PCI(e) Passthrough

Contents

PCI(e) passthrough is a mechanism to give a virtual machine control over a PCI device from the host. This can have some advantages over using virtualized hardware, for example lower latency, higher performance, or more features (e.g., offloading).

But, if you pass through a device to a virtual machine, you cannot use that device anymore on the host or in any other VM.

General Requirements

Since passthrough is a feature which also needs hardware support, there are some requirements to check and preparations to be done to make it work.

Hardware

Your hardware needs to support `IOMMU` (**I**/**O** **M**emory **M**anagement **U**nit) interrupt remapping, this includes the CPU and the mainboard.

Generally, Intel systems with VT-d, and AMD systems with AMD-Vi support this. But it is not guaranteed that everything will work out of the box, due to bad hardware implementation and missing or low quality drivers.

Further, server grade hardware has often better support than consumer grade hardware, but even then, many modern system can support this.

Please refer to your hardware vendor to check if they support this feature under Linux for your specific setup.

Configuration

Once you ensured that your hardware supports passthrough, you will need to do some configuration to enable PCI(e) passthrough.

IOMMU

First, you have to enable IOMMU support in your BIOS/UEFI. Usually the corresponding setting is called `IOMMU` or `VT-d`,but you should find the exact option name in the manual of your motherboard.

For Intel CPUs, you may also need to enable the IOMMU on the kernel command line for older (pre-5.15) kernels by adding:

```
intel_iommu=on
```

For AMD CPUs it should be enabled automatically.

IOMMU Passthrough Mode

If your hardware supports IOMMU passthrough mode, enabling this mode might increase performance. This is because VMs then bypass the (default) DMA translation normally performed by the hyper-visor and instead pass DMA requests directly to the hardware IOMMU. To enable these options, add:

```
iommu=pt
```

to the kernel commandline.

Kernel Modules

You have to make sure the following modules are loaded. This can be achieved by adding them to '*/etc/modules*'

```
vfio
vfio_iommu_type1
vfio_pci
vfio_virqfd
```

After changing anything modules related, you need to refresh your `initramfs`. On Proxmox VE this can be done by executing:

```
# update-initramfs -u -k all
```

Finish Configuration

Finally reboot to bring the changes into effect and check that it is indeed enabled.

```
# dmesg | grep -e DMAR -e IOMMU -e AMD-Vi
```

should display that `IOMMU`, `Directed I/O` or `Interrupt Remapping` is enabled, depending on hardware and kernel the exact message can vary.

It is also important that the device(s) you want to pass through are in a **separate** `IOMMU` group. This can be checked with:

```
# find /sys/kernel/iommu_groups/ -type l
```

It is okay if the device is in an IOMMU group together with its functions (e.g. a GPU with the HDMI Audio device) or with its root port or PCI(e) bridge.

PCI(e) slots

Some platforms handle their physical PCI(e) slots differently. So, sometimes it can help to put the card in a another PCI(e) slot, if you do not get the desired IOMMU group separation.

Unsafe interrupts

For some platforms, it may be necessary to allow unsafe interrupts. For this add the following line in a file ending with '.conf' file in **/etc/modprobe.d/**:

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

Please be aware that this option can make your system unstable.

GPU Passthrough Notes

It is not possible to display the frame buffer of the GPU via NoVNC or SPICE on the Proxmox VE web interface.

When passing through a whole GPU or a vGPU and graphic output is wanted, one has to either physically connect a monitor to the card, or configure a remote desktop software (for example, VNC or RDP) inside the guest.

If you want to use the GPU as a hardware accelerator, for example, for programs using OpenCL or CUDA, this is not required.

Host Device Passthrough

The most used variant of PCI(e) passthrough is to pass through a whole PCI(e) card, for example a GPU or a network card.

Host Configuration

In this case, the host must not use the card. There are two methods to achieve this:

- pass the device IDs to the options of the *vfio-pci* modules by adding

  ```
  options vfio-pci ids=1234:5678,4321:8765
  ```

  to a .conf file in **/etc/modprobe.d/** where 1234:5678 and 4321:8765 are the vendor and device IDs obtained by:

  ```
  # lspci -nn
  ```

- blacklist the driver completely on the host, ensuring that it is free to bind for passthrough, with

```
blacklist DRIVERNAME
```

in a .conf file in **/etc/modprobe.d/**.

For both methods you need to update the `initramfs` again and reboot after that.

Verify Configuration

To check if your changes were successful, you can use

```
# lspci -nnk
```

and check your device entry. If it says

```
Kernel driver in use: vfio-pci
```

or the *in use* line is missing entirely, the device is ready to be used for passthrough.

VM Configuration

To pass through the device you need to set the **hostpciX** option in the VM configuration, for example by executing:

```
# qm set VMID -hostpci0 00:02.0
```

If your device has multiple functions (e.g., '`00:02.0`' and '`00:02.1`'), you can pass them through all together with the shortened syntax ``00:02`. *This is equivalent with checking the ``All Functions`* checkbox in the web-interface.

There are some options to which may be necessary, depending on the device and guest OS:

- **x-vga=on|off** marks the PCI(e) device as the primary GPU of the VM. With this enabled the **vga** configuration option will be ignored.
- **pcie=on|off** tells Proxmox VE to use a PCIe or PCI port. Some guests/device combination require PCIe rather than PCI. PCIe is only available for *q35* machine types.
- **rombar=on|off** makes the firmware ROM visible for the guest. Default is on. Some PCI(e) devices need this disabled.
- **romfile=<path>**, is an optional path to a ROM file for the device to use. This is a relative path under **/usr/share/kvm/**.

Example

An example of PCIe passthrough with a GPU set to primary:

```
# qm set VMID -hostpci0 02:00,pcie=on,x-vga=on
```

PCI ID overrides

You can override the PCI vendor ID, device ID, and subsystem IDs that will be seen by the guest. This is useful if your device is a variant with an ID that your guest's drivers don't recognize, but you want to force those drivers to be loaded anyway (e.g. if you know your device shares the same chipset as a supported variant).

The available options are `vendor-id`, `device-id`, `sub-vendor-id`, and `sub-device-id`. You can set any or all of these to override your device's default IDs.

For example:

```
# qm set VMID -hostpci0 02:00,device-id=0x10f6,sub-vendor-id=0x0000
```

Other considerations

When passing through a GPU, the best compatibility is reached when using *q35* as machine type, *OVMF* (*EFI* for VMs) instead of SeaBIOS and PCIe instead of PCI. Note that if you want to use *OVMF* for GPU passthrough, the GPU needs to have an EFI capable ROM, otherwise use SeaBIOS instead.

SR-IOV

Another variant for passing through PCI(e) devices, is to use the hardware virtualization features of your devices, if available.

SR-IOV (**S**ingle-**R**oot **I**nput/**O**utput **V**irtualization) enables a single device to provide multiple *VF* (**V**irtual **F**unctions) to the system. Each of those *VF* can be used in a different VM, with full hardware features and also better performance and lower latency than software virtualized devices.

Currently, the most common use case for this are NICs (**N**etwork **I**nterface **C**ard) with SR-IOV support, which can provide multiple VFs per physical port. This allows using features such as checksum offloading, etc. to be used inside a VM, reducing the (host) CPU overhead.

Host Configuration

Generally, there are two methods for enabling virtual functions on a device.

- sometimes there is an option for the driver module e.g. for some Intel drivers

  ```
  max_vfs=4
  ```

  which could be put file with *.conf* ending under **/etc/modprobe.d/**. (Do not forget to update your initramfs after that)

  Please refer to your driver module documentation for the exact parameters and options.

- The second, more generic, approach is using the `sysfs`. If a device and driver supports this you can change the number of VFs on the fly. For example, to setup 4 VFs on device 0000:01:00.0 execute:

```
# echo 4 > /sys/bus/pci/devices/0000:01:00.0/sriov_numvfs
```

  To make this change persistent you can use the 'sysfsutils` Debian package. After installation configure it via **/etc/sysfs.conf** or a `FILE.conf' in **/etc/sysfs.d/**.

VM Configuration

After creating VFs, you should see them as separate PCI(e) devices when outputting them with `lspci`. Get their ID and pass them through like a normal PCI(e) device.

Other considerations

For this feature, platform support is especially important. It may be necessary to enable this feature in the BIOS/EFI first, or to use a specific PCI(e) port for it to work. In doubt, consult the manual of the platform or contact its vendor.

Mediated Devices (vGPU, GVT-g)

Mediated devices are another method to reuse features and performance from physical hardware for virtualized hardware. These are found most common in virtualized GPU setups such as Intel's GVT-g and NVIDIA's vGPUs used in their GRID technology.

With this, a physical Card is able to create virtual cards, similar to SR-IOV. The difference is that mediated devices do not appear as PCI(e) devices in the host, and are such only suited for using in virtual machines.

Host Configuration

In general your card's driver must support that feature, otherwise it will not work. So please refer to your vendor for compatible drivers and how to configure them.

Intel's drivers for GVT-g are integrated in the Kernel and should work with 5th, 6th and 7th generation Intel Core Processors, as well as E3 v4, E3 v5 and E3 v6 Xeon Processors.

To enable it for Intel Graphics, you have to make sure to load the module *kvmgt* (for example via `/etc/modules`) and to enable it on the Kernel commandline and add the following parameter:

```
i915.enable_gvt=1
```

After that remember to update the `initramfs`, and reboot your host.

VM Configuration

To use a mediated device, simply specify the `mdev` property on a `hostpciX` VM configuration option.

You can get the supported devices via the *sysfs*. For example, to list the supported types for the device *0000:00:02.0* you would simply execute:

```
# ls /sys/bus/pci/devices/0000:00:02.0/mdev_supported_types
```

Each entry is a directory which contains the following important files:

- *available_instances* contains the amount of still available instances of this type, each *mdev* use in a VM reduces this.
- *description* contains a short description about the capabilities of the type
- *create* is the endpoint to create such a device, Proxmox VE does this automatically for you, if a *hostpciX* option with `mdev` is configured.

Example configuration with an `Intel GVT-g vGPU` (`Intel Skylake 6700k`):

```
# qm set VMID -hostpci0 00:02.0,mdev=i915-GVTg_V5_4
```

With this set, Proxmox VE automatically creates such a device on VM start, and cleans it up again when the VM stops.

Use in Clusters

It is also possible to map devices on a cluster level, so that they can be properly used with HA and hardware changes are detected and non root users can configure them. See Resource Mapping for details on that.

See Also

- PCI Passthrough Examples