia-nvvm\* \
   libnvidia-opticalflow1\* \
   libnvidia-pkcs11-openssl3\* \
   libnvidia-present\* \
   libnvidia-ptxjitcompiler1\* \
   libnvidia-rtcore\* \
   libnvidia-sandboxutils\* \
   libnvidia-tileiras\* \
   libnvidia-vksc-core\* \
   libnvoptix1\* \
   libnvsdm\* \
   libxnvctrl\* \
   nvidia-alternative\* \
   nvidia-detect\* \
   nvidia-driver\* \
   nvidia-egl-common\* \
   nvidia-egl-icd\* \
   nvidia-fabricmanager\* \
   nvidia-imex\* \
   nvidia-kernel\* \
   nvidia-legacy\* \
   nvidia-modprobe\* \
   nvidia-open\* \
   nvidia-opencl\* \
   nvidia-persistenced\* \
   nvidia-powerd\* \
   nvidia-settings\* \
   nvidia-smi\* \
   nvidia-support\* \
   nvidia-suspend-common\* \
   nvidia-vdpau-driver\* \
   nvidia-vulkan\* \
   nvidia-xconfig\* \
   xserver-xorg-video-nvidia\*
```

If the 32 bit compatibility libraries are installed along with the main architecture on `amd64`:

```
# apt remove --autoremove --purge -V \
   libcuda1\*:i386 \
   libegl-nvidia0\*:i386 \
   libgles-nvidia1\*:i386 \
   libgles-nvidia2\*:i386 \
   libglx-nvidia0\*:i386 \
   libglx-nvidia0\*:i386 \
   libnvcuvid1\*:i386 \
   libnvidia-allocator1\*:i386 \
   libnvidia-eglcore\*:i386 \
```

(continues on next page)

(continued from previous page)

```
  libnvidia-encode1\*:i386 \
  libnvidia-fbc1\*:i386 \
  libnvidia-glcore\*:i386 \
  libnvidia-glvkspirv\*:i386 \
  libnvidia-gpucomp\*:i386 \
  libnvidia-ml1\*:i386 \
  libnvidia-nvvm4\*:i386 \
  libnvidia-opticalflow1\*:i386 \
  libnvidia-ptxjitcompiler1\*:i386 \
  libnvidia-tileiras\*:i386 \
  nvidia-driver-libs\*:i386 \
  nvidia-egl-icd\*:i386 \
  nvidia-opencl-icd\*:i386 \
  nvidia-vdpau-driver\*:i386 \
  nvidia-vulkan-icd\*:i386
```

Chapter 30. 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 for instructions.

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 31. Frequently Asked Questions

31.1. 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.

31.2. 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 Optimus laptop), you can perform a compute-only installation, which skips the graphical components of the driver.

31.3. 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.

31.4. 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
```

31.5. 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.

31.6. Third-party packages

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

Chapter 32. Notices

32.1. 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.

32.2. OpenCL

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

32.3. 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