GPU passthrough Nvidia

Das ganze wurde auf einem Intel Server System durchgeführt mit 2x4210 Xeon Prozessoren sowie 512 GB RAM und 4x2018 TI. Ziel ist es die 4 GPU´s in eine Linux VM durch zu reichen.
Proxmox sollte soweit fertig installiert sein. Dazu müssen wir am Host selber Änderungen vornehmen.
Das größte Problem ist Nvidia selbst. Erkennt der Nvidia Treiber eine virutelle Umgebung verweigert er für Konsumer Karten den Betrieb. Dies läßt sich umgehen. Wie steht weiter unten.

Host Anpassungen

Bootloader anpassen

```
#vi /etc/default/grub
```

Ändere folgenden Eintrag

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
```

zu

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on"
```

update den Bootloader

```
 update-grub
```

Danach führe folgendes Commando aus

```
dmesg | grep -e DMAR -e IOMMU
```

Es sollte dann solch eine ähnliche Ausgabe kommen, das ist immer unterschiedlich je nach Hardware

```
[redacted]:/etc/pve/qemu-server# dmesg | grep -e DMAR -e IOMMU
[    0.020164] ACPI: DMAR 0x000000005F549B30 000250 (v01 SUPERM SMCI--MB 00000001 INTL 20091013)
[    2.381277] DMAR: IOMMU enabled
[    3.847260] DMAR: Host address width 46
[    3.847261] DMAR: DRHD base: 0x000000d37fc000 flags: 0x0
[    3.847270] DMAR: dmar0: reg_base_addr d37fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847271] DMAR: DRHD base: 0x000000e0ffc000 flags: 0x0
[    3.847277] DMAR: dmar1: reg_base_addr e0ffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847278] DMAR: DRHD base: 0x000000ee7fc000 flags: 0x0
[    3.847282] DMAR: dmar2: reg_base_addr ee7fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847283] DMAR: DRHD base: 0x000000fbffc000 flags: 0x0
[    3.847287] DMAR: dmar3: reg_base_addr fbffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847288] DMAR: DRHD base: 0x000000aaffc000 flags: 0x0
[    3.847293] DMAR: dmar4: reg_base_addr aaffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847294] DMAR: DRHD base: 0x000000b87fc000 flags: 0x0
[    3.847297] DMAR: dmar5: reg_base_addr b87fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847298] DMAR: DRHD base: 0x000000c5ffc000 flags: 0x0
[    3.847301] DMAR: dmar6: reg_base_addr c5ffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847302] DMAR: DRHD base: 0x0000009d7fc000 flags: 0x1
[    3.847305] DMAR: dmar7: reg_base_addr 9d7fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847306] DMAR: RMRR base: 0x0000006f3c7000 end: 0x0000006f3d7fff
[    3.847307] DMAR: ATSR flags: 0x0
[    3.847308] DMAR: ATSR flags: 0x0
[    3.847309] DMAR: RHSA base: 0x0000009d7fc000 proximity domain: 0x0
[    3.847310] DMAR: RHSA base: 0x000000aaffc000 proximity domain: 0x0
[    3.847311] DMAR: RHSA base: 0x000000b87fc000 proximity domain: 0x0
[    3.847311] DMAR: RHSA base: 0x000000c5ffc000 proximity domain: 0x0
[    3.847312] DMAR: RHSA base: 0x000000d37fc000 proximity domain: 0x1
[    3.847313] DMAR: RHSA base: 0x000000e0ffc000 proximity domain: 0x1
[    3.847313] DMAR: RHSA base: 0x000000ee7fc000 proximity domain: 0x1
[    3.847314] DMAR: RHSA base: 0x000000fbffc000 proximity domain: 0x1
[    3.847316] DMAR-IR: IOAPIC id 12 under DRHD base  0xc5ffc000 IOMMU 6
[    3.847317] DMAR-IR: IOAPIC id 11 under DRHD base  0xb87fc000 IOMMU 5
[    3.847317] DMAR-IR: IOAPIC id 10 under DRHD base  0xaaffc000 IOMMU 4
[    3.847318] DMAR-IR: IOAPIC id 18 under DRHD base  0xfbffc000 IOMMU 3
[    3.847319] DMAR-IR: IOAPIC id 17 under DRHD base  0xee7fc000 IOMMU 2
[    3.847320] DMAR-IR: IOAPIC id 16 under DRHD base  0xe0ffc000 IOMMU 1
[    3.847321] DMAR-IR: IOAPIC id 15 under DRHD base  0xd37fc000 IOMMU 0
[    3.847322] DMAR-IR: IOAPIC id 8 under DRHD base  0x9d7fc000 IOMMU 7
[    3.847323] DMAR-IR: IOAPIC id 9 under DRHD base  0x9d7fc000 IOMMU 7
[    3.847323] DMAR-IR: HPET id 0 under DRHD base 0x9d7fc000
[    3.847325] DMAR-IR: Queued invalidation will be enabled to support x2apic and Intr-remapping.
[    3.849669] DMAR-IR: Enabled IRQ remapping in x2apic mode
[    5.562553] DMAR: dmar5: Using Queued invalidation
[    5.562558] DMAR: dmar4: Using Queued invalidation
[    5.562568] DMAR: dmar2: Using Queued invalidation
[    5.562571] DMAR: dmar1: Using Queued invalidation
[    5.562575] DMAR: dmar0: Using Queued invalidation
[    5.562585] DMAR: dmar7: Using Queued invalidation
[    5.619414] DMAR: Intel(R) Virtualization Technology for Directed I/O
[redacted]:/etc/pve/qemu-server#
```

Danach müssen wir ein paar Module beim Booten laden

```
nano /etc/modules
```

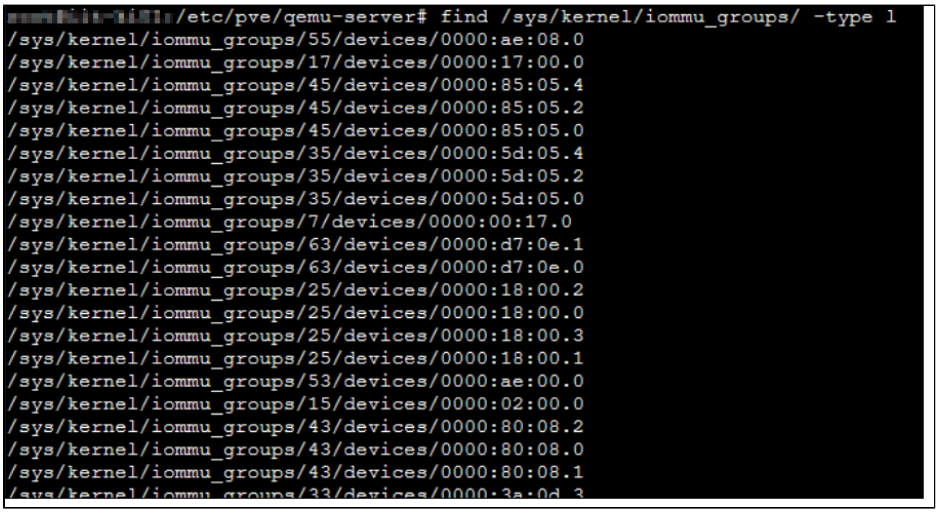
das einfügen

```
vfio
vfio_iommu_type1
vfio_pci
vfio_virqfd
```

Danach rebooten.

Verify IOMMU isolation

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



Die Liste kann lang werden.

Wichtig ist auf dem Host System, er soll keine Treiber laden für die GPU´s, weil sie sonst schon in Benutzung sind.

```
echo "blacklist radeon" >> /etc/modprobe.d/blacklist.conf
echo "blacklist nouveau" >> /etc/modprobe.d/blacklist.conf
echo "blacklist nvidia" >> /etc/modprobe.d/blacklist.conf
```

Was wir später noch brauchen werden sind die PCI ID von den GPU´s. Es stehen zwei Befehle zur Auswahl

```
lspci | grep -i --color 'NVIDIA\|3d\|2d'

lspci -nn -d 10de:
```

```
:/etc/pve/qemu-server# lspci -nn -d 10de:
18:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti] [10de:1e04] (rev a1)
18:00.1 Audio device [0403]: NVIDIA Corporation TU102 High Definition Audio Controller [10de:10f7] (rev a1)
18:00.2 USB controller [0c03]: NVIDIA Corporation TU102 USB 3.1 Controller [10de:1ad6] (rev a1)
18:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 UCSI Controller [10de:1ad7] (rev a1)
3b:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti] [10de:1e04] (rev a1)
3b:00.1 Audio device [0403]: NVIDIA Corporation TU102 High Definition Audio Controller [10de:10f7] (rev a1)
3b:00.2 USB controller [0c03]: NVIDIA Corporation TU102 USB 3.1 Controller [10de:1ad6] (rev a1)
3b:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 UCSI Controller [10de:1ad7] (rev a1)
86:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti] [10de:1e04] (rev a1)
86:00.1 Audio device [0403]: NVIDIA Corporation TU102 High Definition Audio Controller [10de:10f7] (rev a1)
86:00.2 USB controller [0c03]: NVIDIA Corporation TU102 USB 3.1 Controller [10de:1ad6] (rev a1)
86:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 UCSI Controller [10de:1ad7] (rev a1)
af:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti] [10de:1e04] (rev a1)
af:00.1 Audio device [0403]: NVIDIA Corporation TU102 High Definition Audio Controller [10de:10f7] (rev a1)
af:00.2 USB controller [0c03]: NVIDIA Corporation TU102 USB 3.1 Controller [10de:1ad6] (rev a1)
af:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 UCSI Controller [10de:1ad7] (rev a1)
/etc/pve/qemu-server#
```

Prüfen ob alles passt

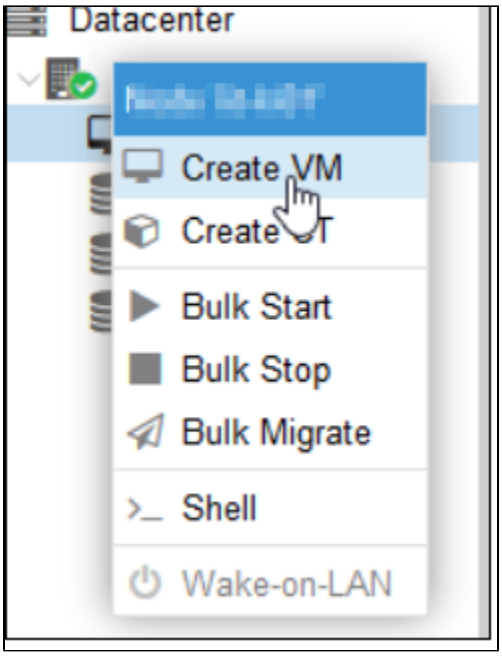
```
dmesg | grep ecap
```

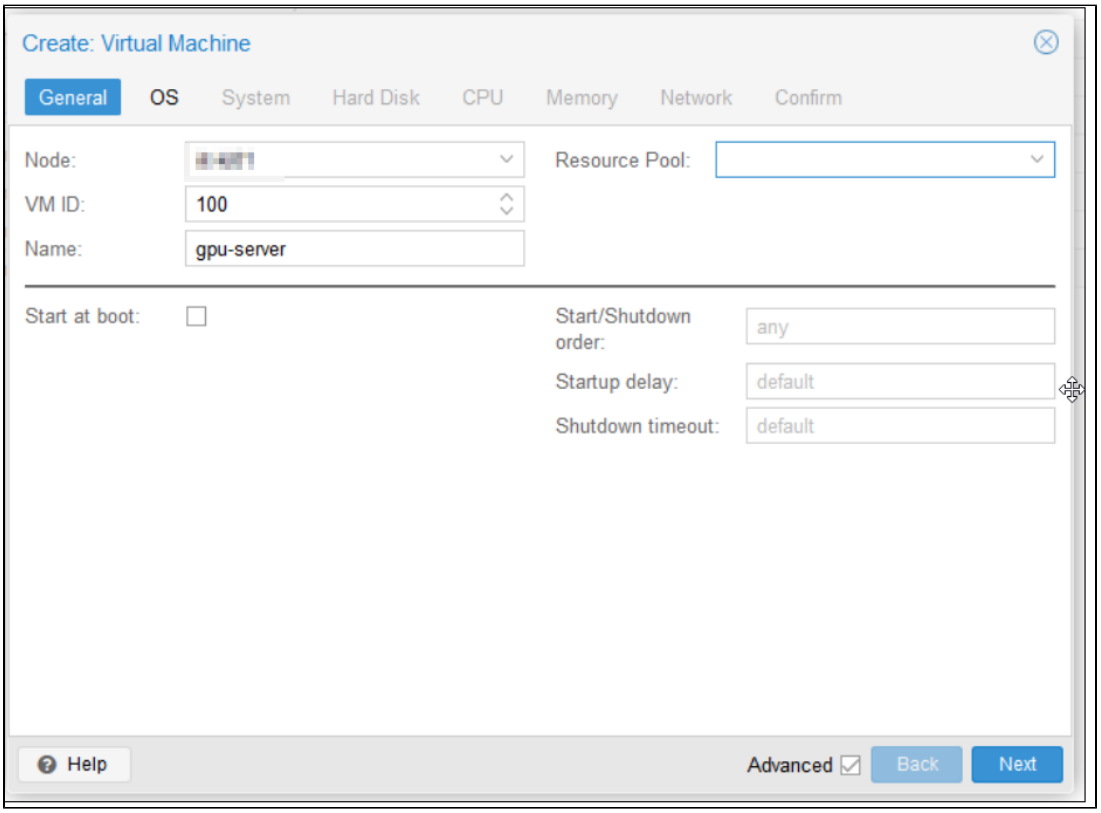
```
/etc/pve/qemu-server# dmesg | grep ecap
[    3.847270] DMAR: dmar0: reg_base_addr d37fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847277] DMAR: dmar1: reg_base_addr e0ffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847282] DMAR: dmar2: reg_base_addr ee7fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847287] DMAR: dmar3: reg_base_addr fbffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847293] DMAR: dmar4: reg_base_addr aaffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847297] DMAR: dmar5: reg_base_addr b87fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847301] DMAR: dmar6: reg_base_addr c5ffc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[    3.847305] DMAR: dmar7: reg_base_addr 9d7fc000 ver 1:0 cap 8d2078c106f0466 ecap f020df
[ 1365.343508] vfio-pci 0000:18:00.0: vfio_ecap_init: hiding ecap 0x1e@0x258
[ 1365.343528] vfio-pci 0000:18:00.0: vfio_ecap_init: hiding ecap 0x19@0x900
[ 3507.667300] vfio-pci 0000:18:00.0: vfio_ecap_init: hiding ecap 0x1e@0x258
[ 3507.667317] vfio-pci 0000:18:00.0: vfio_ecap_init: hiding ecap 0x19@0x900
[ 5596.259352] vfio-pci 0000:18:00.0: vfio_ecap_init: hiding ecap 0x1e@0x258
```

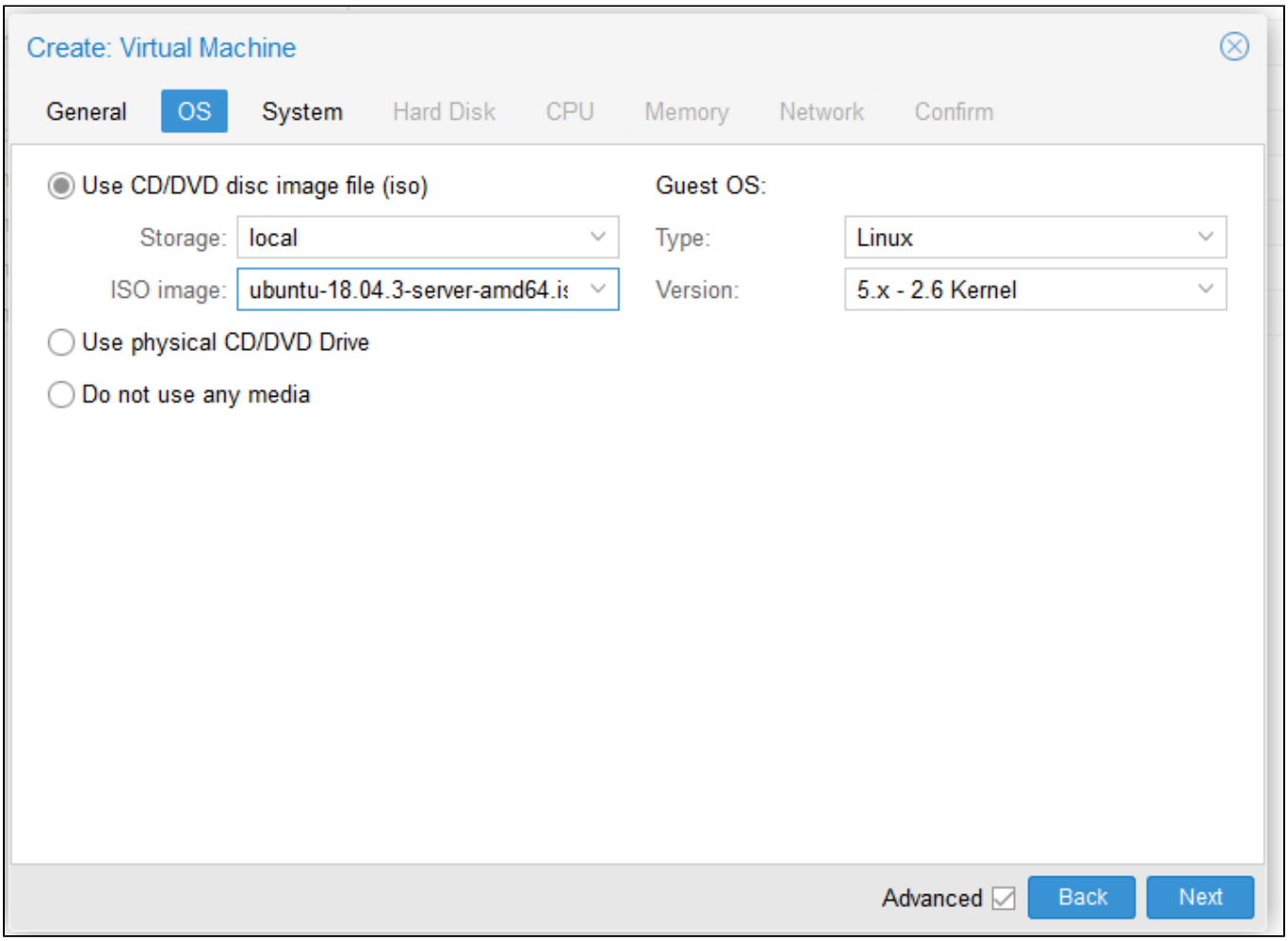
Dann ist das Host System fürs erste Bereit.

VM Bereistellen

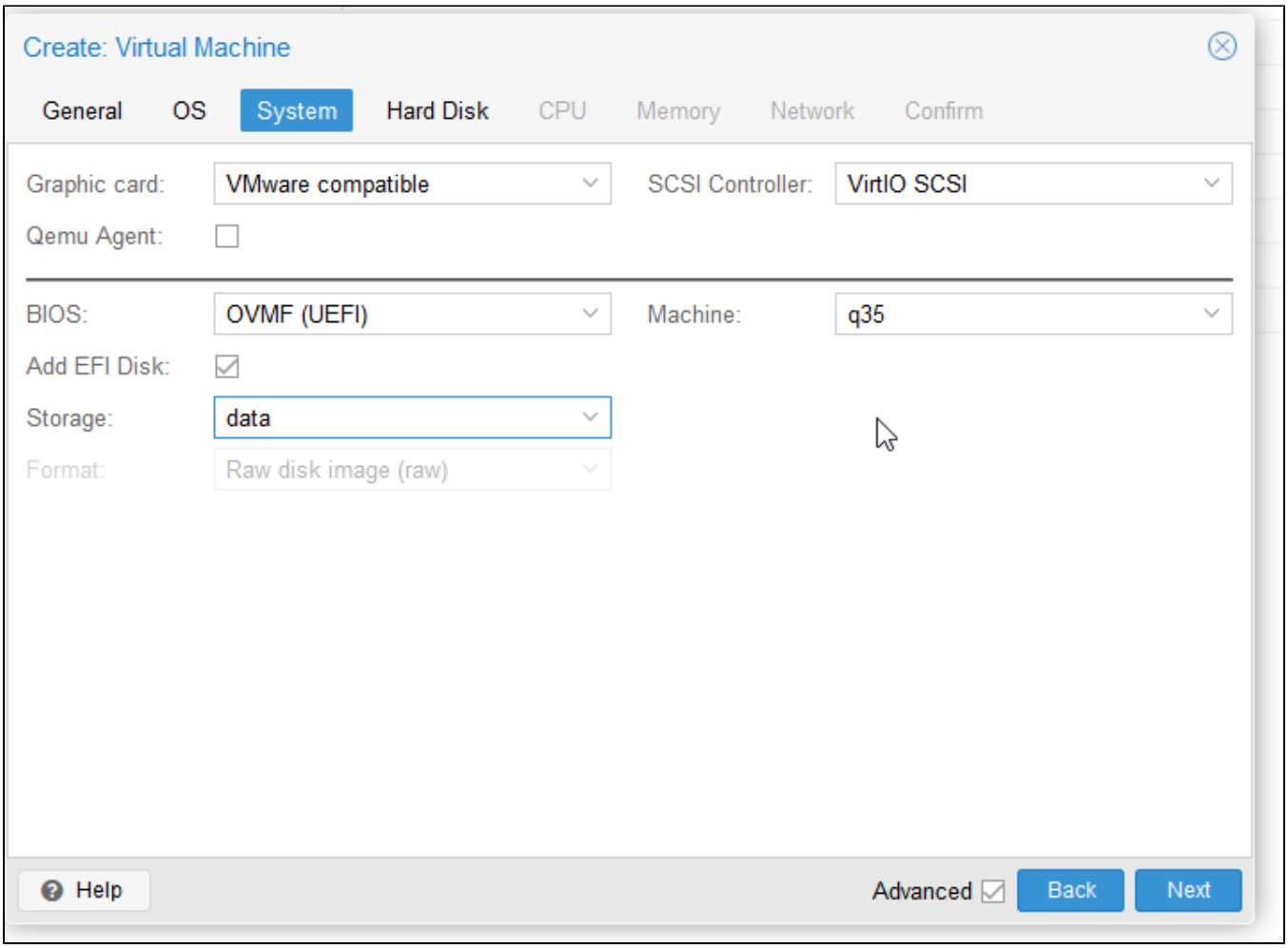
Wichtig ist das wir die VM in UEFI Mode installiert wird und nicht starten bevor es fertig konfiguriert ist.


Datacenter
Create VM
Create CT
Bulk Start
Bulk Stop
Bulk Migrate
Shell
Wake-on-LAN

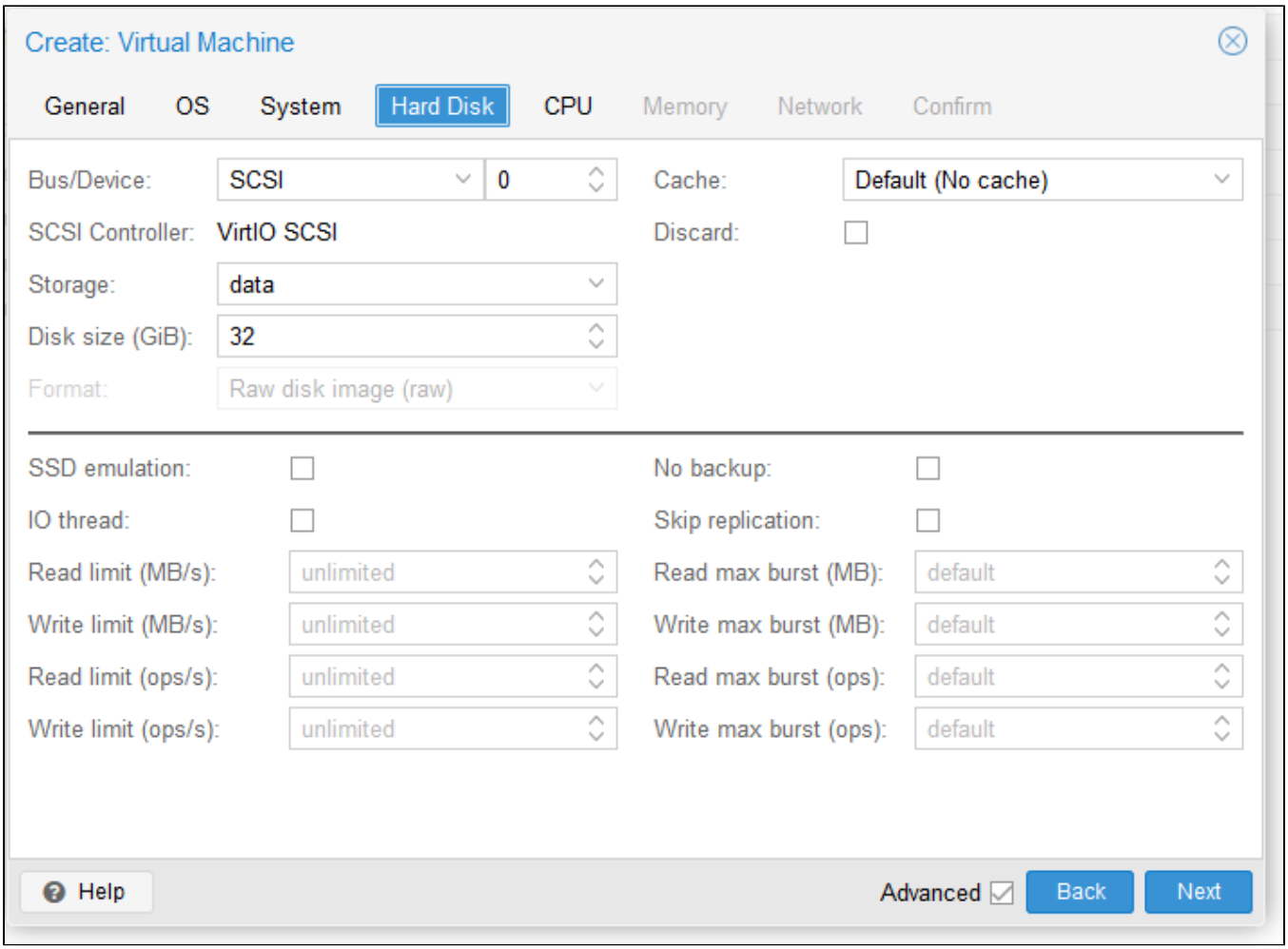

Create: Virtual Machine
General
OS
System
Hard Disk
CPU
Memory
Network
Confirm
Node:
Resource Pool:
VM ID:
100
Name:
gpu-server
Start at boot:
Start/Shutdown order:
any
Startup delay:
default
Shutdown timeout:
default
Help
Advanced
Back
Next

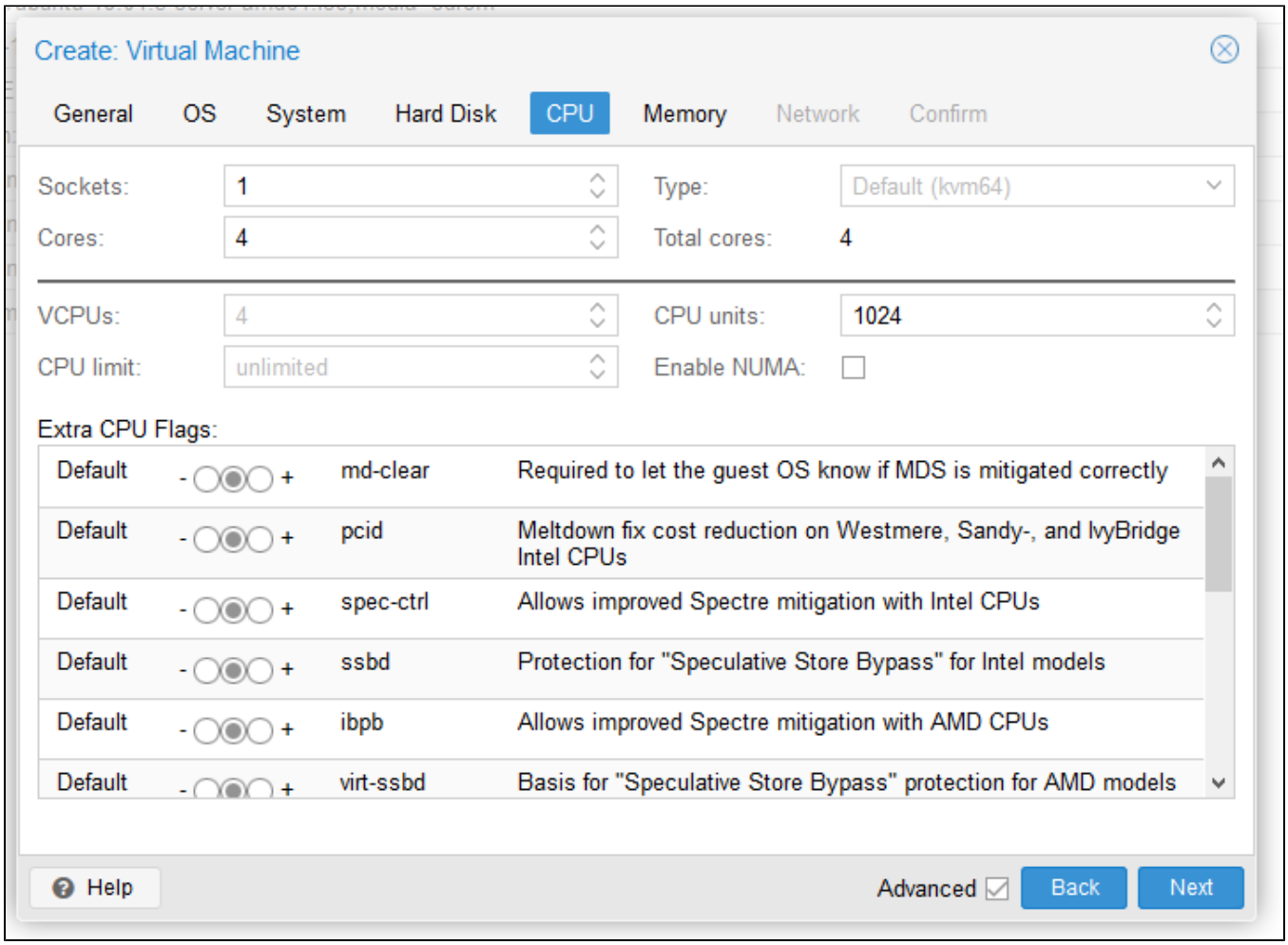


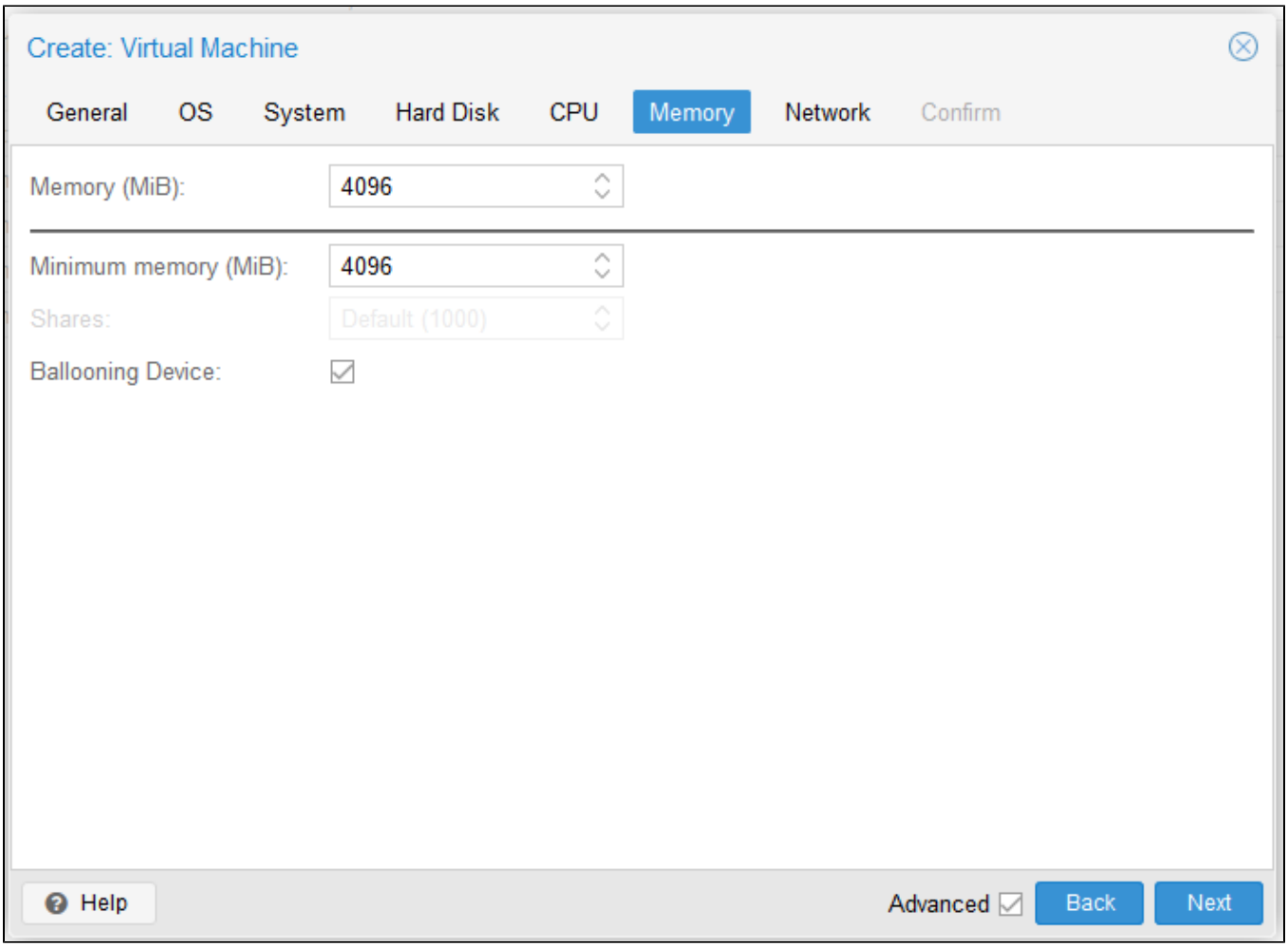
Hier stellen wir einiges um

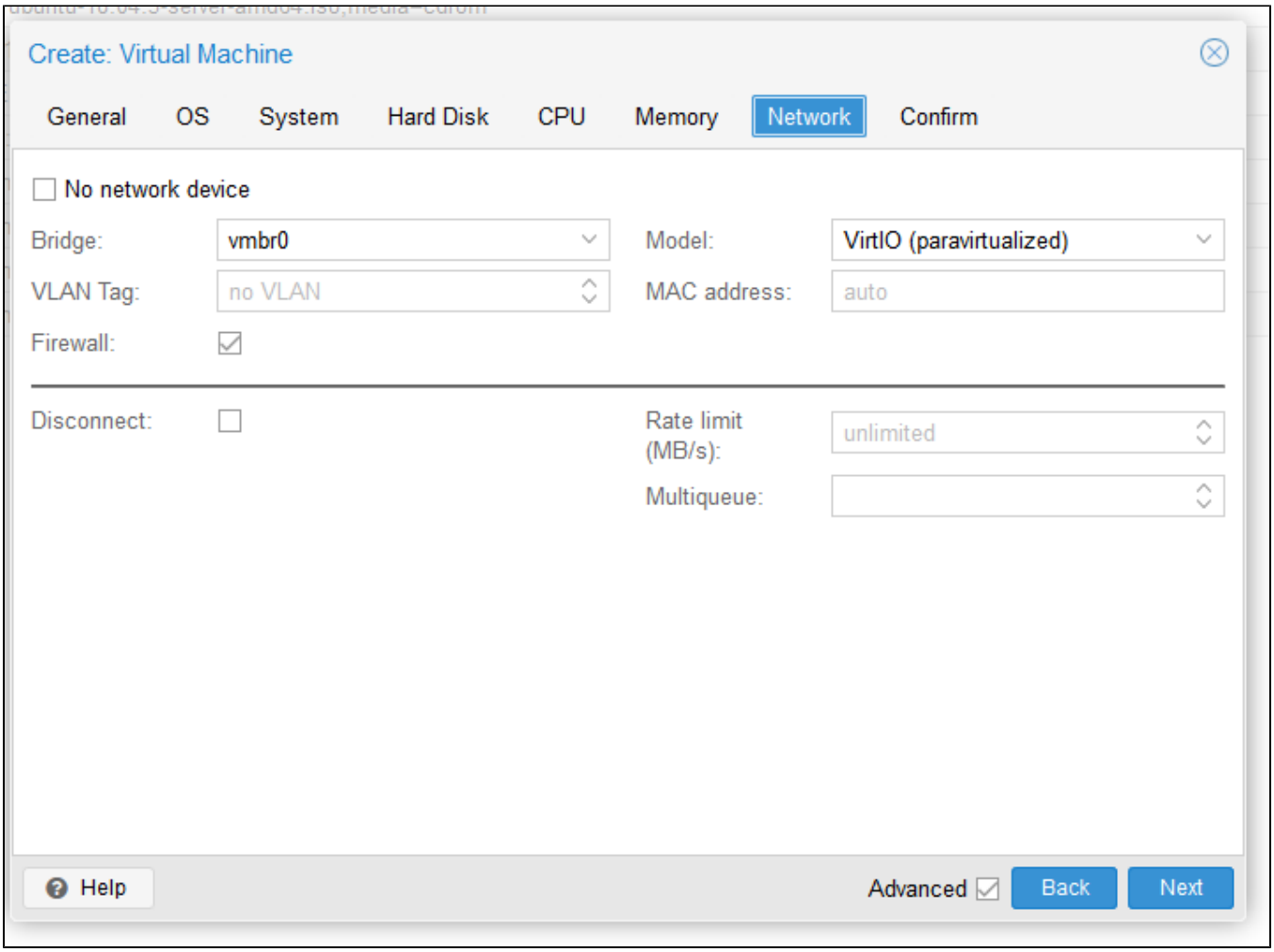


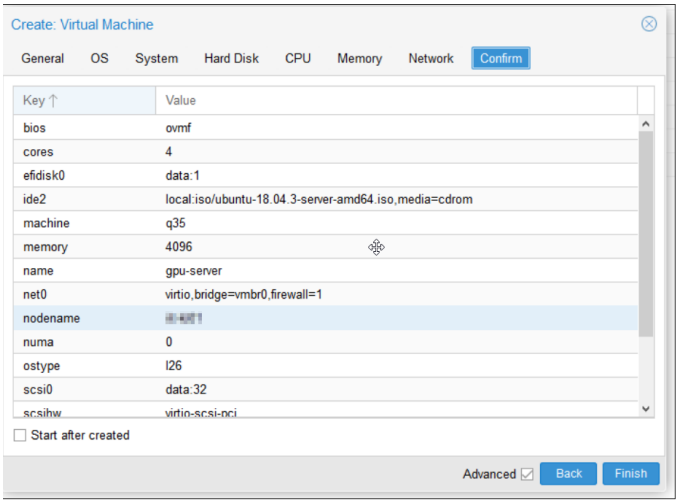
Als erstes die Graka auf VMWARE, das BIOS auf OVMF, sowie Machine auf Q35.
Für den UEFI Bootloader wird noch etwas Speicher benötigt. Diesen zuweisen.


Create: Virtual Machine
General
OS
System
Hard Disk
CPU
Memory
Network
Confirm
Bus/Device:
SCSI
0
Cache:
Default (No cache)
SCSI Controller:
VirtIO SCSI
Discard:
Storage:
data
Disk size (GiB):
32
Format:
Raw disk image (raw)
SSD emulation:
No backup:
IO thread:
Skip replication:
Read limit (MB/s):
unlimited
Read max burst (MB):
default
Write limit (MB/s):
unlimited
Write max burst (MB):
default
Read limit (ops/s):
unlimited
Read max burst (ops):
default
Write limit (ops/s):
unlimited
Write max burst (ops):
default
Help
Advanced
Back
Next


Create: Virtual Machine
General
OS
System
Hard Disk
CPU
Memory
Network
Confirm
Sockets:
1
Type:
Default (kvm64)
Cores:
4
Total cores:
4
VCPUs:
4
CPU units:
1024
CPU limit:
unlimited
Enable NUMA:
Extra CPU Flags:
Default
md-clear
Required to let the guest OS know if MDS is mitigated correctly
Default
pcid
Meltdown fix cost reduction on Westmere, Sandy-, and IvyBridge Intel CPUs
Default
spec-ctrl
Allows improved Spectre mitigation with Intel CPUs
Default
ssbd
Protection for "Speculative Store Bypass" for Intel models
Default
ibpb
Allows improved Spectre mitigation with AMD CPUs
Default
virt-ssbd
Basis for "Speculative Store Bypass" protection for AMD models
Help
Advanced
Back
Next


Create: Virtual Machine
General
OS
System
Hard Disk
CPU
Memory
Network
Confirm
Memory (MiB):
4096
Minimum memory (MiB):
4096
Shares:
Default (1000)
Ballooning Device:
Help
Advanced
Back
Next





Die VM nicht starten. Wir müssen noch ein paar Anpassungen vornehmen.

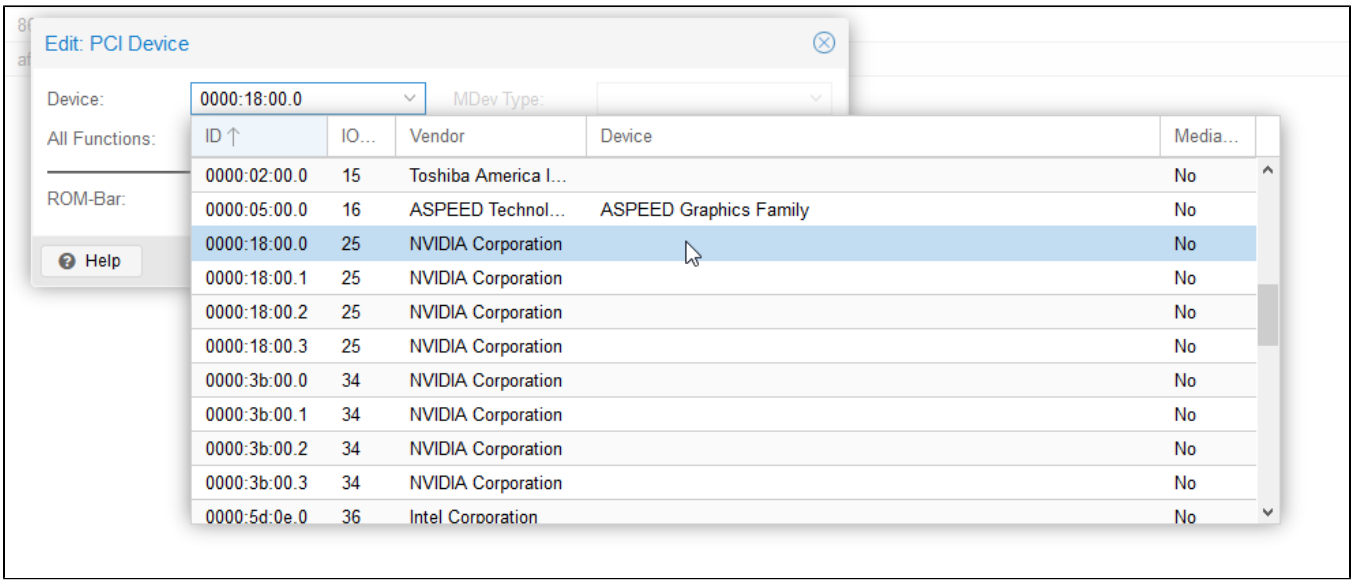
Wir müssen im Config File der VM folgende Option setzen.

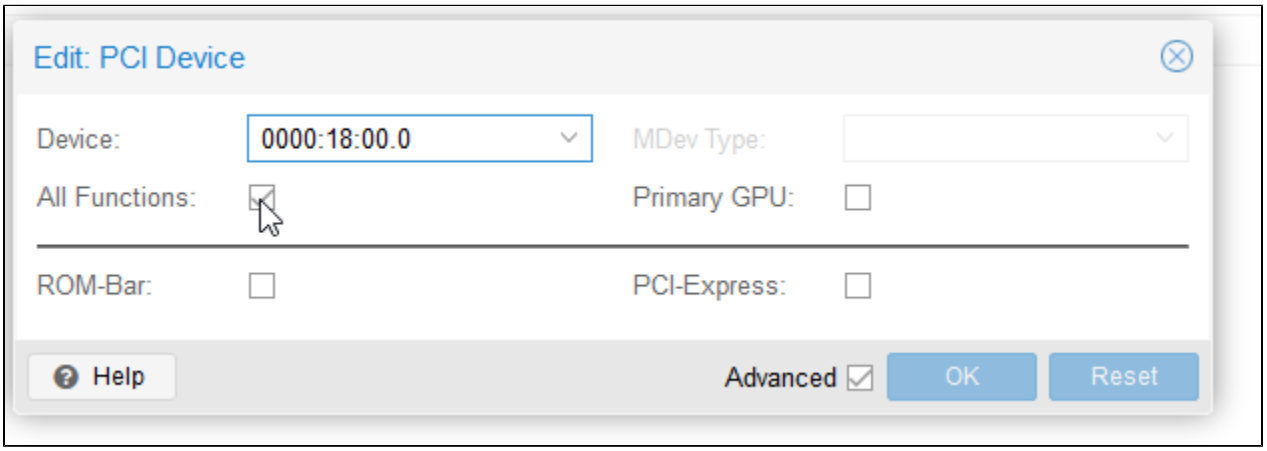
/etc/pve/qemu-server und dann dort die Server ID suchen, ist eine Text Datei.

```
cpu: host,hidden=1
```

Die Option bewirkt folgendes: es verschleiert dem Gast System dass es in einer virtuellen Umgebung läuft.
Dies ist wichtig, da der Nvidia Treiber sich sonst nicht installieren läßt. Nvidia blockiert dies per Treiber. Nur Grid/Tesla Karten dürfen laut Nvidia in einer VM laufen.

Graka der VM zuweisen





Ich habe mich hier für All Functions entschieden, sonst muss man jedes einzelne Geräte der Graka zuweisen, also Sound etc was die Graka noch mitbringt. Wir sind damit auch am Ende der Virtualisierungs angelangt. Es können nur 16 Geräte durchgereicht werden.

Funktionierende Konfig

zu finden unter /etc/pve/qemu-server sowie die Server ID ist.

```
bios: ovmf
bootdisk: scsi0
cores: 4
cpu: host,hidden=1
efidisk0: local-lvm:vm-101-disk-0,size=128K
hostpci0: 18:00,rombar=0
hostpci1: 3b:00,rombar=0
hostpci2: 86:00,rombar=0
hostpci3: af:00,rombar=0
ide2: local:iso/ubuntu-18.04.3-server-amd64.iso,media=cdrom
machine: q35
memory: 4096
name: test3
net0: virtio=DE:05:48:4F:25:BC,bridge=vmbr0,firewall=1
numa: 0
ostype: l26
scsi0: data:vm-101-disk-0,size=32G
scsihw: virtio-scsi-pci
smbios1: uuid=ce133dc1-c7c2-4bce-a805-8d03087cd545
sockets: 1
vga: vmware
vmgenid: ad163397-969a-4e32-8b4d-43e70cc94d7d
```

Dann installieren wir Ubuntu 18.04 im UEFI Mode.

Nach der installation können wir das wie folgt prüfen

```
[ -d /sys/firmware/efi ] && echo UEFI || echo BIOS
```

Danach installieren wir erst mal alle Updates die es für das Ubuntu System gibt. Rebooten. Dann installieren wir den Nvidia Treiber.

Installation Nvidia Treiber

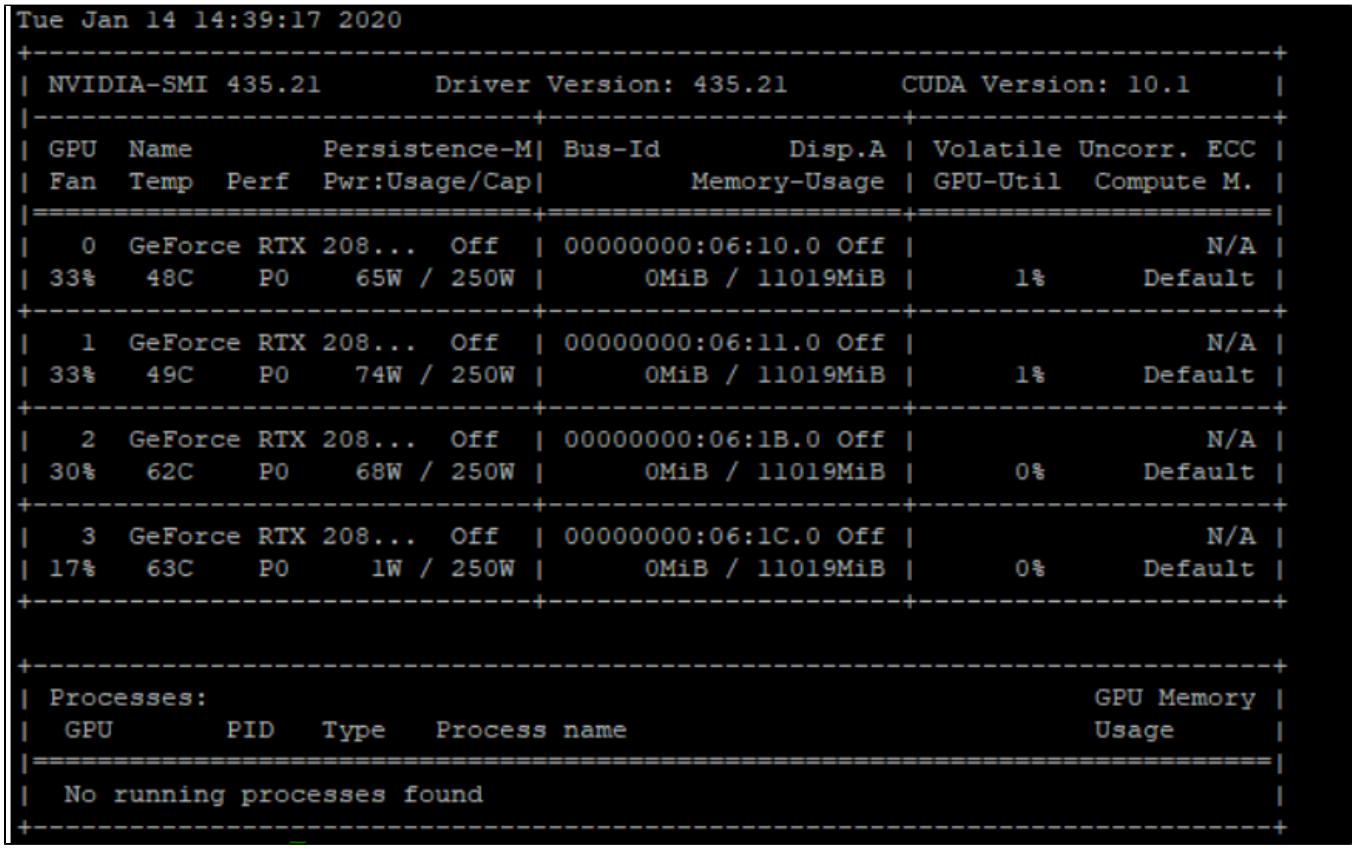
```
apt search nvidia-driver
```

Dann nehmen wir den neusten Nvidia Treiber und installieren diesen

```
apt install nvidia-driver-435
```

rebooten

Nach dem rebooten müßte es dann so aussehen, prüfen tun wir es mit nvidia-smi



Wir sehen 4x 20180 TI

Nun geht es daran Docker etc zu installieren