QEMU & KVM

Quick Emulator & Kernel-based Virtual Machine

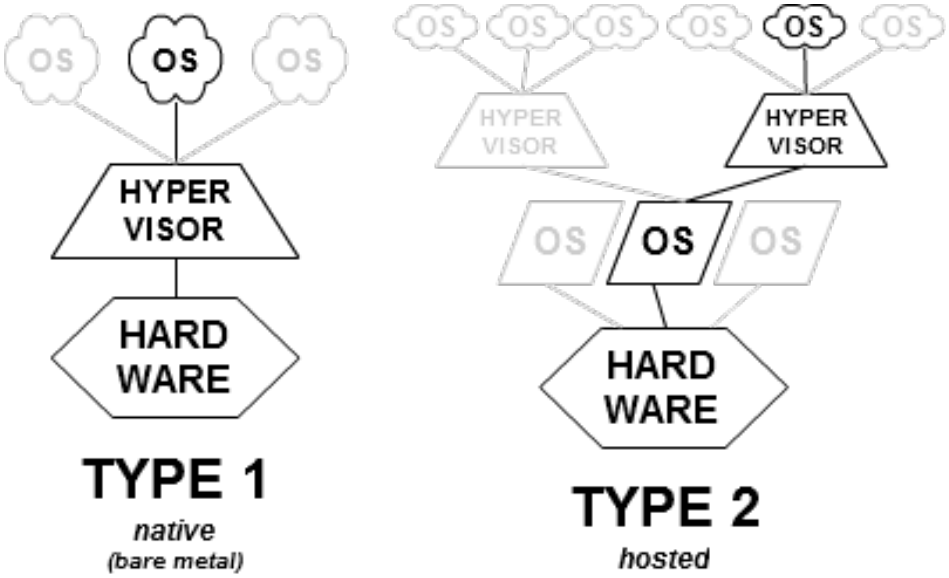
Joseph Lennon
January 2018

Hypervisors

Arbitration layer between hardware and guests.

Hypervisor variants:

- native (type 1)
- hosted (type 2)

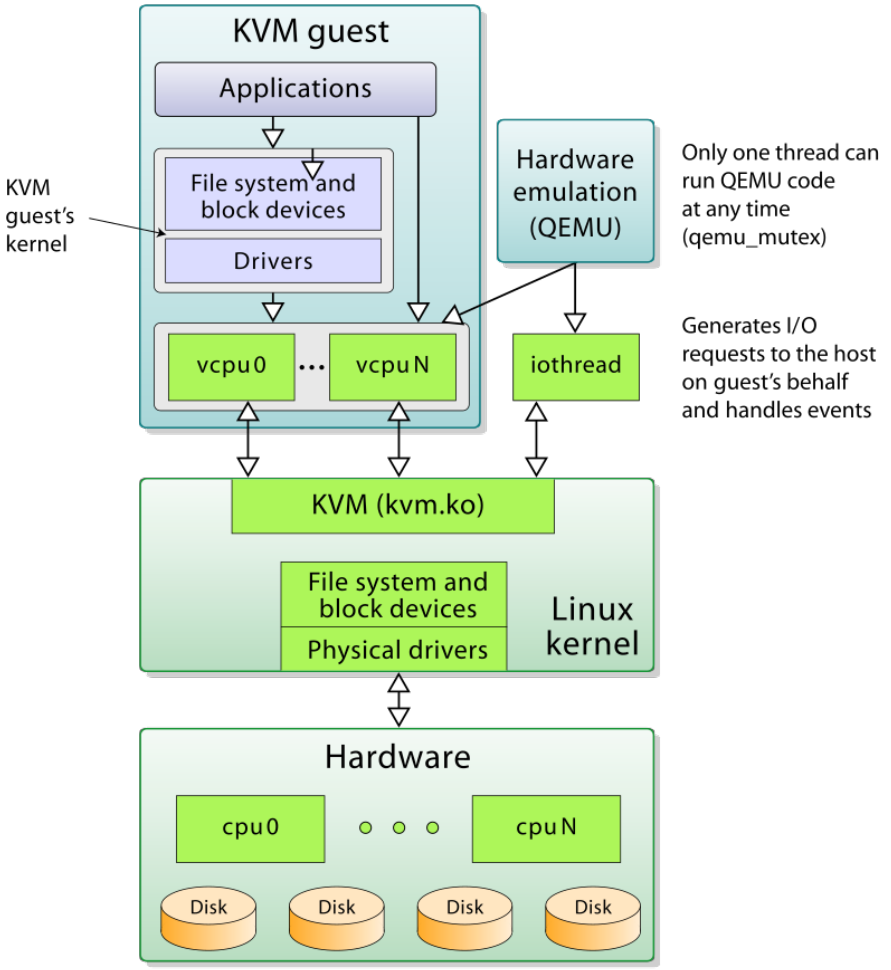


Hypervisors: KVM

Included in the mainline Linux kernel.

Leverages hardware extensions for speedup of virtual containers.

Guests interface directly with KVM module, bypassing “hosted” overhead.

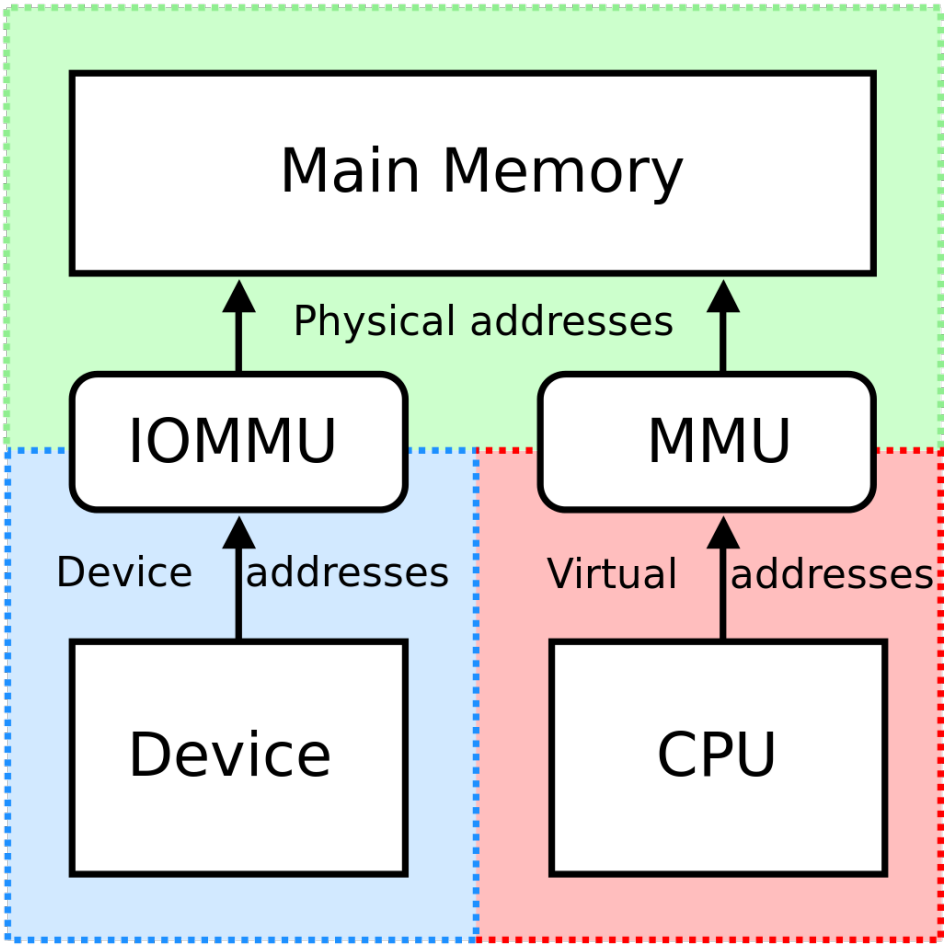


IOMMU

Similar to traditional MMUs, IOMMUs:

- Translate device address space to physical address space [1]
- Enforce memory access permissions

PCI devices are divided into **IOMMU groups** – devices mapping to the same virtual address space, and are thus indistinguishable by the IOMMU. [2]



Demo

Demo 1: General deployment and usage

- Install Libvirt and GUI management utilities
- Create and configure basic guest instance
- Deploy that guest instance

Demo 2: PCI device passthrough

- Enable IOMMU
- (Optional) Apply Access Control Services (ACS) patch
- Isolate PCI devices from host
- Install Libvirt and CLI management utilities
- Assign PCI device to guest and deploy

Demo 1: Libvirt

Libvirt provides a set of management tools for several virtualization backends, including most notably QEMU/KVM, Xen, LXC, and VirtualBox. [1]

Although not required, Libvirt's management utilities make deploying and maintaining guests very convenient.

If UEFI guest support is desired, EFI firmware may be used from most distributions' repositories (e.g. OVMF [2] [3]).

Demo 2: System Requirements

The CPU, chipset, and BIOS/UEFI must support **hardware virtualization** and **IOMMU**.

	Intel	AMD
H/W Virt.	**VT-x**	**AMD-V**
IOMMU	**VT-d**	**AMD-Vi**

Ensure both capabilities are enabled in the BIOS/UEFI.

Demo 2: Enable IOMMU

Verify IOMMU support is built into the kernel:

```
CONFIG_IOMMU_IOVA
CONFIG_INTEL_IOMMU  OR  CONFIG_AMD_IOMMU
```

Enable the driver by appending the relevant kernel argument:

```
intel_iommu=on  OR  amd_iommu=on
```

Verify IOMMU is correctly enabled:

```
# dmesg | grep -E “DMAR|IOMMU”
```

Demo 2: Apply ACS Patch (Optional)

If a device to be passed through shares an IOMMU group with a device not selected for passthrough, the two devices may be discriminated after applying an Access Control Services (ACS) override patch. [1] [2]

After patching and recompiling, append the following kernel argument:

- Globally enable ACS for all PCI devices:

```
pci_acs_override=downstream
```

- Selectively enable ACS for a particular PCI device:

```
pci_acs_override=id:<VENDORID>:<PRODUCTID>
```

Demo 2: Isolate PCI Devices

Devices selected for passthrough cannot be held by their respective drivers and should instead be bound to Virtual Function I/O (VFIO) at boot.

Verify VFIO support is built into the kernel:

- `CONFIG_VFIO`
- `CONFIG_VFIO_PCI`
- `CONFIG_VFIO_IOMMU_TYPE1`
- `CONFIG_VFIO_VIRQFD`

To bind the devices, provide a list of IDs to `vfio-pci`:

```
options vfio-pci ids=<VENDORID>:<PRODUCTID>,...
```

Demo 2: Isolate PCI Devices

If not compiled into the kernel, build the following modules into the initramfs and rebuild:

- `vfio`
- `vfio_pci`
- `vfio_iommu_type1`
- `vfio_virqfd`

Verify `vfio-pci` controls the selected devices:

```
$ lspci -k -d <VENDORID>:<PRODUCTID>
```

It may be necessary to blacklist a driver to forcibly bind a device to `vfio-pci`.