ChatGPT

Requisitos de hardware e BIOS/UEFI

Para GPU passthrough é preciso hardware compatível. O processador e placa-mãe devem suportar IOMMU (Intel VT-d ou AMD-Vi) [1] [2]. Na BIOS/UEFI ative **VT-d** (Intel) ou **AMD-Vi/SVM** (AMD) e, se disponível, escolha UEFI (OVMF) em vez de legacy BIOS. É recomendável ter ao menos duas GPUs (por exemplo, uma integrada para o host e outra dedicada para a VM) ou uma placa primária para o host e passar a GPU secundária para o guest [1] [3]. Verifique se a CPU tem VT-x/VMX ou SVM com `grep vmx /proc/cpuinfo` (Intel) ou `grep svm /proc/cpuinfo` (AMD) [4]. Após ativar na BIOS, confirme no Linux com `dmesg | grep -i -e DMAR -e IOMMU` – deve aparecer algo como "DMAR: Intel VT for Directed I/O" ou grupos IOMMU listando o dispositivo [5] [4]. Se tudo estiver ok, o kernel reconhecerá o IOMMU e organizará dispositivos em grupos de IOMMU.

- CPU com VT-d/AMD-Vi habilitado na BIOS/UEFI [1] [2].
- Placa-mãe UEFI com suporte a IOMMU e OVMF (UEFI para VM).
- GPU(s) dedicadas; geralmente usam-se duas: uma para host, outra para passar à VM.
- Monitores/entradas múltiplas ou iGPU+GPU para evitar problemas de *Error 43* (ver abaixo) [6].

Configuração do host (Ubuntu / Pop!_OS)

1. **Pacotes básicos**: Instale o QEMU, libvirt e virt-manager (GUI) pelo APT. Exemplo no Ubuntu:

   ```
   sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients virt-manager ovmf
   ```

   No Pop!_OS a instalação é semelhante. Reinicie o sistema após a instalação.

2. **Ativar IOMMU no kernel**: Edite `/etc/default/grub` (Ubuntu) ou use **kernelstub** (Pop!_OS) para adicionar as opções de inicialização:

3. Em Ubuntu, em `/etc/default/grub` adicione em `GRUB_CMDLINE_LINUX_DEFAULT`:

   - Para Intel: `intel_iommu=on kvm.ignore_msrs=1`
   - Para AMD: `amd_iommu=on kvm.ignore_msrs=1`
     Então rode `sudo update-grub` e reinicie [7] [5]. Essas opções ativam o IOMMU e evitam travamentos de MSR do Hyper-V para guest Windows.

4. No Pop!_OS (que usa *systemd-boot*), use o `kernelstub`:

   ```
   sudo kernelstub -a "intel_iommu=on kvm.ignore_msrs=1"
   ```

   (ou substitua `intel_iommu` por `amd_iommu` conforme o caso) [8] [9]. Isso injeta os parâmetros no carregador.

Após reiniciar, verifique com `dmesg | grep -i -e DMAR -e IOMMU` que o IOMMU foi habilitado e que cada dispositivo PCI recebeu um "iommu group" [5] [4]. Exemplo de saída esperada:

```
[    4.598150] pci 0000:41:00.0: Adding to iommu group 66
[    4.702729] perf/amd_iommu: Detected AMD IOMMU #0 ...
```

Se não aparecer, reveja BIOS ou parâmetros de kernel.

1. **Módulos VFIO**: Certifique-se que os módulos *vfio* (VFIO) sejam carregados no boot. Em Ubuntu 20.04+ eles podem estar embutidos, mas para garantir, adicione em `/etc/initramfs-tools/modules`:

   ```
   vfio
   vfio_iommu_type1
   vfio_pci
   vfio_virqfd
   ```

   Depois rode `sudo update-initramfs -u` e reinicie [10]. Isso prepara o sistema para usar VFIO.

2. **Isolar a GPU do host**: Antes de passar a GPU para a VM, impeça o host de usá-la. Usualmente isso envolve *blacklist* dos drivers do host para essa GPU. Por exemplo, se for GPU NVIDIA comum, crie `/etc/modprobe.d/vfio.conf` contendo:

   ```
   blacklist nouveau
   blacklist nvidia
   blacklist nvidia_drm
   blacklist snd_hda_intel
   options vfio-pci ids=<VendorID:DeviceID>,<VendorID:AudioID>
   ```

   Substitua `<VendorID:DeviceID>` pelos IDs PCI da sua GPU e do seu áudio onboard (que podem ser obtidos com `lspci -nnk | grep -A2 VGA`) [11]. Em seguida atualize initramfs (`sudo update-initramfs -u`) e reinicie. O `lspci -nnk` deve mostrar agora que esses dispositivos NÃO estão atribuídos aos drivers do host.

   Exemplo:

   ```
   /etc/modprobe.d/vfio.conf:
   blacklist nouveau
   blacklist snd_hda_intel
   options vfio-pci ids=10de:1a81,10de:10f0
   ```

   Após `update-initramfs` e reboot, `lspci -nnk` não deve mais listar “driver in use: nouveau” nem “snd_hda_intel” para a GPU e seu áudio [11].

1. **Grupos IOMMU e ACS**: Verifique os grupos de IOMMU em `/sys/kernel/iommu_groups/`. Cada GPU (VGA e áudio) idealmente está em um grupo separado. Se ambos (GPU+áudio) aparecerem no mesmo grupo, será necessário passar ambos juntos. Em placas *mirrored* ou *idênticas*, às vezes há dependências de ACS. Em casos onde vários dispositivos ficam juntos em

um só grupo, pode-se usar a opção de kernel `pcie_acs_override=downstream` (discuta riscos antes) [12]. Adicione isso em `GRUB_CMDLINE_LINUX_DEFAULT` junto com `iommu=1` e atualize o GRUB [12]. *Nota:* essa solução é um *workaround* e pode comprometer isolamento de PCI.

Configuração do QEMU/KVM

Com o host preparado, crie ou edite a VM Windows com GPU passthrough. Pode usar **virt-manager** (GUI) ou `virsh/virsh-edit`:

- **Máquina virtual**: escolha *arquitetura x86_64*, chipset **Q35** e firmware **UEFI (OVMF)** para compatibilidade com GPUs modernas [13]. Instale Windows normalmente em disco dedicado ou imagem de disco bruto. Desative quaisquer drivers gráficos virtuais (ex: não habilite "cirrus" ou "VGA"). Ative virtualização de CPU com modo *host-passthrough* no XML (ou na GUI, use CPU em modo *host-passthrough*) para melhor desempenho.
- **Dispositivos PCI (GPU)**: na configuração (XML/libvirt), adicione a GPU como dispositivo PCI do tipo hostdev. Exemplo de snippet XML libvirt:

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x41' slot='0x00' function='0x0'/>
  </source>
</hostdev>
```

Aqui *bus=0x41 slot=0x00 func=0x0* deve corresponder ao **B:D:S** da GPU (use `lspci` para descobrir) [14]. Adicione também a entrada PCI de áudio da GPU (por ex. *slot 0x01*) no mesmo grupo IOMMU. Isso passa a GPU física inteira para a VM. Em virt-manager, isso equivale a "Add Hardware → PCI Host Device" e selecionar as linhas correspondentes à GPU e à parte de áudio [15] [16].

- **Linha de comando QEMU**: se preferir CLI, passe o dispositivo com `-device vfio-pci,host=XX:XX.X,multifunction=on,x-vga=on`. Exemplo para um dispositivo em 05:00.0 e 05:00.1 (GPU e áudio):

```
-device vfio-pci,host=05:00.0,bus=1,addr=00.0,multifunction=on,x-vga=on
\
-device vfio-pci,host=05:00.1,bus=1,addr=00.1
```

Isso atribui esses barramentos PCI diretamente à VM [17].

- **Drivers no Guest**: instale drivers proprietários dentro do Windows na VM (NVIDIA ou AMD oficiais). Verifique no Gerenciador de Dispositivos do Windows se a GPU aparece corretamente.

Isolamento da GPU

Para garantir que o host não use a GPU passada:
- **Blacklisting**: como visto, coloque *blacklists* (nouveau, nvidia, radeon, etc.) em `/etc/modprobe.d/` [11].
- **VFIO**: verifique em `lspci -nnk` que os dispositivos da GPU não tenham "Kernel driver in use" (se em branco ou 'vfio-pci', está correto) [18].
- **Switch de vídeo**: se possível, use apenas iGPU ou outra GPU para saída de vídeo do host. Em notebooks híbridos, escolha o modo *integrado* ou *hybrid* via utilitários do sistema (Pop!_OS tem suporte a switching) [19]. Isso evita que o Windows veja a GPU incorretamente e acione o erro 43.

Alternativas a QEMU/KVM

Além do KVM, há outras plataformas de virtualização:

- **VirtualBox**: possui aceleração gráfica 3D via GPU virtualizada, mas **não suporta PCI Passthrough** de forma confiável em hosts Linux modernos (a funcionalidade era experimental e foi removida em versões recentes) [20]. Não é recomendado para passthrough de GPU.
- **VMware Workstation/Player**: também **não suportam passthrough de GPU** físico. O guest pode usar um "SVGA" 3D virtualizado, mas não consegue acesso dedicado à GPU do host [21].
- **VMware ESXi/VSphere**: sim suporta passthrough de GPU (VT-d), mas é um hypervisor bare-metal voltado a servidores, não para desktops Linux comuns.
- **Proxmox VE**: é uma distribuição baseada em KVM/QEMU com GUI web; os passos são semelhantes ao Ubuntu/KVM, mas pode facilitar a configuração (mesmo motor KVM).
- **Xen**: hypervisor alternativo que suporta passthrough, mas exige configuração diferente e não é comum em desktops Linux.

Em geral, **KVM/QEMU** é a opção recomendada em Linux desktop por ser livre, bem suportado e de alto desempenho. VMware ou soluções corporativas são mais complexas ou comerciais, e VirtualBox não oferece passthrough real.

Dificuldades comuns e soluções

- **Dispositivos no mesmo IOMMU group**: se a GPU e outro dispositivo crucial (ex. áudio, USB, etc.) compartilharem o mesmo grupo, só podem ser passados juntos. A solução pode ser usar `pcie_acs_override=downstream` no kernel (corrigindo grupos) [12], ou reorganizar placas em slots diferentes.
- **Erro "Code 43" no Windows**: placas NVIDIA GeForce/RTX frequentemente exibem Código 43 no Windows (detecta hypervisor). A solução é *ocultar o hypervisor* do Windows. No XML da VM, sob `<features>`, habilite `<hyperv><vendor_id state="on" value="whatever"/></hyperv>` e `<kvm><hidden state="on"/></kvm>` [16] [22]. Alternativamente, na linha de comando do QEMU use `-cpu host,kvm=off`. Isso faz com que o Windows não identifique o KVM e evite o erro 43.
- **Drivers host carregando**: se após reboot o GPU ainda aparece com driver (ex. "Kernel driver in use: nouveau" em `lspci`), verifique o arquivo de blacklist e `update-initramfs`. Pode ser necessário adicionar entradas em `/etc/modules` ou `/etc/initramfs-tools/modules` e regenerar initramfs [11] [10].

- **Sistema hospedeiro fica sem interface de vídeo**: como a GPU primária foi passada, use SSH/VNC/console ou tenha uma segunda GPU/iGPU para o host. Em emergências, pode ser necessário desabilitar o passthrough por kernelstub/GRUB se não conseguir voltar à interface gráfica.
- **Problemas de reinicialização**: alguns sistemas exigem `iommu=pt` ou `noapic` em parâmetros do kernel. Se a VM não inicia (fica travada em "loading files"), tente adicionar `kvm.ignore_msrs=1` (como já citado) e certifique-se de que o disco do Windows está configurado corretamente em virtIO ou SCSI.
- **Permissões do libvirt**: erros de QEMU ao iniciar a VM (permissão) podem ser resolvidos executando `virt-manager` como root ou ajustando políticas polkit, mas isso varia.

Fontes e recursos: A documentação oficial da Canonical aborda GPU passthrough com QEMU/KVM no Ubuntu [17] [14]. A System76 detalha o uso do *kernelstub* no Pop!_OS [8] [9]. Guias comunitários (AskUbuntu, Level1Techs) oferecem tutoriais passo-a-passo com comandos úteis [7] [11] [16]. Em caso de dúvidas, consulte também os manuais do QEMU/KVM e posts de fóruns especializados para casos específicos de hardware.

---

[1] [15] [16] [20] [22] GitHub - bryansteiner/gpu-passthrough-tutorial
https://github.com/bryansteiner/gpu-passthrough-tutorial

[2] [3] [4] [6] [7] [11] [13] [18] virtualization - Ubuntu 22.04 GPU passthrough (QEMU) - Ask Ubuntu
https://askubuntu.com/questions/1406888/ubuntu-22-04-gpu-passthrough-qemu

[5] [14] [17] GPU virtualisation with QEMU/KVM - Ubuntu Server documentation
https://documentation.ubuntu.com/server/how-to/graphics/gpu-virtualization-with-qemu-kvm/

[8] graphics card - Pop!_OS : setting kernel parameters (boot loader options) permanently is not working - Super User
https://superuser.com/questions/1682713/pop-os-setting-kernel-parameters-boot-loader-options-permanently-is-not-wor

[9] Kernelstub Usage - System76 Support
https://support.system76.com/articles/kernelstub/

[10] kernel - How do I confirm that VFIO is working in 20.04? - Ask Ubuntu
https://askubuntu.com/questions/1247058/how-do-i-confirm-that-vfio-is-working-in-20-04

[12] Problems/Solutions - Ubuntu 18.04 - VFIO PCIe Passthrough w/ Looking Glass - Linux - Level1Techs Forums
https://forum.level1techs.com/t/problems-solutions-ubuntu-18-04-vfio-pcie-passthrough-w-looking-glass/130786

[19] Graphics Switching (Pop!_OS) - System76 Support
https://support.system76.com/articles/graphics-switch-pop/

[21] VMware Workstation 17 Pro supports use of host GPU? | VMware Workstation
https://community.broadcom.com/vmware-cloud-foundation/discussion/vmware-workstation-17-pro-supports-use-of-host-gpu