

# KVM Virtual Machines

- [Bridge Zero Copy Transmit](#)
- [PCI Passthrough](#)
- [QEMU Device Properties](#)
- [SR-IOV](#)
- [Mount QCOW2](#)
- [Direct Boot Kernel](#)
- [Serial Only](#)
- [EFI](#)

# Bridge Zero Copy Transmit

“ Zero copy transmit mode is effective on large packet sizes. It typically reduces the host CPU overhead by up to 15% when transmitting large packets between a guest network and an external network, without affecting throughput.

Source: [Red Hat - Network Tuning Techniques](#)

```
# /etc/modprobe.d/vhost-net.conf  
+ options vhost_net experimental_zcopytx=1
```

# PCI Passthrough

## Ensure IOMMU Is Activated

“ First step of this process is to make sure that your hardware is even capable of this type of virtualization. You need to have a motherboard, CPU, and BIOS that has an IOMMU controller and supports Intel-VT-x and Intel-VT-d or AMD-v and AMD-vi. Some motherboards use different terminology for these, for example they may list AMD-v as SVM and AMD-vi as IOMMU controller.

## Ensure Kernel Modules

### Debian

```
# /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
+ vfio_pci
+ vfio
+ vfio_iommu_type1
+ vfio_virqfd
```

## Bind `vfio-pci` Driver to Devices

Now you can bind the `vfio-pci` driver to your devices at startup so they can be passed through to a VM. There are two ways of doing this, the first way is quick and easy but forces you to blacklist an entire driver which would stop you from being able to use that driver for another device that you aren't passing through. The second way is a little more complicated but allows you to target individual devices without blacklisting an entire driver.

### 1) Blacklist Drivers

By running `lspci -knn` you can easily find out which drivers are being used for a device so you know what driver to blacklist in addition to their `<vendor>:<device>` identifier. Armed with both of these we can blacklist the drivers we don't want being used and let the `vfio-pci` driver know which device(s) to bind to.

Below is an example of blacklisting the driver `i915` (Intel iGPU driver) so I can pass through my iGPU to a virtual machine. The driver is blacklisted so it won't load and the device identified by `<vendor>:<device>` is added as a parameter to the `vfio-pci` driver so it knows which device to bind with.

```
# /etc/modprobe.d/blacklist.conf
+ blacklist i915
```

```
# /etc/modprobe.d/vfio.conf
+ options vfio-pci ids=8086:3e92 disable_vga=1
```

## 2) Alias Devices

Using `lspci -knn` it is easy to find a devices [B/D/F identifier](#) and its `<vendor>:<device>` identifier. Then we can find its *modalias* by running `cat /sys/bus/pci/devices/<B/D/F>/modalias`. Armed with both of these we can let the `vfio-pci` module know which devices to bind to.

```
# /etc/modprobe.d/vfio.conf
+ # Intel UHD 630 (8086:3e92)
+ alias pci:v00008086d00003E92sv00001458sd0000D000bc03sc80i00 vfio-pci
+
+ options vfio-pci ids=8086:3e92 disable_vga=1
```

# Rebuild `initramfs`

## Debian

```
update-initramfs -u
```

# Update Bootloader

## Update Kernel Parameters

## Grub2

```
# /etc/default/grub
- GRUB_CMDLINE_LINUX_DEFAULT="quiet"
+ GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=igfx_off iommu=pt video=efifb:off"
```

## Systemd

```
# /etc/kernel/cmdline
- root=ZFS=rpool/ROOT/pve-1 boot=zfs
+ root=ZFS=rpool/ROOT/pve-1 boot=zfs intel_iommu=igfx_off iommu=pt video=efifb:off
```

## Rebuild Bootloader Options

### Grub

```
update-grub
```

### systemd-boot

```
bootctl update
```

### Proxmox

```
pve-efiboot-tool refresh
```

# QEMU Device Properties

## Example: Rename Device

## Example: Move MSI-X

“ The QEMU vfio-pci device option is `x-msix-relocation=` which allows specifying the bar to use for the MSI-X tables, ex. `bar0...bar5`. Since this device uses a 64bit `bar0`, we can either extend that BAR or choose another, excluding `bar1`, which is consumed by the upper half of `bar0`.

To set these properties you can edit the VM configuration and add an `args` parameter.

```
args: -set device.hostpci1.x-msix-relocation=bar2
```

# SR-IOV

## Ensure IOMMU Is Activated

“ First step of this process is to make sure that your hardware is even capable of this type of virtualization. You need to have a motherboard, CPU, and BIOS that has an IOMMU controller and supports Intel-VT-x and Intel-VT-d or AMD-v and AMD-vi. Some motherboards use different terminology for these, for example they may list AMD-v as SVM and AMD-vi as IOMMU controller.

## Update Bootloader

### Update Kernel Parameters

**\*\*NOTE\*\*** Be sure to replace `intel_iommu=on` with `amd_iommu=on` if you're running on AMD instead of Intel.

### Grub2

```
# /etc/default/grub
- GRUB_CMDLINE_LINUX_DEFAULT="quiet"
+ GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt"
```

### Systemd

```
# /etc/kernel/cmdline
- root=ZFS=rpool/ROOT/pve-1 boot=zfs
+ root=ZFS=rpool/ROOT/pve-1 boot=zfs intel_iommu=on iommu=pt
```

## Rebuild Bootloader Options

### Grub

```
update-grub
```

## systemd-boot

```
bootctl update
```

## Proxmox

```
pve-efiboot-tool refresh
```

# Enable Virtual Functions

Find the link name you want to add virtual function to using `ip link`. In this scenario we're going to say we want to add 4 virtual functions to link `eth2`. You can find the maximum number of virtual function possible by reading the `sriov_totalvfs` from sysfs...

```
cat /sys/class/net/enp10s0f0/device/sriov_totalvfs  
7
```

To enable virtual functions you just `echo` the number you want to `sriov_numvfs` in sysfs...

```
echo 4 > /sys/class/net/enp10s0f0/device/sriov_numvfs
```

# Make Persistent

Sysfs is a virtual file system in Linux kernel 2.5+ that provides a tree of system devices. This package provides the program 'systool' to query it: it can list devices by bus, class, and topology.

In addition this package ships a configuration file `/etc/sysfs.conf` which allows one to conveniently set sysfs attributes at system bootup (in the init script `etc/init.d/sysfsutils`).

```
apt install sysfsutils
```

# Configure sysfsutils

To make these changes persistent, you need to update `/etc/sysfs.conf` so that it gets set on startup.

```
echo "class/net/eth2/device/sriov_numvfs = 4" >> /etc/sysfs.conf
```



# Mount QCOW2

Load Kernel module

```
modprobe nbd
```

Connect the image to NBD (Network Block Device) device and then mount that device/partition

```
qemu-nbd --connect=/dev/nbd0 /var/lib/vz/images/100/vm-100-disk-1.qcow2  
mount /dev/nbd0p1 /mnt/somepoint/
```

When done unmount, disconnect, and if necessary unload the Kernel module.

```
umount /mnt/somepoint/  
qemu-nbd --disconnect /dev/nbd0  
rmmod nbd
```

# Direct Boot Kernel

Provide path to Kernel and optionally initrd

```
qemu-system-aarch64 ... -kernel /boot/vmlinuz-6.9.0-rc6+ -initrd /boot/initrd.img-6.9.0-rc6+
```

# Serial Only

## AMD64

```
qemu-system-x86_64 ... -nographic -append "root=/dev/vda rw console=ttyS0" -hda rootfs.img
```

## ARM64

```
qemu-system-aarch64 ... -nographic -append "root=/dev/vda rw console=ttyAMA0" -hda rootfs.img
```

Some emulated consoles will need a speed appended like console=ttyAMA0,115200

# EFI

To use OVMF/AAVMF for EFI add these parameters to `qemu-system-*`. Normally you can find `OVMF_CODE.fd` and `OVMF_VARS.fd` (or variants of them) in `/usr/share`

```
-drive if=pflash,format=raw,readonly,file=OVMF_CODE-pure-efi.fd  
-drive if=pflash,format=raw,file=OVMF_VARS.fd
```