

Virtualization Made in Saxony - What Is a Virtual Machine?

From Java Code to the Hypervisor: A Look Behind the Scenes

1. Why You Should Care and Why I am Here



1.1 Why You Should Care - and What to Expect

1.1 Why You Should Care - and What to Expect

- You deploy software into VMs - core digital infrastructure

1.1 Why You Should Care - and What to Expect

- You deploy software into VMs - core digital infrastructure
- We @ Cyberus Technology build it in Dresden: digital sovereignty for Germany and Europe

1.1 Why You Should Care - and What to Expect

- You deploy software into VMs - core digital infrastructure
- We @ Cyberus Technology build it in Dresden: digital sovereignty for Germany and Europe
- My way: Angular & Spring → low-level
(virtualization is no rocket science)

1.1 Why You Should Care - and What to Expect

- You deploy software into VMs - core digital infrastructure
- We @ Cyberus Technology build it in Dresden: digital sovereignty for Germany and Europe
- My way: Angular & Spring → low-level
(virtualization is no rocket science)
- Today: terminology, what a VM really is, Cloud Hypervisor, live demos
(everything focuses on `x86_64`)

1.2 \$ whoami



1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

Work

Software Engineer at Cyberus Technology

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

Work

Software Engineer at Cyberus Technology

Focus

Virtualization, Linux/KVM, x86, and Rust

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

Work

Software Engineer at Cyberus Technology

Focus

Virtualization, Linux/KVM, x86, and Rust

Open Source

Maintainer of:

- `uefi-rs` ([GitHub](#))
- `cloud-hypervisor` with focus on live migration ([GitHub](#))
- Many more Rust crates (OS development, low-level)

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

Work

Software Engineer at Cyberus Technology

Focus

Virtualization, Linux/KVM, x86, and Rust

Open Source

Maintainer of:

- `uefi-rs` ([GitHub](#))
- `cloud-hypervisor` with focus on live migration ([GitHub](#))
- Many more Rust crates (OS development, low-level)

Community

Organizer of the [Dresden Systems Meetup](#)

1.2 \$ whoami



Philipp Schuster

SYSTEMS SOFTWARE ENGINEER

 @phip1611

 phip1611.de

Work

Software Engineer at Cyberus Technology

Focus

Virtualization, Linux/KVM, x86, and Rust

Open Source

Maintainer of:

- `uefi-rs` ([GitHub](#))
- `cloud-hypervisor` with focus on live migration ([GitHub](#))
- Many more Rust crates (OS development, low-level)

Community

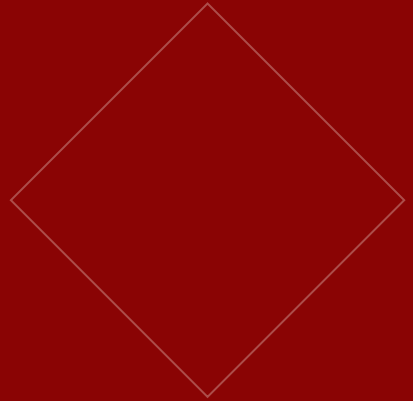
Organizer of the [Dresden Systems Meetup](#)

Talks

Regular speaker at meetups and conferences - and more to come
(JUG Saxony Event Series & Day, EuroRust, local meetups)

2. Virtualization Made in Saxony

Crafted with love and passion by Cyberus Technology in Dresden.



2.1 Cyberus Technology

CYBERUS
TECHNOLOGY

2.1 Cyberus Technology



- Founded in 2017 by six founders in Dresden (today ≈35)

2.1 Cyberus Technology

CYBERUS
TECHNOLOGY

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software
- In Dresden, Germany (not silicon valley, not USA)

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software
- In Dresden, Germany (not silicon valley, not USA)
- You might have heard of us ...

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software
- In Dresden, Germany (not silicon valley, not USA)
- You might have heard of us ...
 - Leading role in discovery of CPU vulnerabilities Meltdown & Spectre (2018)

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software
- In Dresden, Germany (not silicon valley, not USA)
- You might have heard of us ...
 - Leading role in discovery of CPU vulnerabilities Meltdown & Spectre (2018)
 - We developed the KVM-backend for VirtualBox

2.1 Cyberus Technology

- Founded in 2017 by six founders in Dresden (today ≈35)
- Independent, profitable
- Best-in-class expertise in x86 and virtualization
- Focus in reproducible, trustworthy, and solid engineering and software
- In Dresden, Germany (not silicon valley, not USA)
- You might have heard of us ...
 - Leading role in discovery of CPU vulnerabilities Meltdown & Spectre (2018)
 - We developed the KVM-backend for VirtualBox
 - Major contributor to Cloud Hypervisor (next to Meta and Microsoft)

2.2 Our Mission

*... is to build **open-source digital infrastructure** that stands the test of time. We engineer modular, transparent and **reproducible software** that "just works," empowering organisations to run critical systems safely for decades.*

Our goal is digital sovereignty in Germany and Europe (while building cool stuff)!

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
 - **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases
 - **BSI-accredited ("Zulassung") for *VS-NfD*¹ and *NATO Restricted*** 🎉
- (Since September 15th 2026 - [BSI Website](#))

¹ *VS-NfD*: "Verschlusssache - Nur für den Dienstgebrauch", the German classification entry level for classified information

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases
 - **BSI-accredited ("Zulassung") for *VS-NfD*¹ and *NATO Restricted*** 🎉
(Since September 15th 2026 - [BSI Website](#))
 - The **only** virtualization stack in Germany

¹ *VS-NfD*: "Verschlusssache - Nur für den Dienstgebrauch", the German classification entry level for classified information

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases
 - **BSI-accredited ("Zulassung") for VS-NfD¹ and NATO Restricted** 🎉
(Since September 15th 2026 - [BSI Website](#))
 - The **only** virtualization stack in Germany
- Enabling European solutions for sovereign clouds

¹ VS-NfD: "Verschlusssache - Nur für den Dienstgebrauch", the German classification entry level for classified information

2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases
 - **BSI-accredited ("Zulassung")** for *VS-NfD*¹ and *NATO Restricted* 🎉
(Since September 15th 2026 - [BSI Website](#))
 - The **only** virtualization stack in Germany
- Enabling European solutions for sovereign clouds
- We deliver components to integrators

¹ *VS-NfD*: "Verschlusssache - Nur für den Dienstgebrauch", the German classification entry level for classified information

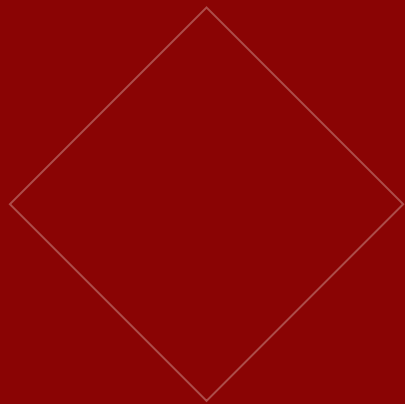
2.3 Our Offerings & Value Proposition

Selected items and one unique selling point.

- Virtualization based on Linux/KVM is main focus
- **Cyberus Hypervisor** (Cloud Hypervisor + Linux/KVM + Service & Expertise)
 - For cloud and embedded use cases
 - **BSI-accredited ("Zulassung") for *VS-NfD*¹ and *NATO Restricted*** 🎉
(Since September 15th 2026 - [BSI Website](#))
 - The **only** virtualization stack in Germany
- Enabling European solutions for sovereign clouds
- We deliver components to integrators
- We are building an open-source based cloud platform with SAP ([Apeiro / CobaltCore](#))

¹ *VS-NfD*: "Verschlusssache - Nur für den Dienstgebrauch", the German classification entry level for classified information

3. Background



3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

- **Hypervisor:** Privileged software component running in kernel-space (e.g. Linux/KVM)

3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

- **Hypervisor:** Privileged software component running in kernel-space (e.g. Linux/KVM)
- **Virtual Machine Monitor (VMM):** Unprivileged software component (like a regular user-space application, e.g. QEMU, VirtualBox, Cloud Hypervisor)

3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

- **Hypervisor:** Privileged software component running in kernel-space
(e.g. Linux/KVM)
- **Virtual Machine Monitor (VMM):** Unprivileged software component
(like a regular user-space application, e.g. QEMU, VirtualBox, Cloud Hypervisor)
- **Cloud Hypervisor (CH):** A VMM written in Rust, using Linux/KVM as hypervisor
(Naming things is hard! 🤔)

3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

- **Hypervisor:** Privileged software component running in kernel-space
(e.g. Linux/KVM)
- **Virtual Machine Monitor (VMM):** Unprivileged software component
(like a regular user-space application, e.g. QEMU, VirtualBox, Cloud Hypervisor)
- **Cloud Hypervisor (CH):** A VMM written in Rust, using Linux/KVM as hypervisor
(Naming things is hard! 🤪)
- **Virtualization Stack:** Hypervisor + VMM [+ Management Software]

3.0 Terminology: Hypa Hypa Hypervisor?

Many different wordings out there. I prefer these fine-grained definitions:

- **Hypervisor:** Privileged software component running in kernel-space (e.g. Linux/KVM)
- **Virtual Machine Monitor (VMM):** Unprivileged software component (like a regular user-space application, e.g. QEMU, VirtualBox, Cloud Hypervisor)
- **Cloud Hypervisor (CH):** A VMM written in Rust, using Linux/KVM as hypervisor (Naming things is hard! 🤔)
- **Virtualization Stack:** Hypervisor + VMM [+ Management Software]
- **Guest:** Software running in a VM (OS + user applications), e.g. Windows or Debian

3.1 What is Virtualization?

3.1 What is Virtualization?

Virtualization is the abstraction of a physical (or otherwise underlying) resources into (multiple) virtual instances so that they can act independently.

3.2 What is a VM?

3.2 What is a VM?

- Java Virtual Machine? 🤔

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated
 - Abstracting the host hardware to create multiple execution environments

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated
 - Abstracting the host hardware to create multiple execution environments
 - Behave like the normal host computing platform for the guest¹

¹ the software (operating system) running in the VM

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated
 - Abstracting the host hardware to create multiple execution environments
 - Behave like the normal host computing platform for the guest¹
- **Hardware-assisted virtualization** (Hardware-accelerated VM)

¹ the software (operating system) running in the VM

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated
 - Abstracting the host hardware to create multiple execution environments
 - Behave like the normal host computing platform for the guest¹
- **Hardware-assisted virtualization** (Hardware-accelerated VM)
 - Hardware (CPU) offers features to run guests directly on the CPU
(Alternative would be emulation → slow)

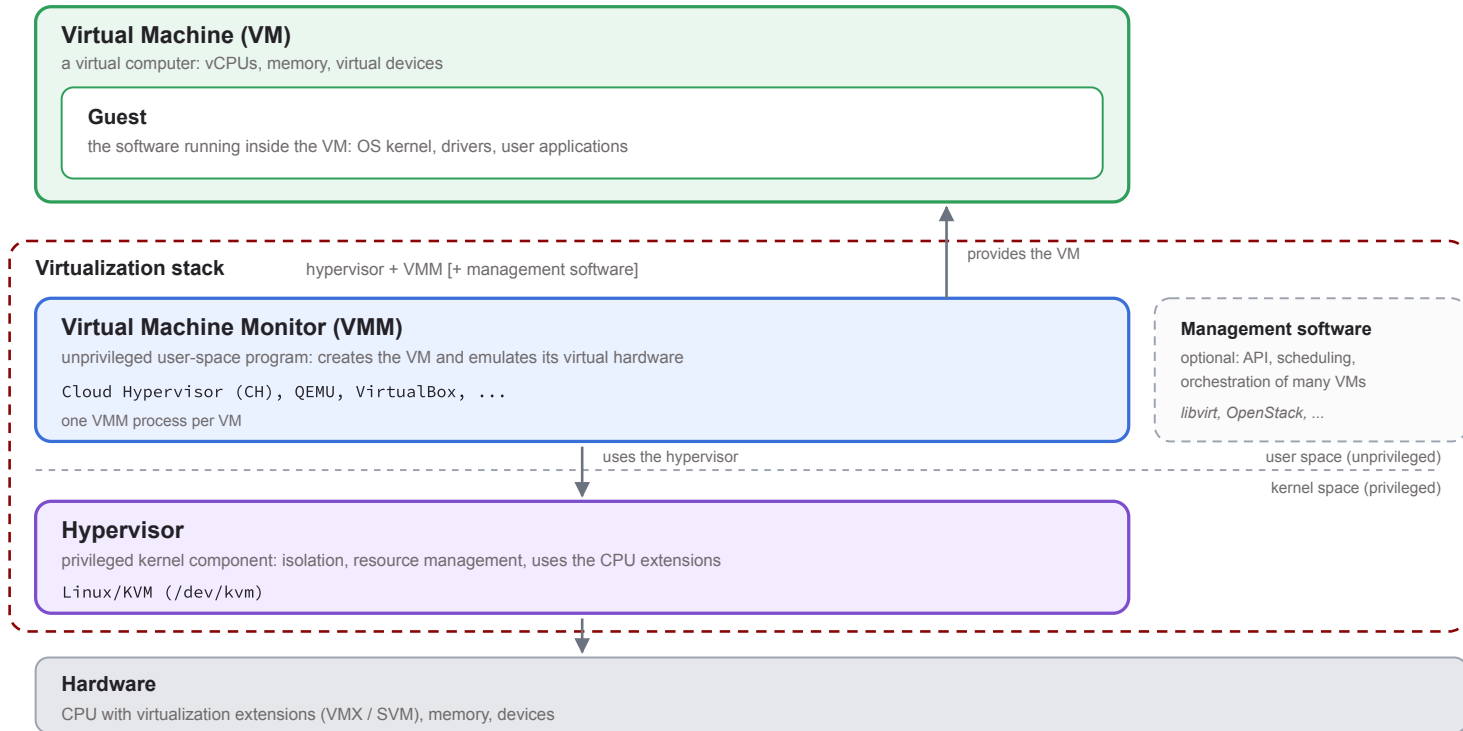
¹ the software (operating system) running in the VM

3.2 What is a VM?

- Java Virtual Machine? 🤔
 - Virtual in the sense that it defines an abstract machine (the Java bytecode)
 - A Java Runtime (JRE) implements the Java Virtual Machine + runtime libraries
- Virtual Machine (this talk):
 - Environment like the host but encapsulated
 - Abstracting the host hardware to create multiple execution environments
 - Behave like the normal host computing platform for the guest¹
- **Hardware-assisted virtualization** (Hardware-accelerated VM)
 - Hardware (CPU) offers features to run guests directly on the CPU
(Alternative would be emulation → slow)
 - For example: Intel VMX, AMD SVM

¹ the software (operating system) running in the VM

Who is who in a virtualization stack



3.3 Isolation of Workloads: Containers vs. VMs

Why VMs when we have Containers?!

3.3 Isolation of Workloads: Containers vs. VMs

Why VMs when we have Containers?!

- Container (e.g., Docker / OCI):

Isolation boundary is enforced by the trusted shared host kernel

3.3 Isolation of Workloads: Containers vs. VMs

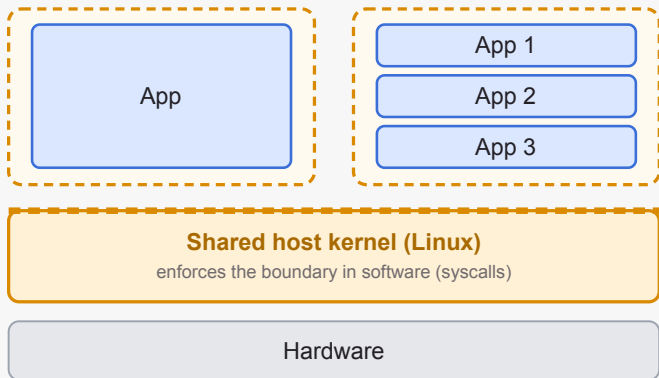
Why VMs when we have Containers?!

- Container (e.g., Docker / OCI):
Isolation boundary is enforced by the trusted shared host kernel
- VM:
Stronger isolation boundary enforced by hypervisor with hardware support

Containers vs. VMs: who enforces isolation?

☐ Customer responsibility

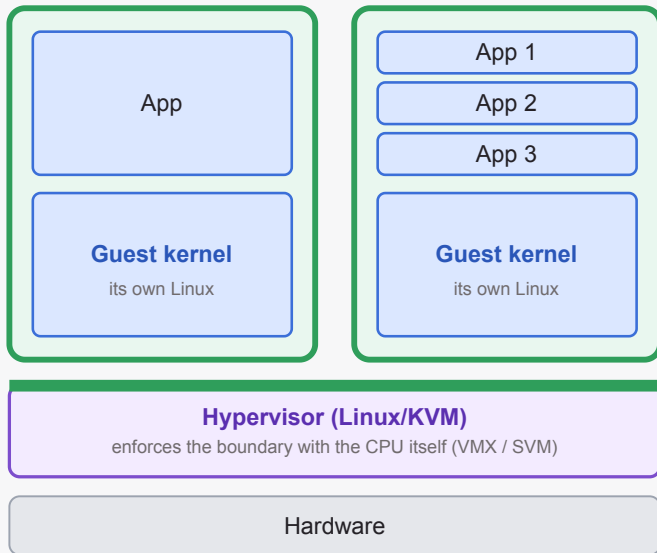
Containers



Attack surface: the whole Linux syscall interface

Lightweight: just the app

Virtual machines



Attack surface: a handful of virtual devices

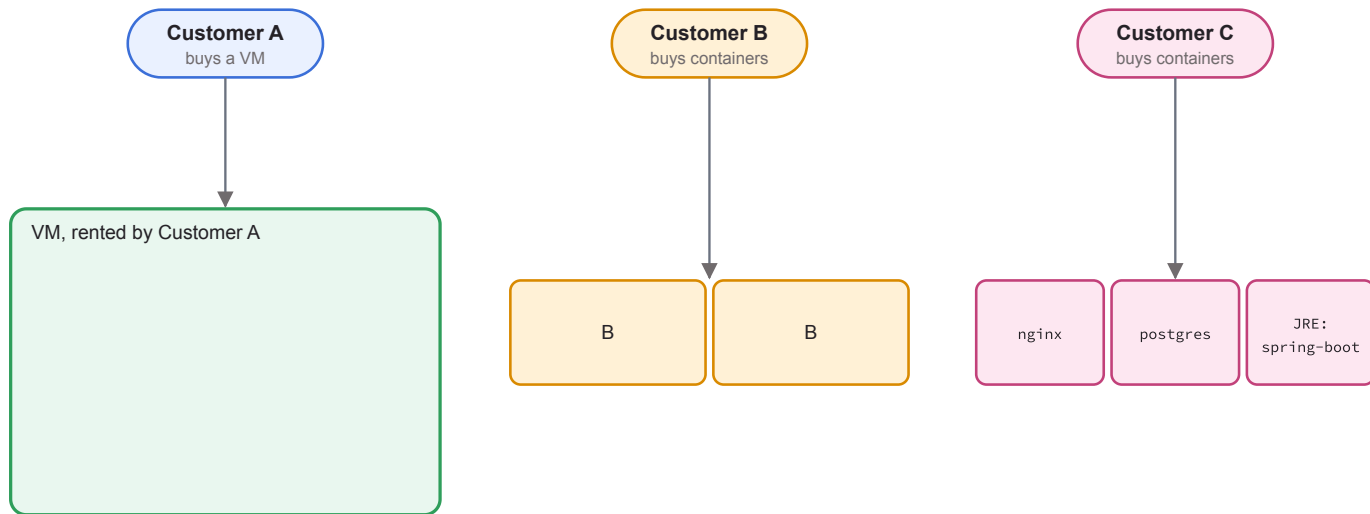
Heavier: app + its own kernel

3.4 Containers and VMs in a Typical Cloud Setup

When to use what?

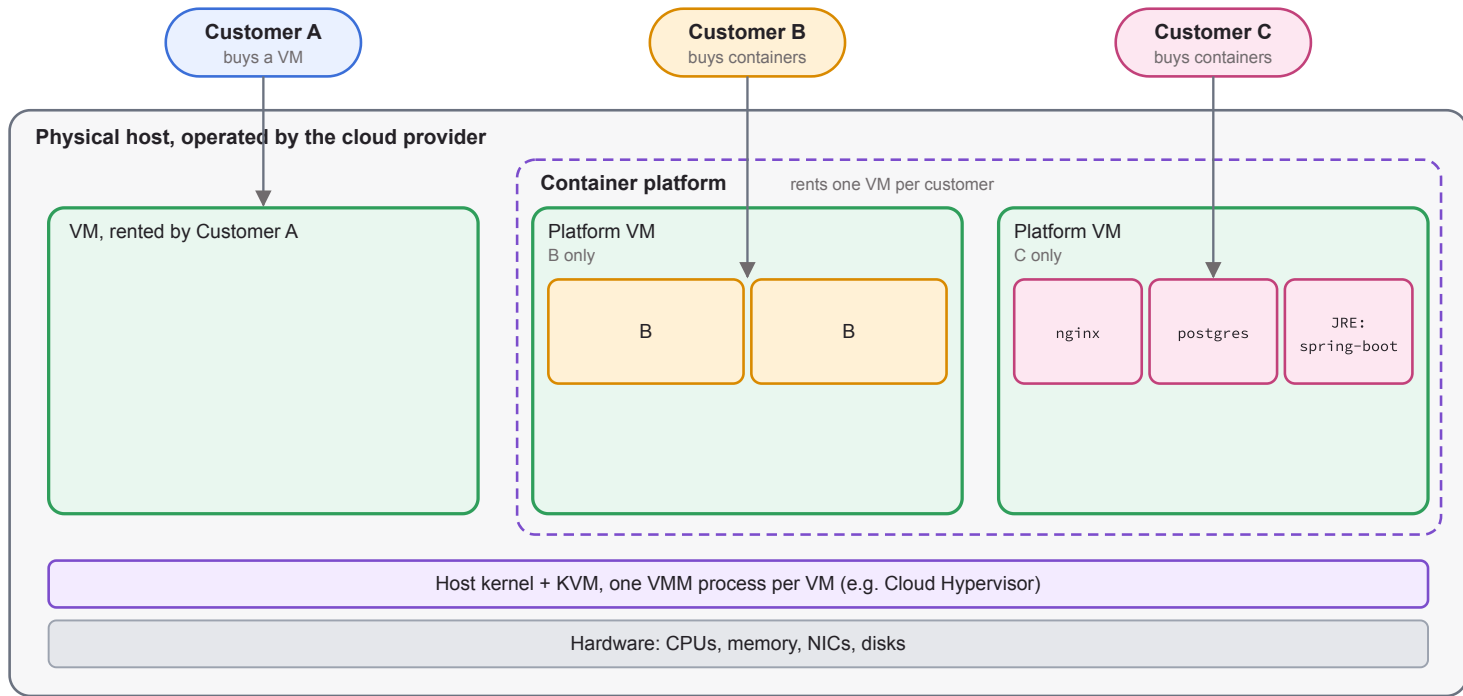
Let's have a look at the following figure:

What customers buy vs. where it actually runs



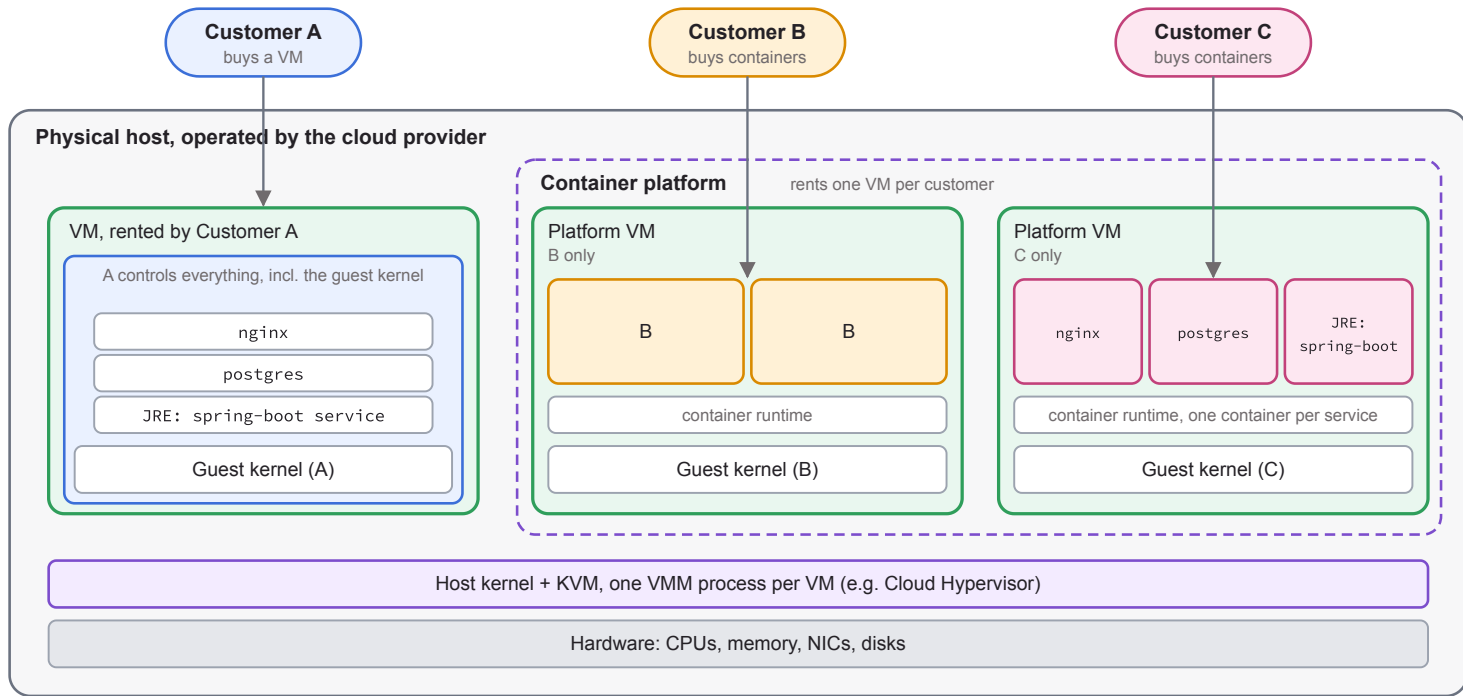
What each customer orders.

What customers buy vs. where it actually runs



Where it actually runs: every customer ends up in a VM of their own.

What customers buy vs. where it actually runs



Container customers never see a VM, yet each gets one (in a typical setup): the VM boundary separates customers.

3.5 Why VMs?

3.5 Why VMs?

- For infrastructure providers

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)
 - Utilize host hardware (€ / \$)

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)
 - Utilize host hardware (€ / \$)
- For developers

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)
 - Utilize host hardware (€ / \$)
- For developers
 - Bring your own OS (and kernel/drivers)

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)
 - Utilize host hardware (€ / \$)
- For developers
 - Bring your own OS (and kernel/drivers)
 - An environment that can be easily deployed

3.5 Why VMs?

- For infrastructure providers
 - Run workloads with strict isolation on shared hardware (multi tenancy)
 - Utilize host hardware (€ / \$)
- For developers
 - Bring your own OS (and kernel/drivers)
 - An environment that can be easily deployed
 - Or messed up and redeployed

3.6 What Is a VM (Technically)?

From the perspective of the host platform (hypervisor).

3.6 What Is a VM (Technically)?

From the perspective of the host platform (hypervisor).

- One VM $\leftrightarrow$ One VMM¹ process

¹ one instance of Cloud Hypervisor (or QEMU, VirtualBox, ...)

3.6 What Is a VM (Technically)?

From the perspective of the host platform (hypervisor).

- One VM $\leftrightarrow$ One VMM¹ process
- One thread per vCPU²

¹ one instance of Cloud Hypervisor (or QEMU, VirtualBox, ...)

² *virtual CPU*: a CPU of the VM - for the guest it looks and behaves like a real one

3.6 What Is a VM (Technically)?

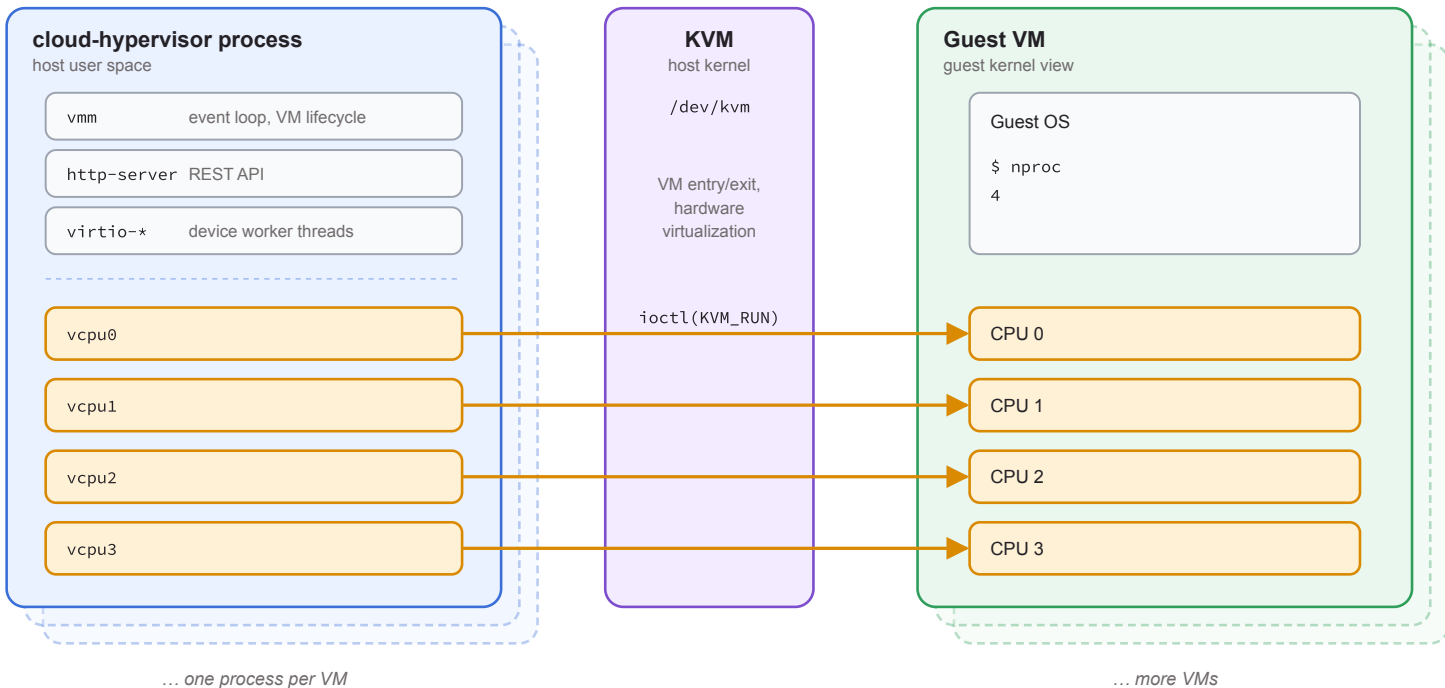
From the perspective of the host platform (hypervisor).

- One VM $\leftrightarrow$ One VMM¹ process
- One thread per vCPU²
- Multiple VMs $\rightarrow$ multiple VMM processes $\rightarrow$ all sharing same hypervisor

¹ one instance of Cloud Hypervisor (or QEMU, VirtualBox, ...)

² *virtual CPU*: a CPU of the VM - for the guest it looks and behaves like a real one

One VM = one Cloud Hypervisor process, one vCPU = one host thread



3.7 Detour: How Does Hardware Access Work?

Software talks to a device - on real hardware and in a VM alike.

3.7 Detour: How Does Hardware Access Work?

Software talks to a device - on real hardware and in a VM alike.

- Read/write access to/from physical memory addresses (MMIO¹) or I/O ports²

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

² *I/O ports*: a separate x86 address space, accessed with the `in` and `out` instructions

3.7 Detour: How Does Hardware Access Work?

Software talks to a device - on real hardware and in a VM alike.

- Read/write access to/from physical memory addresses (MMIO¹) or I/O ports²
- These are accesses with **side effects**: they talk to a device

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

² *I/O ports*: a separate x86 address space, accessed with the `in` and `out` instructions

3.8 What Is a VM (Technically)?

From the perspective of the guest (code running on CPU inside VM).

3.8 What Is a VM (Technically)?

From the perspective of the guest (code running on CPU inside VM).

- Normal x86 platform

3.8 What Is a VM (Technically)?

From the perspective of the guest (code running on CPU inside VM).

- Normal x86 platform
- Normal boot flow to initialize the system

3.8 What Is a VM (Technically)?

From the perspective of the guest (code running on CPU inside VM).

- Normal x86 platform
- Normal boot flow to initialize the system
- Accesses hardware as it would in a baremetal system

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

- VMM: Manages virtual hardware and lifecycle of a VM

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

- VMM: Manages virtual hardware and lifecycle of a VM
- Provide at least one vCPU

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

- VMM: Manages virtual hardware and lifecycle of a VM
- Provide at least one vCPU
 - Abstraction of the VMM to handle CPU state in software

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

- VMM: Manages virtual hardware and lifecycle of a VM
- Provide at least one vCPU
 - Abstraction of the VMM to handle CPU state in software
 - VMM uses KVM to put that state into the hardware on VM entry

3.9 What does a VMM need to create a VM?

The thing that runs your VM. Simplified.

- VMM: Manages virtual hardware and lifecycle of a VM
- Provide at least one vCPU
 - Abstraction of the VMM to handle CPU state in software
 - VMM uses KVM to put that state into the hardware on VM entry
- Configures each vCPUs registers as a real CPU would execute code
(The hardware will then set the registers accordingly and execute guest code)

3.9 What does a VMM need to create a VM?

Simplified.

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:
 - Accessing guest RAM just works

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:
 - Accessing guest RAM just works
 - Accessing an MMIO¹ region (a virtual device) leaves the VM ("VM exit")

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:
 - Accessing guest RAM just works
 - Accessing an MMIO¹ region (a virtual device) leaves the VM ("VM exit")
- On such a VM exit:

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

3.9 What does a VMM need to create a VM?

Simplified.

- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:
 - Accessing guest RAM just works
 - Accessing an MMIO¹ region (a virtual device) leaves the VM ("VM exit")
- On such a VM exit:
 - The hypervisor handles it if it can, otherwise the VMM does

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

3.9 What does a VMM need to create a VM?

Simplified.

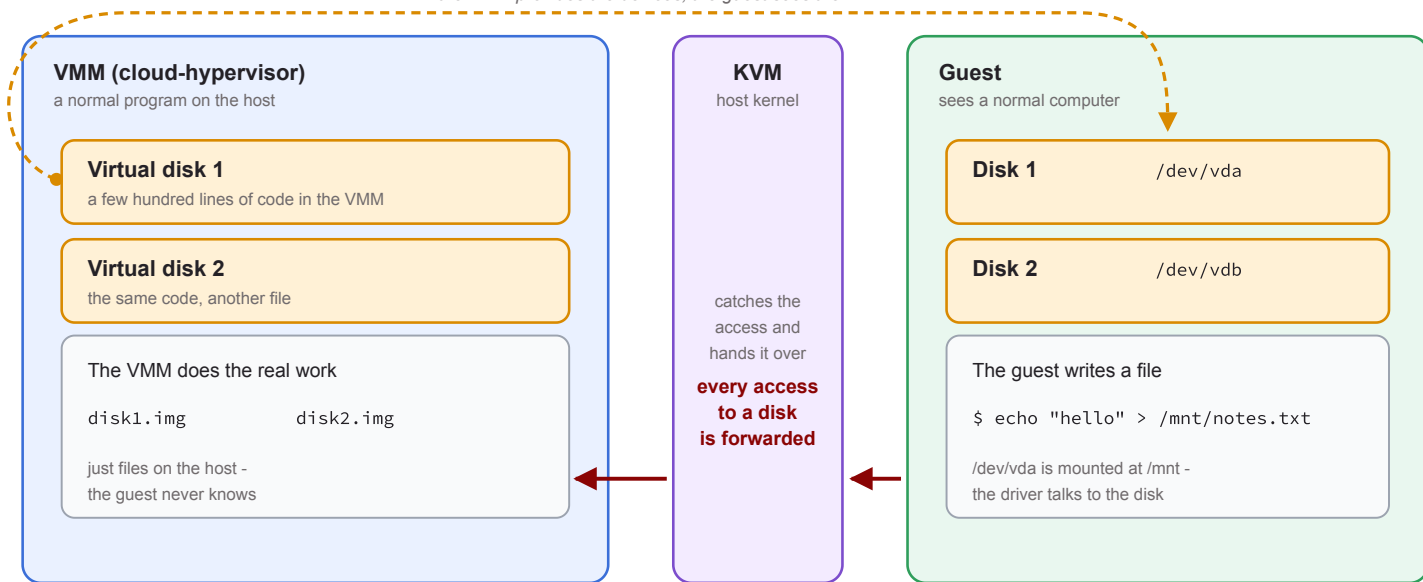
- Add (virtual) hardware, e.g. to the PCI bus
- Configures the guest physical memory space so that:
 - Accessing guest RAM just works
 - Accessing an MMIO¹ region (a virtual device) leaves the VM ("VM exit")
- On such a VM exit:
 - The hypervisor handles it if it can, otherwise the VMM does
 - Afterwards, the vCPU thread enters the VM again

¹ *Memory-Mapped I/O*: device registers mapped into the physical address space

The VMM provides the virtual hardware - the guest just uses it

Example: two virtual disks

the VMM provides the devices, the guest sees them



Virtual hardware is just software in the VMM.

3.10 Linux KVM

Kernel-based Virtual Machine

3.10 Linux KVM

Kernel-based Virtual Machine

- Part of Linux (since 2.6.20 in 2007)

3.10 Linux KVM

Kernel-based Virtual Machine

- Part of Linux (since 2.6.20 in 2007)
- Hypervisor in Linux kernel

3.10 Linux KVM

Kernel-based Virtual Machine

- Part of Linux (since 2.6.20 in 2007)
- Hypervisor in Linux kernel
- Abstraction layer:

3.10 Linux KVM

Kernel-based Virtual Machine

- Part of Linux (since 2.6.20 in 2007)
- Hypervisor in Linux kernel
- Abstraction layer:
 - Hardware virtualization (Intel VMX, AMD SVM)

3.10 Linux KVM

Kernel-based Virtual Machine

- Part of Linux (since 2.6.20 in 2007)
- Hypervisor in Linux kernel
- Abstraction layer:
 - Hardware virtualization (Intel VMX, AMD SVM)
 - vCPU in VMM → Linux thread → physical CPU executing guest code

3.11 CH vs. QEMU vs. VirtualBox vs. VMware

VMMs and Hypervisors in comparison.

Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.

Linux / KVM

Hardware

CPU with virtualization extensions (VMX / SVM)

VirtualBox

Hardware

CPU with virtualization extensions (VMX / SVM)

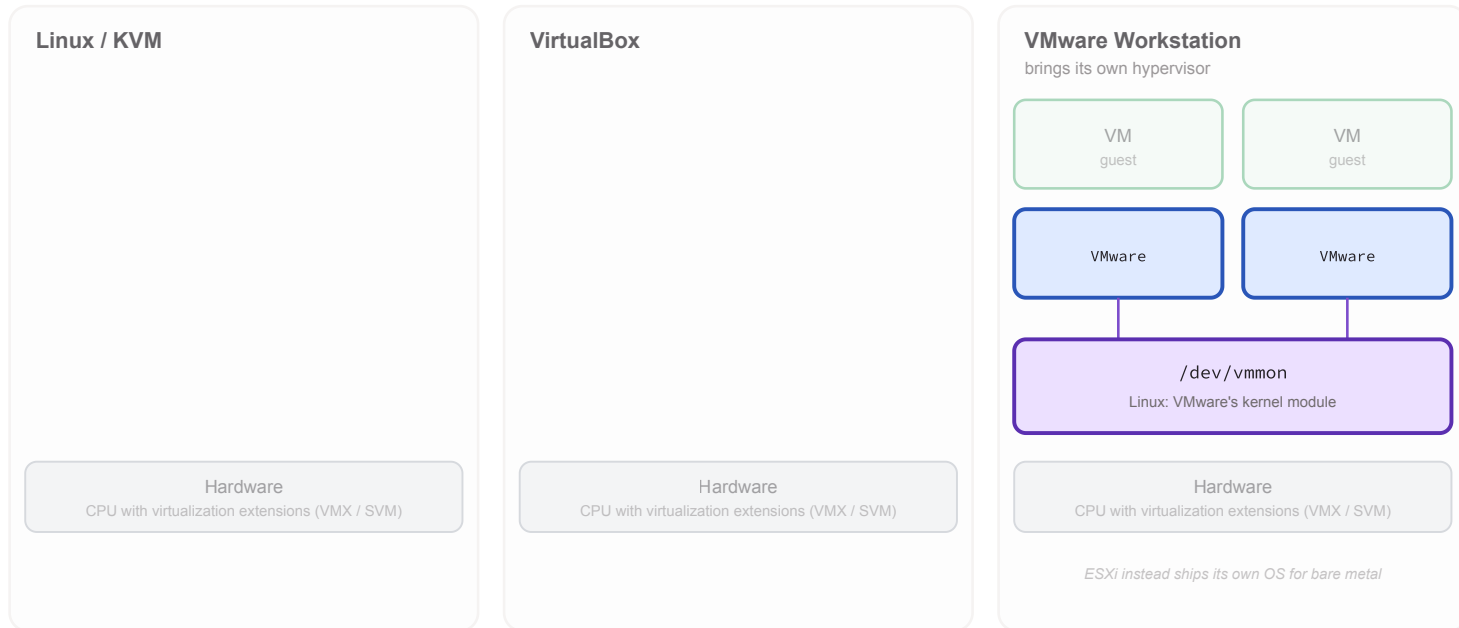
VMware Workstation

Hardware

CPU with virtualization extensions (VMX / SVM)

Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.



Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.

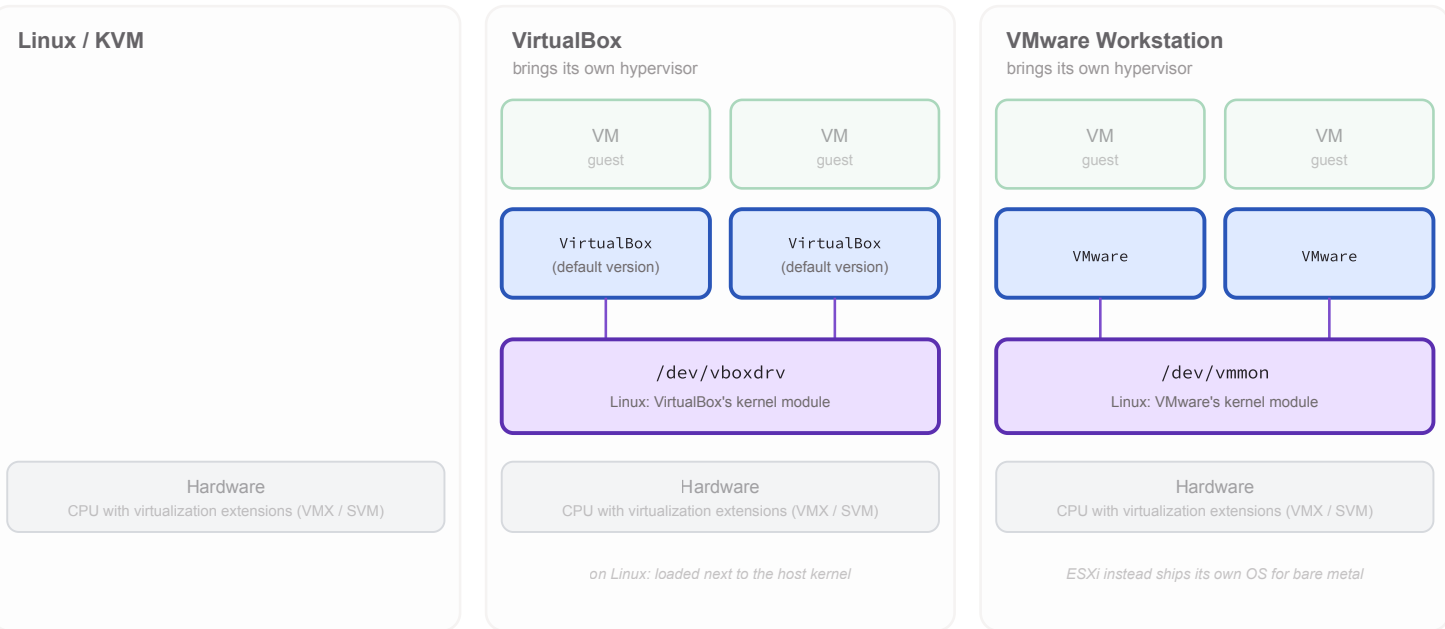
Linux / KVM

VirtualBox

brings its own hypervisor

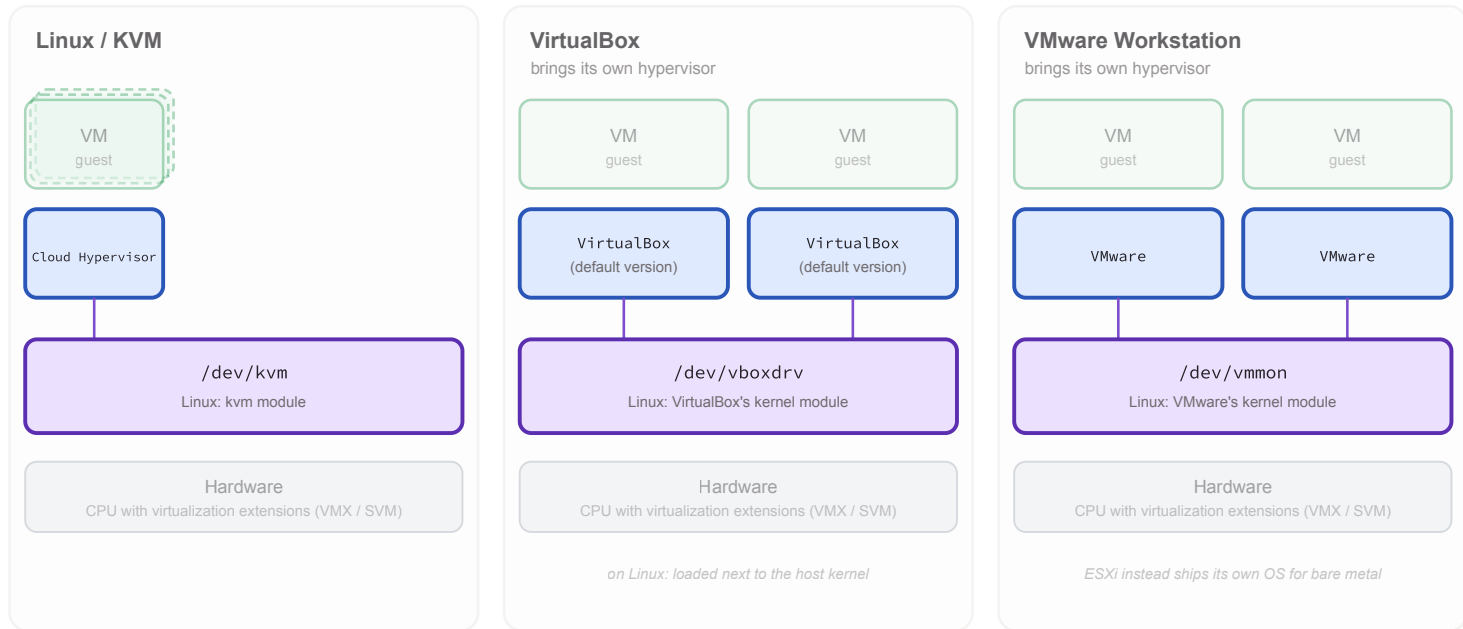
VMware Workstation

brings its own hypervisor



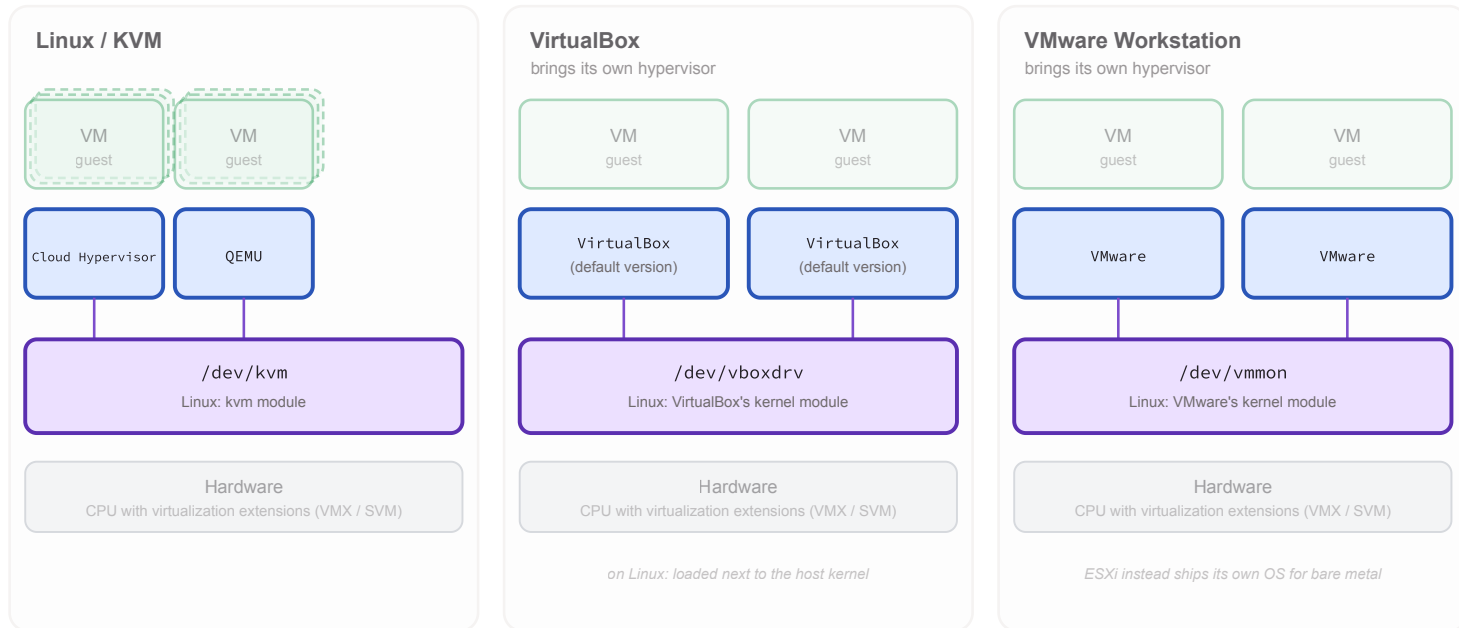
Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.



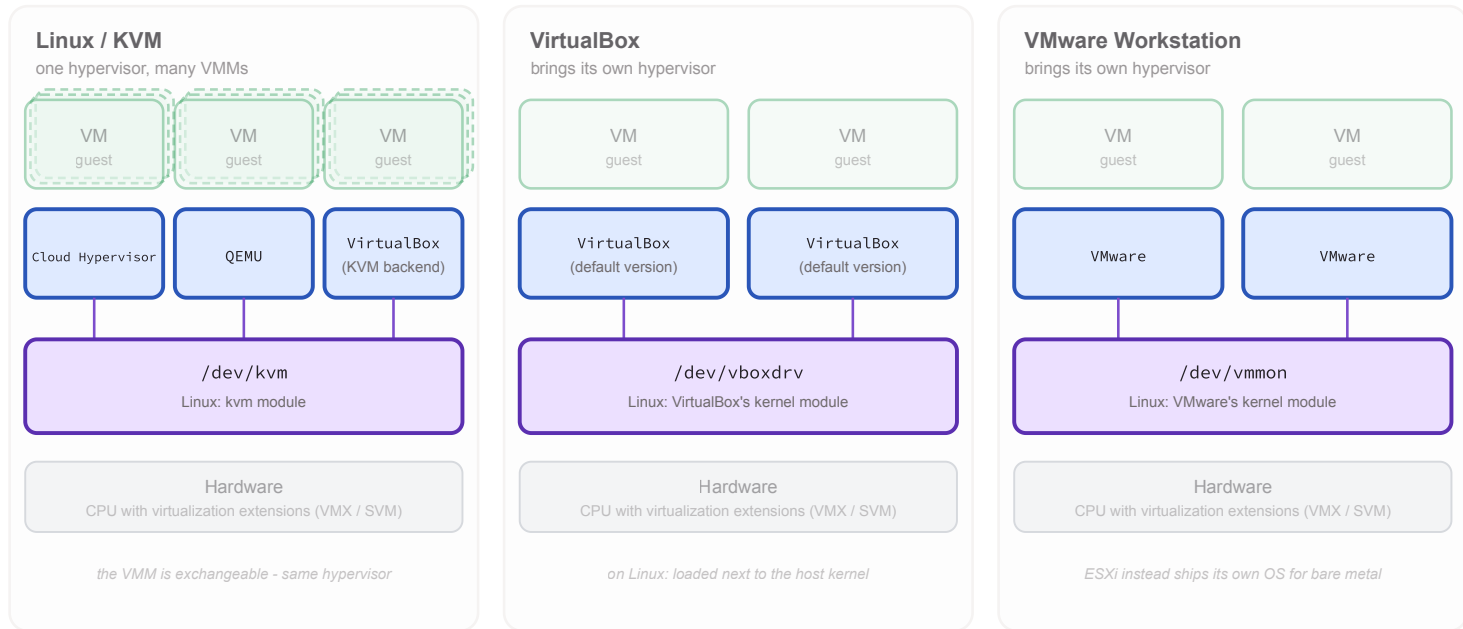
Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.



Always the same shape: a VMM in user space, a hypervisor in the kernel

Only the parts differ.



With KVM the hypervisor is shared and the VMM is just a program - VirtualBox and VMware ship both halves themselves.

3.12 Recap

3.12 Recap

- VM: isolated computing environment, behaving like the host platform

3.12 Recap

- VM: isolated computing environment, behaving like the host platform
 - Easy to deploy software to (and mess-up and retry) compared to full real hardware

3.12 Recap

- VM: isolated computing environment, behaving like the host platform
 - Easy to deploy software to (and mess-up and retry) compared to full real hardware
- VM: VMM + Hypervisor
 - Cloud Hypervisor + Linux/KVM

3.12 Recap

- VM: isolated computing environment, behaving like the host platform
 - Easy to deploy software to (and mess-up and retry) compared to full real hardware
- VM: VMM + Hypervisor
 - Cloud Hypervisor + Linux/KVM
- 1 host thread → 1 vCPU → 1 CPU in VM

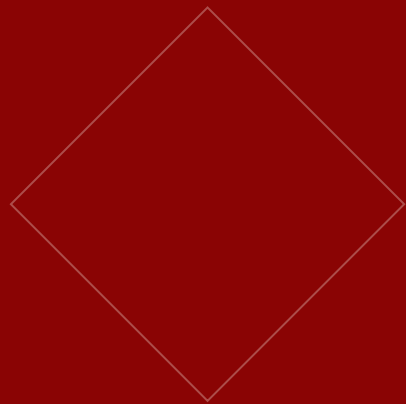
3.12 Recap

- VM: isolated computing environment, behaving like the host platform
 - Easy to deploy software to (and mess-up and retry) compared to full real hardware
- VM: VMM + Hypervisor
 - Cloud Hypervisor + Linux/KVM
- 1 host thread → 1 vCPU → 1 CPU in VM
- Most implementation work happens inside the VMM

3.12 Recap

- VM: isolated computing environment, behaving like the host platform
 - Easy to deploy software to (and mess-up and retry) compared to full real hardware
- VM: VMM + Hypervisor
 - Cloud Hypervisor + Linux/KVM
- 1 host thread → 1 vCPU → 1 CPU in VM
- Most implementation work happens inside the VMM
- We'll focus in Cloud Hypervisor (VMM) with Linux/KVM (hypervisor) in the following

4. Cloud Hypervisor



4.1 What is Cloud Hypervisor?

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads
 - Almost no legacy devices, 64-bit only, virtio-based devices¹

¹ *virtio* = specification for (virtual) devices: guest knows that it is virtualized and uses drivers made for that

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads
 - Almost no legacy devices, 64-bit only, virtio-based devices¹
 - No display/graphics model, only basic virtual hardware

¹ *virtio* = specification for (virtual) devices: guest knows that it is virtualized and uses drivers made for that

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads
 - Almost no legacy devices, 64-bit only, virtio-based devices¹
 - No display/graphics model, only basic virtual hardware
 - Live migration

¹ *virtio* = specification for (virtual) devices: guest knows that it is virtualized and uses drivers made for that

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads
 - Almost no legacy devices, 64-bit only, virtio-based devices¹
 - No display/graphics model, only basic virtual hardware
 - Live migration
- ~150.000 SLOC (including tests, scripts, etc.)

¹ *virtio* = specification for (virtual) devices: guest knows that it is virtualized and uses drivers made for that

4.1 What is Cloud Hypervisor?

- *Cloud Hypervisor is an open source **Virtual Machine Monitor (VMM)** that runs on top of the **KVM hypervisor** - Naming is complicated 🙄*
- Modern VMM written in Rust modern Cloud workloads
 - Almost no legacy devices, 64-bit only, virtio-based devices¹
 - No display/graphics model, only basic virtual hardware
 - Live migration
- ~150.000 SLOC (including tests, scripts, etc.)
- Started 2019 in Intel, now mainly driven by Microsoft, Meta, Crusoe, Cyberus Technology

¹ *virtio* = specification for (virtual) devices: guest knows that it is virtualized and uses drivers made for that

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

- Rust is a mature language: learnings from decades of pain with C++ as language and corresponding toolchains

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

- Rust is a mature language: learnings from decades of pain with C++ as language and corresponding toolchains
- Fearless contributions via streamlined unit testing and high code quality

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

- Rust is a mature language: learnings from decades of pain with C++ as language and corresponding toolchains
- Fearless contributions via streamlined unit testing and high code quality
- Fantastic IDE support

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

- Rust is a mature language: learnings from decades of pain with C++ as language and corresponding toolchains
- Fearless contributions via streamlined unit testing and high code quality
- Fantastic IDE support
- Easy onboarding of new engineers

4.2 A VMM in Rust?

About developer productivity, safety, and robustness out of this world.

- Rust is a mature language: learnings from decades of pain with C++ as language and corresponding toolchains
- Fearless contributions via streamlined unit testing and high code quality
- Fantastic IDE support
- Easy onboarding of new engineers
- Easy to approach language for complex domain: encapsulate complexity with ease

4.3 Involvement of Cyberus Technology

4.3 Involvement of Cyberus Technology

- We are building major parts of our business around this open-source project

4.3 Involvement of Cyberus Technology

- We are building major parts of our business around this open-source project
- "Upstream first" policy: our work goes into the public project first,
no private forks

4.3 Involvement of Cyberus Technology

- We are building major parts of our business around this open-source project
- "Upstream first" policy: our work goes into the public project first, no private forks
- We are among the top contributors for the past 18 months

4.4 What We Are Working On

Fields the whole company is involved in.

4.4 What We Are Working On

Fields the whole company is involved in.

- Live migration¹

¹ Process of migrating a VM from host A to B while it keeps running

4.4 What We Are Working On

Fields the whole company is involved in.

- Live migration¹
- VM and vCPU lifecycle management

¹ Process of migrating a VM from host A to B while it keeps running

4.4 What We Are Working On

Fields the whole company is involved in.

- Live migration¹
- VM and vCPU lifecycle management
- Performance improvements

¹ Process of migrating a VM from host A to B while it keeps running

4.4 What We Are Working On

Fields the whole company is involved in.

- Live migration¹
- VM and vCPU lifecycle management
- Performance improvements
- Testing

¹ Process of migrating a VM from host A to B while it keeps running

4.4 What We Are Working On

Fields the whole company is involved in.

- Live migration¹
- VM and vCPU lifecycle management
- Performance improvements
- Testing

My role: upstream maintainer of live migration

¹ Process of migrating a VM from host A to B while it keeps running

4.5 Hands-on Cloud Hypervisor

Spawning VMs and debugging!

4.5 Hands-on Cloud Hypervisor

Spawning VMs and debugging!

- CLI and spawning a VM

4.5 Hands-on Cloud Hypervisor

Spawning VMs and debugging!

- CLI and spawning a VM
- Networking into the guest

4.5 Hands-on Cloud Hypervisor

Spawning VMs and debugging!

- CLI and spawning a VM
- Networking into the guest
- Some Live Debugging & daily workflows

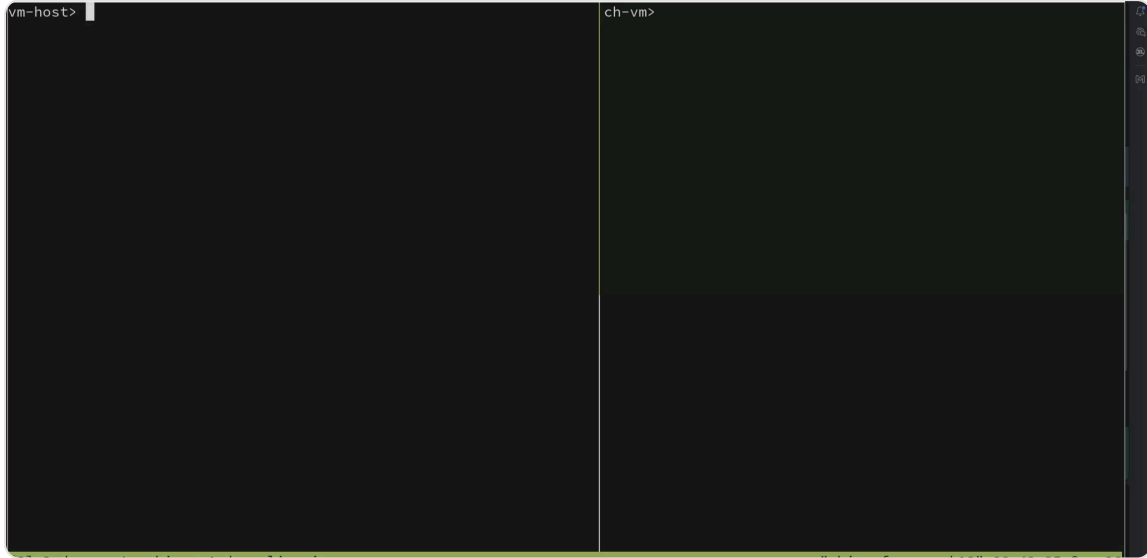
4.5 Hands-on Cloud Hypervisor

Spawning VMs and debugging!

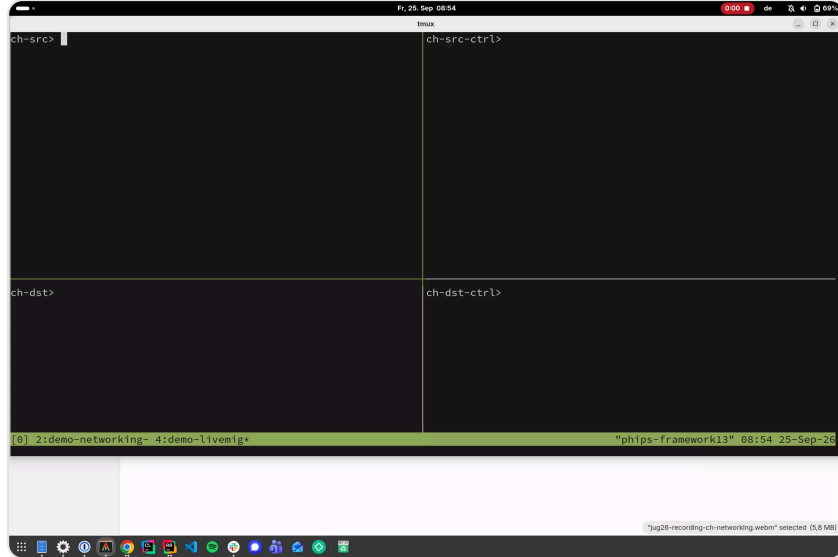
- CLI and spawning a VM
- Networking into the guest
- Some Live Debugging & daily workflows
- Live Migration

DEMO TIME

4.6 Recording: Networking into the Guest



4.7 Recording: Live Migration



4.8 My Development Flow

4.8 My Development Flow

- Unit tests

4.8 My Development Flow

- Unit tests
- Live VM testing with direct kernel boot

4.8 My Development Flow

- Unit tests
- Live VM testing with direct kernel boot
 - I have tooling to build minimal Linux kernel and minimal initrd
(My own tiny Linux distribution)

4.8 My Development Flow

- Unit tests
- Live VM testing with direct kernel boot
 - I have tooling to build minimal Linux kernel and minimal initrd
(My own tiny Linux distribution)
 - Sufficient for 99% of my testing cases

4.8 My Development Flow

- Unit tests
- Live VM testing with direct kernel boot
 - I have tooling to build minimal Linux kernel and minimal initrd
(My own tiny Linux distribution)
 - Sufficient for 99% of my testing cases
- We have more mature integration test suites as well

5. Digital Sovereignty for Core Digital Infrastructure

Virtualization made in Saxony by Cyberus Technology



5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²
- You get: start VMs + API socket: pause, resume, hotplug, migrate, ...

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²
- You get: start VMs + API socket: pause, resume, hotplug, migrate, ...
- You bring: Broader VM management (storage, networking, lifecycle, customer → VM)

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²
- You get: start VMs + API socket: pause, resume, hotplug, migrate, ...
- You bring: Broader VM management (storage, networking, lifecycle, customer → VM)
- We provide `libvirt`³ integration

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

³ [GitHub: cyberus-technology/libvirt.git](https://github.com/cyberus-technology/libvirt.git)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²
- You get: start VMs + API socket: pause, resume, hotplug, migrate, ...
- You bring: Broader VM management (storage, networking, lifecycle, customer → VM)
- We provide `libvirt`³ integration
- With trivial patches to OpenStack, you can "just use it" via libvirt backend

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

³ [GitHub: cyberus-technology/libvirt.git](https://github.com/cyberus-technology/libvirt.git)

5.1 How to Use Our Virtualization Stack

For production deployments with service: Cyberus Hypervisor. Else: Open Source

- Use stock Linux with `CONFIG_KVM=y` (default on almost all distros)
- Use upstream Cloud Hypervisor¹ or check out our development version²
- You get: start VMs + API socket: pause, resume, hotplug, migrate, ...
- You bring: Broader VM management (storage, networking, lifecycle, customer → VM)
- We provide `libvirt`³ integration
- With trivial patches to OpenStack, you can "just use it" via libvirt backend
- For VS-NfD / NATO Restricted: commercial Cyberus Hypervisor package

¹ [GitHub: cloud-hypervisor/cloud-hypervisor](https://github.com/cloud-hypervisor/cloud-hypervisor)

² [GitHub: cyberus-technology/cloud-hypervisor](https://github.com/cyberus-technology/cloud-hypervisor)

³ [GitHub: cyberus-technology/libvirt.git](https://github.com/cyberus-technology/libvirt.git)

5.2 Virtualization Made in Saxony

CYBERUS
TECHNOLOGY

5.2 Virtualization Made in Saxony

CYBERUS
TECHNOLOGY

- No need for big US-based companies

5.2 Virtualization Made in Saxony

CYBERUS
TECHNOLOGY

- No need for big US-based companies
- We build core digital infrastructure here in Dresden

5.2 Virtualization Made in Saxony

- No need for big US-based companies
- We build core digital infrastructure here in Dresden
- **BSI-accredited ("Zulassung") for *VS-NfD* and *NATO Restricted*** 🎉

(Since September 15th 2026 - [BSI Website](#))

5.2 Virtualization Made in Saxony

- No need for big US-based companies
- We build core digital infrastructure here in Dresden
- **BSI-accredited ("Zulassung") for VS-NfD and NATO Restricted** 🎉
(Since September 15th 2026 - [BSI Website](#))
- The only virtualization stack in Germany

5.2 Virtualization Made in Saxony

- No need for big US-based companies
 - We build core digital infrastructure here in Dresden
 - **BSI-accredited ("Zulassung") for *VS-NfD* and *NATO Restricted*** 🇪🇺
- (Since September 15th 2026 - [BSI Website](#))
- The only virtualization stack in Germany
 - We love the technology, and we love doing something for Germany's & Europe's sovereignty

5.2 Virtualization Made in Saxony

- No need for big US-based companies
- We build core digital infrastructure here in Dresden
- **BSI-accredited ("Zulassung") for VS-NfD and NATO Restricted** 🇪🇺

(Since September 15th 2026 - [BSI Website](#))

- The only virtualization stack in Germany
- We love the technology, and we love doing something for Germany's & Europe's sovereignty
- Thank you!



Cloud Hypervisor



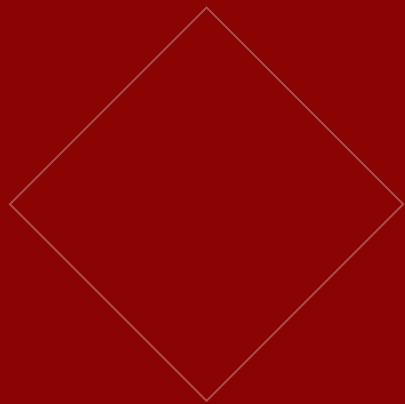
Cyberus Technology



Slides

Backup

Slides for questions and for when the live demo misbehaves.



Linux KVM - The `ioctl` Interface

Kernel-based Virtual Machine

- Entry point to KVM: `/dev/kvm`
 - `open("/dev/kvm")` → KVM system FD
 - `ioctl(kvm_fd, KVM_CREATE_VM, ...)` → VM FD
 - `ioctl(vm_fd, KVM_CREATE_VCPU, ...)` → vCPU FD
 - A vCPU thread calls `ioctl(vcpu_fd, KVM_RUN, 0)`
 - The physical CPU running that thread then executes guest code (using hardware virtualization features)
- That's it :)

Live Demo - Networking

```
# host
sudo ip tuntap add dev tap0 mode tap
sudo ip addr add dev tap0 192.168.200.1/24
sudo ip link set dev tap0 up

# guest
ip addr add dev eth0 192.168.200.2/24
ip link set dev eth0 up
```

Live Demo - Spawning a VM

```
cloud-hypervisor \  
  --kernel /etc/bootitems/linux/kernel_minimal/stable.bzImage \  
  --initramfs /etc/bootitems/linux/initrd_minimal/default \  
  --cmdline "console=ttyS0" \  
  --memory size=2048M,prefault=on \  
  --cpus boot=4 \  
  --serial tty \  
  --console file=/tmp/foo \  
  --api-socket path=/tmp/chv1.sock \  
  --event-monitor path=/tmp/events_ch.txt \  
  --net tap=tap0
```