

VMWare - Redhat 9 Install NVidia Pass through and Container Setup

Installing an NVIDIA driver in a Red Hat 9 virtual machine (VM) with GPU passthrough involves several steps, including configuring both the VMware ESXi host and the RHEL guest OS. GPU passthrough dedicates the entire physical GPU to a single VM, bypassing the hypervisor.

Reviewed October 2026 against RHEL 9.8 and NVIDIA data center driver branch 615. This guide uses an NVIDIA **Tesla M10 (Maxwell)**. NVIDIA driver branch **R580** is the last branch that supports Maxwell, Pascal and Volta GPUs. It is a Long Term Support Branch with end of life in June 2028, so a Maxwell card must stay on the **580** proprietary DKMS stream. Newer branches drop Maxwell, and from branch 615 NVIDIA ships only the open kernel modules, which need Turing or newer.

Step 1: Enable GPU passthrough on the ESXi host

1. In the host BIOS/UEFI, make sure **Intel VT-d** (AMD: **IOMMU**) is enabled and PCI mapping above 4 GB is allowed.
2. In the vSphere Client select the host, then **Configure > Hardware > PCI Devices** and open **All PCI Devices**. (ESXi Host Client: **Manage > Hardware > PCI Devices**.)
3. Select the NVIDIA GPU and click **Toggle Passthrough**.
4. If the device shows an orange icon, put the host in maintenance mode and reboot it before using the device. The passthrough status should then show as Enabled/Active.

Step 2: Configure the RHEL 9 virtual machine

1. Power off the virtual machine. **This is required before assigning PCI devices.**
2. Edit the settings of the RHEL 9 VM in the vSphere Client.
3. Under **VM Options > Boot Options** set **Firmware** to **EFI**, because GPU passthrough VMs will not boot correctly on legacy BIOS.
4. **Turn off Secure Boot (required):** with the VM still powered off, go to **Edit Settings > VM Options > Boot Options** and uncheck **Secure Boot**. With Secure Boot on, the NVIDIA kernel modules built in Step 4 are unsigned and will not load unless you sign them yourself and enroll the key (MOK). This guide does not use module signing, so Secure Boot must be off.

5. Under **Virtual Hardware**, click **Add New Device**, choose **PCI Device** and select the NVIDIA GPU you enabled for passthrough.
6. Under **Memory**, reserve all guest memory (All locked). A passthrough device needs the full memory reservation.
7. Go to **VM Options > Advanced**, click **Edit Configuration** and add these parameters:
 - `pciPassthru.use64bitMMIO = TRUE`
 - `pciPassthru.64bitMMIOSizeGB = 64`: the value depends on GPU memory. Broadcom's rule is to add up the framebuffer memory of every GPU in the VM. If the total is a power of 2, use the next power of 2. If it falls between two powers of 2, round up twice. One Tesla M10 GPU (8 GB) needs at least **16**; **64** also works.
 - `hypervisor.cpuid.v0 = FALSE`: used in the tested setup. It is not part of Broadcom's GPU passthrough KBs.
8. Save the VM settings and power on the VM.

Step 3: Prepare the OS

VMware tools: Broadcom recommends the open-vm-tools package shipped by Red Hat.

```
sudo dnf install -y open-vm-tools
```

Enable CodeReady Builder and EPEL: the NVIDIA driver RPMs depend on packages that only exist in EPEL (for example DKMS).

```
sudo subscription-manager repos --enable=codeready-builder-for-rhel-9-x86_64-rpms
sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

Install build dependencies: DKMS needs the kernel headers and development packages. On RHEL 9 NVIDIA uses `kernel-devel-matched`, which also installs the matching devel package for future kernel updates so DKMS can rebuild the module.

```
sudo dnf groupinstall "Development Tools"
sudo dnf install kernel-devel-matched kernel-headers
```

Disable the Nouveau driver (optional): the open-source Nouveau driver can conflict with the NVIDIA driver. The NVIDIA RPMs disable nouveau automatically during install (`nvidia-boot-update`). To do it by hand:

```
sudo bash -c "echo blacklist nouveau > /etc/modprobe.d/blacklist-nvidia-nouveau.conf"
sudo bash -c "echo options nouveau modeset=0 >> /etc/modprobe.d/blacklist-nvidia-nouveau.conf"
sudo dracut --force
sudo reboot
```

Step 4: Install the NVIDIA driver (580 branch, proprietary DKMS)

For Maxwell GPUs such as the Tesla M10, do **not** use `nvidia-driver:latest-dkms`, `nvidia-driver:open-dkms` or `nvidia-open`. In NVIDIA's RHEL 9 repository `latest-dkms` now resolves to the 610 branch, which no longer supports Maxwell. The open kernel modules only support Turing and newer GPUs.

Add the repository: NVIDIA's network repository for RHEL 9.

```
sudo dnf config-manager --add-repo
https://developer.download.nvidia.com/compute/cuda/repos/rhel9/x86_64/cuda-rhel9.repo
sudo dnf clean expire-cache
```

Install the driver: use the `dnf module` stream locked to the **580** branch with proprietary DKMS modules. Its default profile installs the full driver plus `kmod-nvidia-latest-dkms`.

```
sudo dnf module install nvidia-driver:580-dkms
```

Later updates stay on the 580 branch with a normal `sudo dnf update`.

Secure Boot: the DKMS-built proprietary modules are not signed by Red Hat. Red Hat-signed precompiled NVIDIA modules exist only for the open kernel modules (RHEL 10), which Maxwell GPUs cannot use. Secure Boot must be off (Step 2). Check it inside the VM before installing the driver:

```
mokutil --sb-state
```

The output should be `SecureBoot disabled`. If it says `SecureBoot enabled`, power off the VM and uncheck Secure Boot under **Edit Settings > VM Options > Boot Options**.

Reboot the VM:

```
sudo reboot
```

Step 5: Verify the installation

After the VM reboots, run `nvidia-smi`. If the install worked, it shows the GPU, which confirms that the RHEL guest is using it through passthrough:

```
$ nvidia-smi
+-----+
+-----+
```

```

| NVIDIA-SMI 580.95.05      Driver Version: 580.95.05   CUDA Version: 13.0   |
+-----+-----+-----+
| GPU  Name      Persistence-M | Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|                               | MIG M. |
+=====+
| 0 Tesla M10      Off | 00000000:XX:00.0 Off |          N/A |
| N/A   36C   P8      8W /  53W |      3MiB / 8192MiB |      0%      Default |
|                               |          N/A |
+-----+-----+-----+
+-----+
| Processes:                                         |
| GPU  GI  CI           PID  Type  Process name          GPU Memory |
|      ID  ID                                   Usage      |
+=====+
| No running processes found                        |
+-----+

```

Step 6: Set up containers

Add NVIDIA's container toolkit repository (the CUDA repository above also carries the same packages):

```

curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | \
sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo

```

Install the NVIDIA Container Toolkit:

```

sudo dnf install -y nvidia-container-toolkit

```

The older `nvidia-docker2` / `nvidia-docker` and `nvidia-container-runtime` repositories and packages are not part of NVIDIA's current install guide; use the repository above.

Docker

Configure Docker to use the NVIDIA runtime and restart it:

```

sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker

```

Test it:

```
$ docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

+-----+											
NVIDIA-SMI 580.95.05			Driver Version: 580.95.05			CUDA Version: 13.0					
+-----+											
GPU Name		Persistence-M		Bus-Id		Disp.A		Volatile Uncorr. ECC			
Fan Temp Perf		Pwr:Usage/Cap				Memory-Usage		GPU-Util		Compute M.	
						MIG M.					
=====+											
=====											
0 Tesla M10		Off		00000000:XX:00.0		Off		N/A			
N/A 36C P8		8W / 53W		3MiB / 8192MiB		0%		Default			
						N/A					
+-----+											
+-----+											
Processes:											
GPU GI CI		PID Type		Process name				GPU Memory			
ID ID						Usage					
=====											
=====											
No running processes found											
+-----+											

Podman (CDI)

For Podman, RHEL's built-in container engine, NVIDIA recommends CDI instead of `nvidia-ctk runtime configure`. Since toolkit v1.18 the `nvidia-cdi-refresh` service generates the CDI spec automatically at `/var/run/cdi/nvidia.yaml`. It runs when the toolkit or driver is installed or upgraded and at boot. List the devices:

```
nvidia-ctk cdi list
```

Regenerate the spec if needed (for example after removing a driver):

```
sudo systemctl restart nvidia-cdi-refresh.service
# or manually
sudo nvidia-ctk cdi generate --output=/var/run/cdi/nvidia.yaml
```

Test it:

```
podman run --rm --security-opt=label=disable --device=nvidia.com/gpu=all ubuntu nvidia-smi
```

Troubleshooting: driver missing after a kernel update

With `kernel-devel-matched` installed, DKMS rebuilds the module automatically after kernel updates. If `nvidia-smi` stops working after a kernel update, check which kernels the module is built for:

```
dkms status
```

Example output where the module is only built for an older kernel:

```
nvidia/580.95.05, 5.14.0-427.42.1.el9_4.x86_64, x86_64: installed
```

Make sure `kernel-devel` for the running kernel is installed, then build the module for the current kernel. Use the version shown by `dkms status`:

```
sudo dkms install nvidia/580.95.05
```

References

- [NVIDIA Container Toolkit install guide](#) and [CDI support](#)
- [Ollama Docker image](#) (container configuration)
- [NVIDIA blog: driver module streams on RHEL](#) (background)

Sources (checked October 2026)

- [Red Hat: RHEL release dates](#) (RHEL 9.8 GA 2026-05-19)
- [NVIDIA Driver Installation Guide: RHEL](#), [Recent Updates](#) (proprietary modules removed), [Advanced Options](#) (nvidia-boot-update)
- [NVIDIA supported drivers](#) and [NVIDIA blog: R580 last branch for Maxwell/Pascal/Volta](#)
- [NVIDIA Container Toolkit sample workload](#)
- [Broadcom KB 334594](#) (EFI + 64-bit MMIO sizing), [KB 312208](#) (DirectPath I/O requirements), [KB 309986](#) (Toggle Passthrough), [KB 313456](#) (open-vm-tools)

Revision #3

Created 27 September 2025 17:19:35 by Admin

Updated 11 October 2026 04:27:00 by Admin