blog.vararu.org

Fieldnotes for running Proxmox with Windows and macOS guests

30 April 2020

These are my notes and a high-level guide for how I run Windows and macOS VMs in Proxmox.

What is this and why even bother?

Proxmox VE is a headless Ubuntu-based distro. It comes with QEMU preinstalled, which allows you to run Kernel-based Virtual Machines, or KVMs. KVMs run on a type-1 hypervisor, also known as a native or bare-metal hypervisor. Proxmox also comes with a web interface for configuring and managing your virtual machines.

KVMs are interesting because of the fact that they allow near-native performance. In my experience, the CPU/Memory/Disk overhead is imperceivable in normal use, and only visible in benchmarks where it's anywhere from 0-5%. I would normally frown on using a virtual machine for day to day computing, but when the cost is so low, it starts to make sense.

QEMU also allows PCI passthrough of native hardware to a VM, such as graphics cards or sound cards. This allows for true native performance in things like games or audio editing software. r/VFIO is a subreddit dedicated to discussing gaming on virtual machines.

My use-case is using macOS for productivity, while running Windows at the same time for gaming. This isn't possible on my current setup, as I still need an extra graphics card, but I am evaluating this setup for a while longer before I commit to buying more hardware.

Preparation

- Make sure VT-x and VT-d are enabled in your BIOS
- Ensure you have a minimum of 2 drives; one for Proxmox, one for VMs
- Ensure you have a working computer to connect to the Proxmox web interface, or via SSH, or to generally do work on while you spend the next days/weeks fiddling with your new VM obsession

Installing Proxmox

Download a copy of ProxmoxVE from https://www.proxmox.com/en/downloads.

To create a bootable drive on macOS (see also the Proxmox wiki page for this):

```
$ hdiutil convert -format UDRW -o proxmox-ve_*.dmg proxmox-ve_*.iso
$ diskutil list
$ diskutil unmountDisk /dev/diskX
$ sudo dd if=proxmox-ve_*.dmg of=/dev/rdiskX bs=1m
```

Once that's done, plug it into your computer, and boot off of the drive. The setup process is straightforward:

- Set "Target Harddisk", I used my NVME SSD

- Set country, UK in my case
- Set up a root password
- Set up networking, for me: hostname to `proxmox.server`, IP Address to `10.0.0.5`, Netmask `255.255.255.0`, Gateway and DNS to `10.0.0.1`

Rest of the installation should carry on by itself. Proxmox will automatically set itself up to be first in the boot order.

At this stage, you can log in using `root/password` from the regular keyboard connected to the machine. Proxmox is Ubuntu-based, but headless and without many programs preinstalled. I like to install a few things I use a lot:

```
$ apt update
$ apt upgrade -y
$ apt install vim tmux htop
```

The Proxmox VE web interface will be running in the background. Open https://10.0.0.5:8006/ on a different computer and tell your browser to ignore the bad certificate warning.

`root/password` gets you into the web interface. The “No valid subscription” message is only relevant for Enterprise use, so you can close it and ignore it.

Another way of accessing your server is via SSH. I copy my SSH key via `ssh-copy-id` to make this easier:

```
$ ssh-copy-id root@10.0.0.5
$ ssh root@10.0.0.5
```

I use this a lot to transfer images (via `scp`) and make OS configuration changes.

I configure the following for both my Windows and my macOS VMs:

```
# enable IOMMU groups
$ vim /etc/default/grub
# set GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on"
# you'll need to set amd_iommu=on for AMD CPUs
$ update-grub

# enable VFIO kernel modules
$ vim /etc/modules
vfio
vfio_iommu_type1
vfio_pci
vfio_virqfd
$ echo "options vfio_iommu_type1 allow_unsafe_interrupts=1" >
/etc/modprobe.d/iommu_unsafe_interrupts.conf

# ignore errors that commonly appear in STDOUT when running Windows/macOS VMs
$ vim /etc/modprobe.d/kvm.conf
options kvm ignore_msrs=Y
options kvm report_ignored_msrs=0
```

```
# blacklist init-ing graphics cards from proxmox so they can be passed through
$ vim /etc/modprobe.d/blacklist.conf
blacklist nouveau
blacklist nvidia
blacklist nvidiafb
blacklist radeon
# optional, I don't use these yet:
# blacklist amdgpu
# blacklist snd_hda_codec_hdmi
# blacklist snd_hda_intel
# blacklist snd_hda_codec
# blacklist snd_hda_core

# find your graphics cards
$ lspci -v
# find the one you want, e.g. my Radeon is 01:00
# inspect that
$ lspci -n -s 01:00
# grab the graphics card IDs to use in the next step
$ echo "options vfio-pci ids=1002:67df,1002:aaf0 disable_vga=1" >
/etc/modprobe.d/vfio.conf
$ update-initramfs -u

# enable nested virtualisation, such as docker for mac, intel only
modprobe -r kvm_intel
modprobe kvm_intel nested=1

$ reset
$ reboot now
```

Reboot the machine to enable IOMMUs and unload your graphics card(s) from Proxmox.

Through the web interface, go to Datacenter > proxmox > Disks > ZFS. Create a new ZFS, I named mine `sandisk` using `/dev/sda`. Note that the drive needs to be completely clean, which can be accomplished via SSH using `fdisk /dev/sda`.

Creating the Windows VM

Windows is a lot easier to get up and running than macOS, so it's a better starting point to learn the basics.

Most of what follows is based on this guide about setting up GPU passthrough in Windows r/homelab.

Grab a Windows 10 ISO (from Microsoft) and the VirtIO drivers from Fedora.

Transfer them using `scp`:

```
$ scp * root@10.0.0.5:/var/lib/vz/template/iso
```

Click "Create VM" in the top right of the Proxmox web interface. Customise as such (anything not mentioned is the default):

- Tick "advanced" at the bottom
- General > Name: I chose "win10"
- OS > ISO image: Win10.iso, type: Microsoft Windows, version: 10/2016/2019
- System > graphics card: none (because I use PCI passthrough. You can leave as Default to try VNC first), BIOS: OVMF, machine: q35, add EFI Disk to `sandisk`
- Hard Disk > SATA 0, storage: sandisk, size: 64, cache: Write back (unsafe), tick discard and SSD emulation (for TRIM)
- CPU > cores: 8
- Memory > 20480, disable ballooning (I don't use this feature, but you might want to for your Windows VM)
- Network > vmbr0, model: VirtIO (paravirtualized)

Once done, add more hardware:

- CD/DVD drive on IDE 0 with `virtio-win.iso`
- Passthrough USB Keyboard and Mouse (I untick USB 3.0 to make it consistent with the macOS setup)
- Passthrough PCI sound card (tick nothing) and graphics card (tick rombar and pci-express)

Edit the virtual machine configuration and add the following:

```
$ vim /etc/pve/qemu-server/100.conf
args: -cpu 'host,+kvm_pv_unhalt,+kvm_pv_eoi,hv_vendor_id=NV43FIX,kvm=off'
cpu: host,hidden=1,flags=+pcid
```

Start the VM. With my configuration, I boot it directly onto the graphics card, using a real mouse and keyboard, but you can use VNC in the beginning to make things easier.

At this point, follow your usual Windows setup. Crucially, when the drive selection screen comes up, click on "Load driver." Install the VirtIO Network Adapters from NetKVM > w10 > amd64. This will allow you to connect to the internet later on in the setup process.

Once Windows is installed, remove the IDE CD drives.

At this point, Windows activation failed for me because it detected a new motherboard. I managed to activate it by clicking on "Change product key" and re-entering my Windows 7 Pro key that I used to activate Windows 10 a few years ago.

Everything else should work more or less out of the box. The only drivers I had to install are the Radeon ones for my graphics card.

Of note: Hibernating the virtual machine doesn't work because of the passthrough PCI graphics card:

```
VM 100 qmp command 'savevm-end' failed - VM snapshot not started
TASK ERROR: VM 100 qmp command 'savevm-start' failed - State blocked by non-migratable device '0000:00:1c.0:00.0/vfio-pci'
```

Creating the macOS VM

macOS is more challenging than Windows.

Most of what follows is based on Nick Sherlock's excellent guide

Use fetch-macOS.py to create a bootable installer. On a Mac:

```
$ ./fetch-macOS.py
$ hdiutil convert BaseSystem.dmg -format RdWr -o Catalina-installer.iso
$ mv Catalina-installer.iso.img Catalina-installer.iso
```

Next, download the latest .iso.gz of Nick's OpenCore build for Catalina on Proxmox.

Transfer them using `scp`:

```
$ scp * root@10.0.0.5:/var/lib/vz/template/iso
```

Click "Create VM" in the top right of the Proxmox web interface. Customise as such (anything not mentioned is the default):

- General > Name: I chose "catalina"
- OS > ISO image: OpenCore.iso, type: Other
- System > graphics card: `VMWare compatible` for VNC, BIOS: OVMF, machine: q35, add EFI Disk to `sandisk`
- Hard Disk > SATA 0, storage: sandisk, size: 256, cache: Write back (unsafe), tick discard and SSD emulation (for TRIM)
- CPU > cores: 8
- Memory > 20480, disable ballooning (no drivers for macOS)
- Network > vmbr0, model: VMWare vmxnet3

Once done, add more hardware:

- CD/DVD drive on IDE 0 with `Catalina-installer.iso`
- Passthrough USB Keyboard and Mouse (untick USB 3.0 as they won't work in macOS otherwise)

You'll need the macOS OSK for the next step. You can fetch this from a real Mac by reading from the SMC. Alternatively, it's present in the kholia/OSX-KVM GitHub repository and widely available on the internet. You can also probably extract it from this string which is in a popular encoding that only uses 64 characters:

```
b3VyaGFyZHdvcmtieXRoZXNld29yZHNndWFyZGVkcGxlYXNlZG9udHN0ZWFsKGMpQXBwbGVDb21wdXRlckluYwo=
```

Edit the virtual machine configuration and add the following:

```
$ vim /etc/pve/qemu-server/101.conf
args: -device isa-applesmc,osk="THE-OSK-YOU-EXTRACTED-GOES-HERE" -smbios type=2 -device usb-kbd,bus=ehci.0,port=2 -cpu host,kvm=on,vendor=GenuineIntel,+kvm_pv_unhalt,+kvm_pv_eoi,+hypervisor,+invtsc,vmx,rdtscp
```

Note the `-cpu` argument is for Intel CPUs only.

Replace `media=cdrom` with `cache=unsafe` for the two IDE drives.

Go to Options and set the boot order to prioritise IDE 2, where OpenCore.iso is.

Start the VM. With my configuration, I boot it directly onto the graphics card, using a real mouse and keyboard, but you can use VNC in the beginning to make things easier.

Boot into macOS Base System. Run Disk Utility, format the "External" drive as unencrypted APFS. Then "Reinstall macOS."

Choose "macOS Installer" on subsequent boots until it finishes. Don't sign in with your Apple ID just yet, as you'll probably want to update the SMBIOS before this.

Once in macOS, override the EFI with the OpenCore one:

```
$ diskutil list
# use this to replace the target drive accordingly in the next step
$ sudo dd if=/dev/disk1s1 of=/dev/disk2s1
```

Then remove the two CD drives, and change the boot order to prioritise the SATA 0 drive.

At this stage, I attach my video card via PCI-E passthrough. My RX 570 works out of the box in macOS with hardware acceleration, but works better with Lilu and Whatevergreen.

To edit the OpenCore configuration:

```
$ sudo mkdir /Volumes/EFI
$ sudo mount -t msdos /dev/disk0s1 /Volumes/EFI
$ vim /Volumes/EFI/OC/config.plist
```

Check out the Dortania OpenCore Desktop Guide for more tips and tricks: https://dortania.github.io/OpenCore-Desktop-Guide/

My hardware

- MOTHERBOARD: Asus Z170-A
- CPU: Intel i7 7700K 4c/8t
- RAM: Kingston Hyper-X DDR4 24GB 2133MHz
- ETHERNET: Built-in Intel 1000E
- AUDIO: Built-in Realtek ALC892
- GPU: Built-in Intel HD630
- GPU: AMD Radeon RX 570
- WIRELESS: Gigabyte GC-WB867D-I (unused)
- STORAGE: Samsung 960 EVO 250GB SSD
- STORAGE: Sandisk 1TB SSD
- STORAGE: Western Digital 1TB HDD

More hardware I should buy:

- AUDIO: Something like a Focusrite Scarlett Solo to pass to macOS as my on-board audio does not work (Focusrite might not be compatible with Catalina, need to check this)
- GPU: The RX 570 works great in macOS, so I should probably get something for Windows, something like a 5700 XT or 2060/2070. I had a 1080 Ti before, which I sold because I didn't use it enough in demanding games, so I probably don't need the strongest card available
- CPU: Probably an AMD Ryzen 3900X (12c/24t) or 3950X (16c/32t), assuming you can run this setup on them (some conflicting info)
- MOBO: Something to match the new CPU, preferably with 3 or so M.2 slots for more SSDs for my OS drives

Important folders in proxmox

- `/etc/pve/qemu-server`: contains the .conf files for your VMs, which you have to manually edit sometimes because the Proxmox interface doesn't include every possible QEMU option and argument
- `/var/lib/vz/template/iso`: Proxmox creates this folder out of the box which you can use for disk images such as the Windows ISO, Clover/OpenCore ISO, the macOS ISO, and so on

Resources used

This guide is mostly written for myself to help me re-trace my steps. These are some of the resources I used:

- https://www.reddit.com/r/homelab/comments/b5xpua/the_ultimate_beginners_guide_to_gpu_passthrough/
- https://www.nicksherlock.com/2020/04/installing-macos-catalina-on-proxmox-with-opencore/
- https://github.com/kholia/OSX-KVM
- https://github.com/Leoyzen/KVM-Opencore
- https://dortania.github.io/OpenCore-Desktop-Guide/

Versions used

- Proxmox VE 6.1.2
- Windows 10 1909
- macOS Catalina 10.15.4

Comments (0)

Leave a comment via GitHub.

Written by Theodor Vararu. Source code on GitHub. RSS.