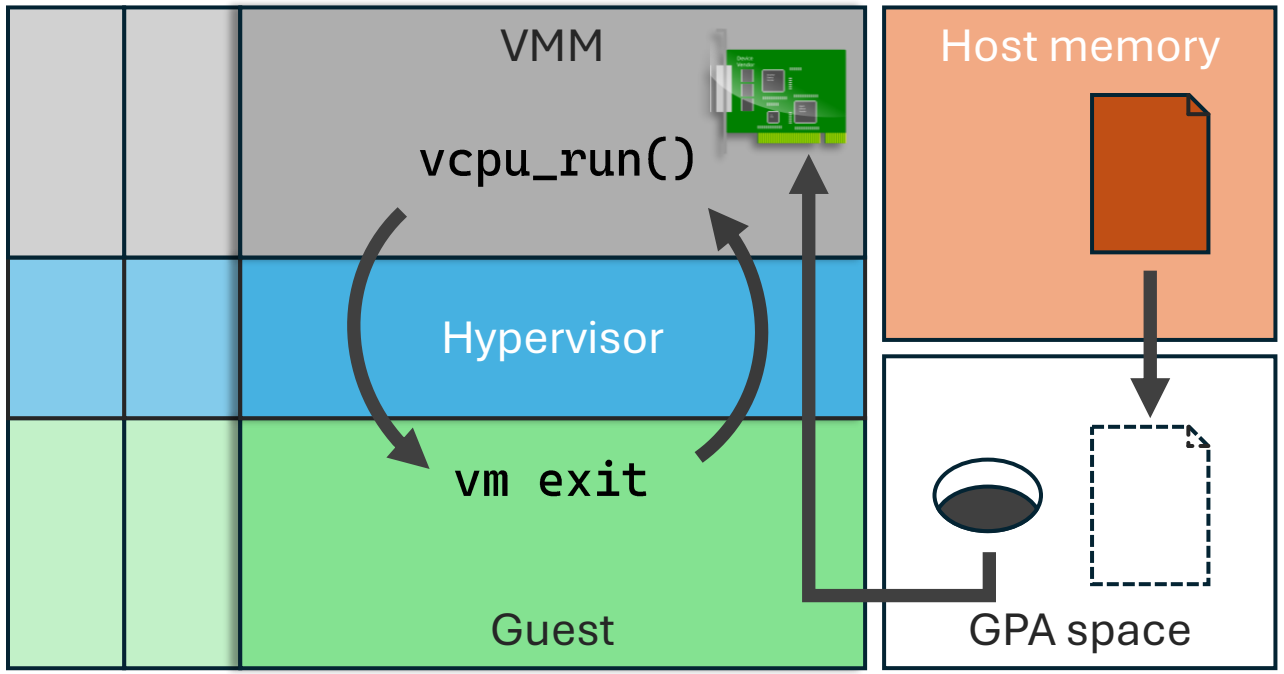
Introducing `-accel mshv` in QEMU

FOSDEM 2026, 2026-31-01

Magnus Kulke, Azure Core Upstream

QEMU Accelerators

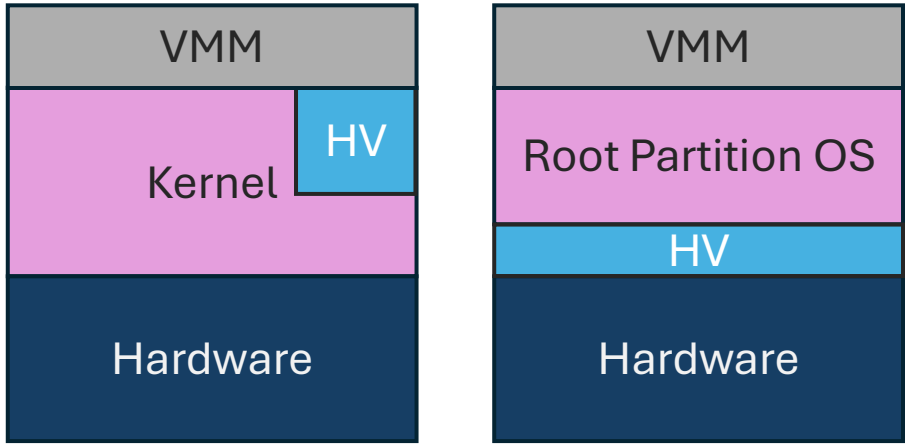
Examples: KVM, HVF, WHPX



- VMM: maps RAM/MMIO into guest space
- VMM: pass control to HV to execute guest code on vCPU
- HV: trap unmapped GPA access, yield back to VMM
- VMM: emulate device

KVM/Microsoft Hypervisor (MSHV)

- Both are Type 1 hypervisors
  - MSHV: thin dedicated layer between Hardware and root partition
  - KVM embeds hypervisor functionality into the Linux kernel

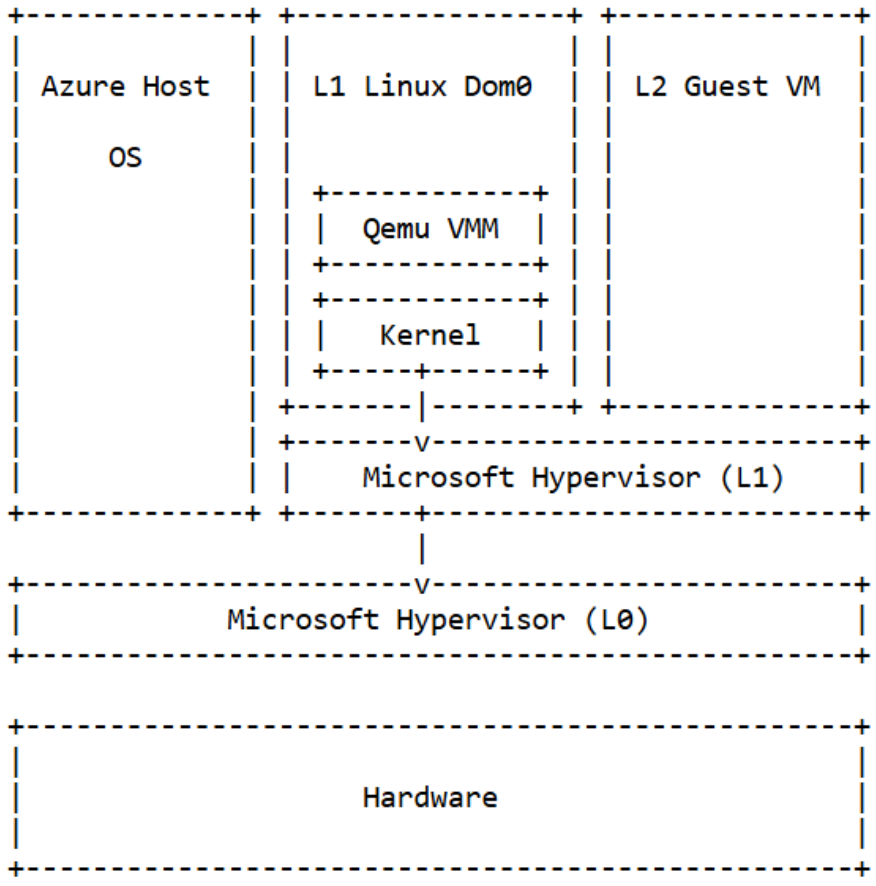


KVM

Microsoft Hypervisor

Linux Root Partitions for MSHV

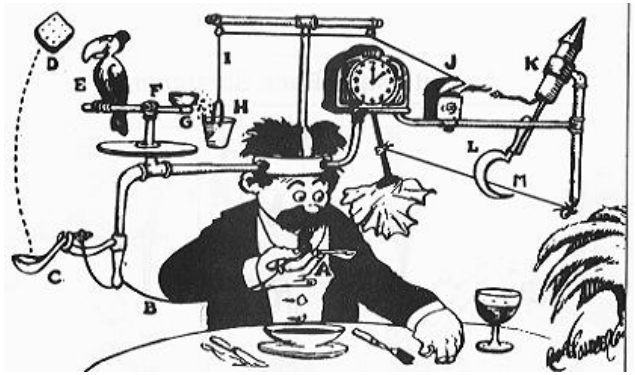
- /dev/mshv – like dev/kvm: ioctl/fd interface to manage guests + resources
- Used in AKS *Pod Sandboxing:* minimal container-optimized Linux w/ Kata + Cloud-Hypervisor on Azure Linux
- Baremetal*, Nested HV*, or Direct Virtualization
- First parts upstreamed in 6.15, incrementally more features are added



* Not yet in mainline kernel

Why QEMU + Linux + MSHV?

- Sandbox untrusted code
  - FaaS
  - Coding agents
  - “Hostile multitenancy”
- Enable CX’s to build platforms w/ virt
- Linux is the #1 workload on Azure*
- QEMU is a popular and comprehensive VMM, not just for cloud-native workloads
- Foundation for higher level virt solutions (libvirt, Kubevirt)

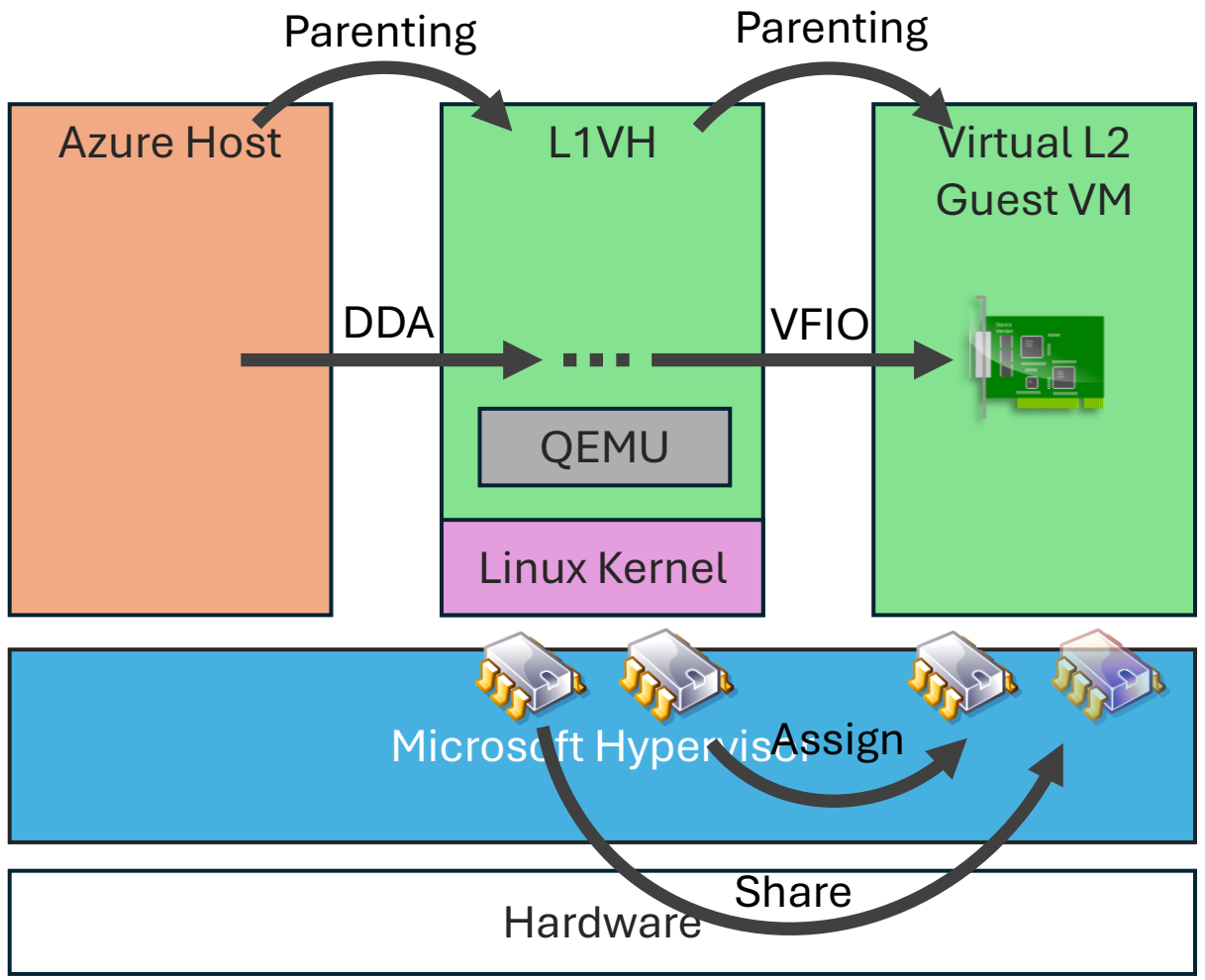


* Lessons from Running Linux at Hyperscale (OSS NA 2024)

Direct Virtualization (Hierarchical Virt.)

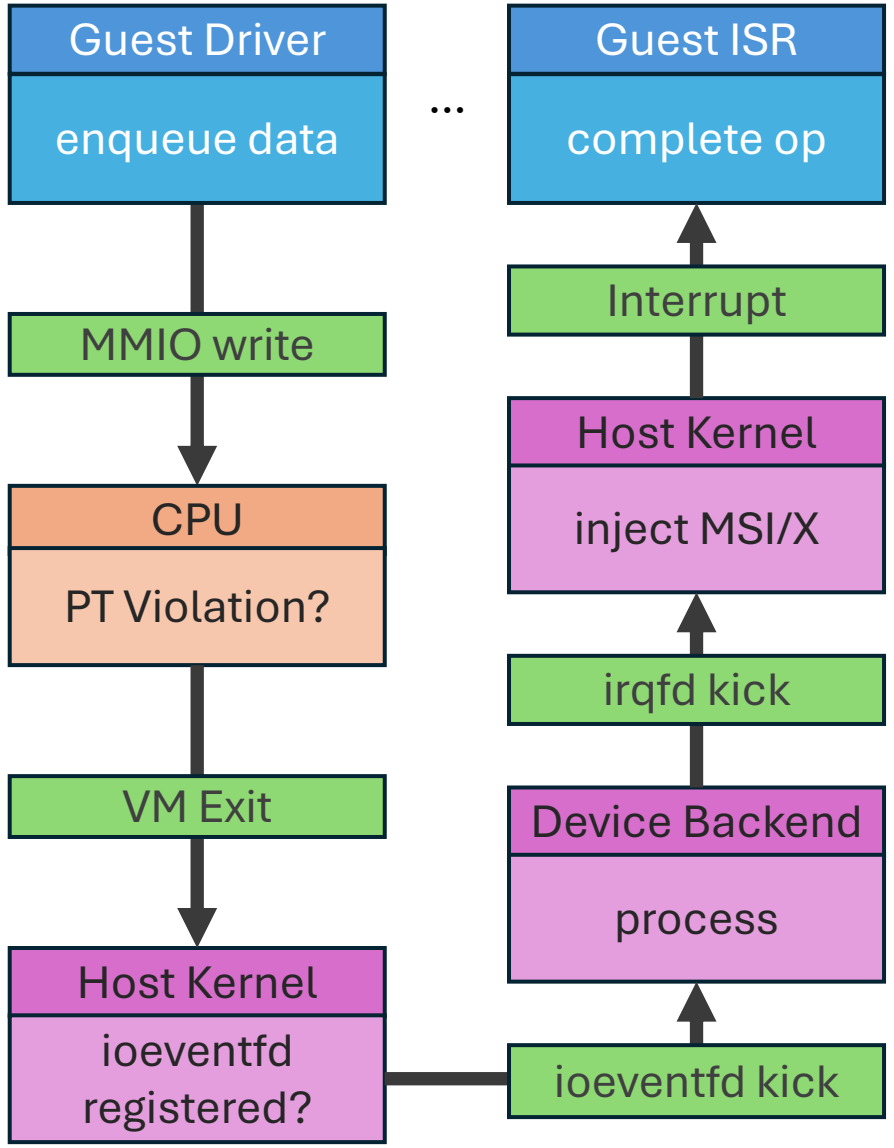
- L1 Virtual Host (L1VH): spawns Virtual L2 Guest
- L2: sibling with shared or assigned resources from L1VH
- Avoid: CPU and I/O perf hit of nested virtualization (reducing world switches)
- Enable: L1 => L2 Device assignment (GPU, NVMe)
- Announced Nov 2025*, public preview on Azure soon

* Inside Azure Innovations (Microsoft Ignite 2025)



Integration Points

- x86 instruction decoder/emu for MMIO (generalized/extended HVF impl)
- accel-irq: “split irqchip” in MSHV driver, generalized KVM impl
- irqfd/ioeventfd: use fastpaths to avoid VMM in data plane
- vfio-pci: hook up to generic accel-irq



Challenges: VFIO

- VMM bridges VFIO device w/ irqchip via eventfd
- KVM and MSHV expect different setup sequences
- IRQs are dropped
- Options:
  - Use MSHV-specific setup path
  - Adjust driver behavior
  - Adjust QEMU behavior

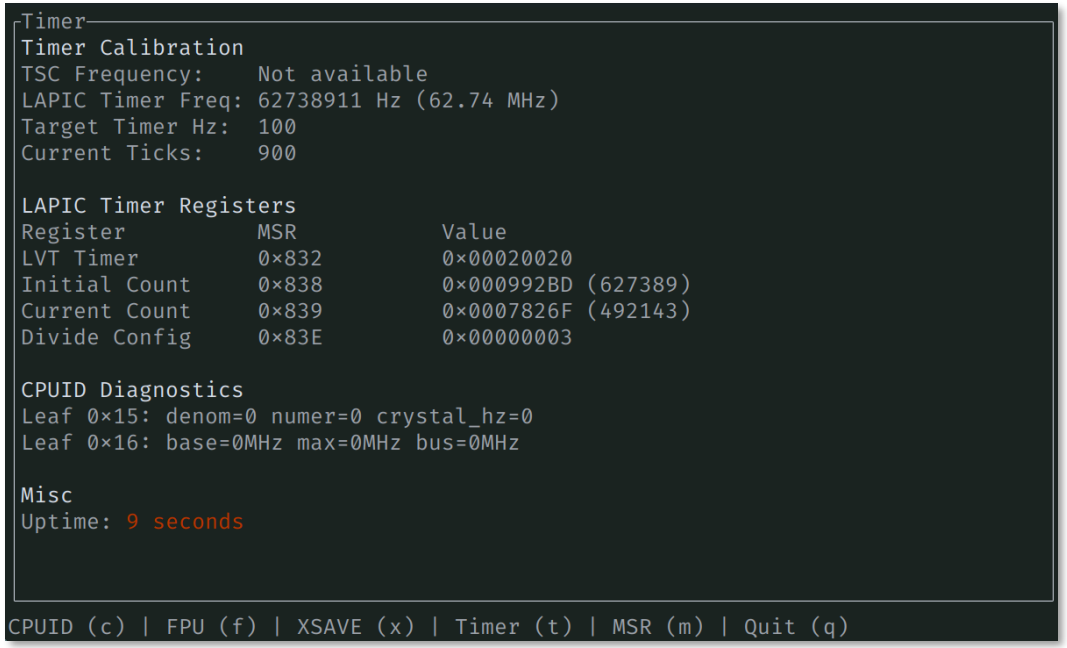
```
gsi = irqchip.add_msi_route(vfio_dev, vector_n)
irqchip.commit_routes()                    // mshv ioctl
event_fd = init_notifier()
irqchip.add_irqfd_notifier(gsi, event_fd) // mshv ioctl
vfio_dev.set_irqs(vector_n, event_fd)     // vfio ioctl
```

Simplified QEMU sequence

```
event_fd = init_notifier()
vfio_dev.set_irqs(vector_n, event_fd)     // vfio ioctl
gsi = irqchip.add_msi_route(vfio_dev, vector_n)
irqchip.commit_routes()                    // mshv ioctl
irqchip.add_irqfd_notifier(gsi, event_fd) // mshv ioctl
```

Simplified Cloud-Hypervisor sequence

Challenges: Migration



x86_64 baremetal debugging (❤️ ratatui)

- x86 guests have a lot of (implicit) state
- VMM doesn’t need to care about most of it (CPU registers)
- Migration needs to capture, serialize, roundtrip-able state
- LAPIC, XSAVE, MSRs, SynIC, …
- Long tail of subtle hard-to-reproduce issues

Upcoming

- QEMU CPU model support
- Device passthrough
- Live Migration
- ARM

thx!