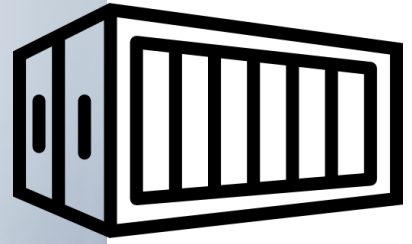
CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Container

(2) Betriebssystem-Virtualisierung



creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 8

Standardisierung von Deployment Units (Container)

8 Standardisierung von Deployment Units

8.1 Hintergrund (PaaS)

8.2 Betriebssystem-Virtualisierung

8.3 Container Runtime Environments
- Kernel-Namespaces
- Process Capabilities
- Control Groups
- Union Filesystems
- High- und Low-Level Container-Laufzeitumgebungen

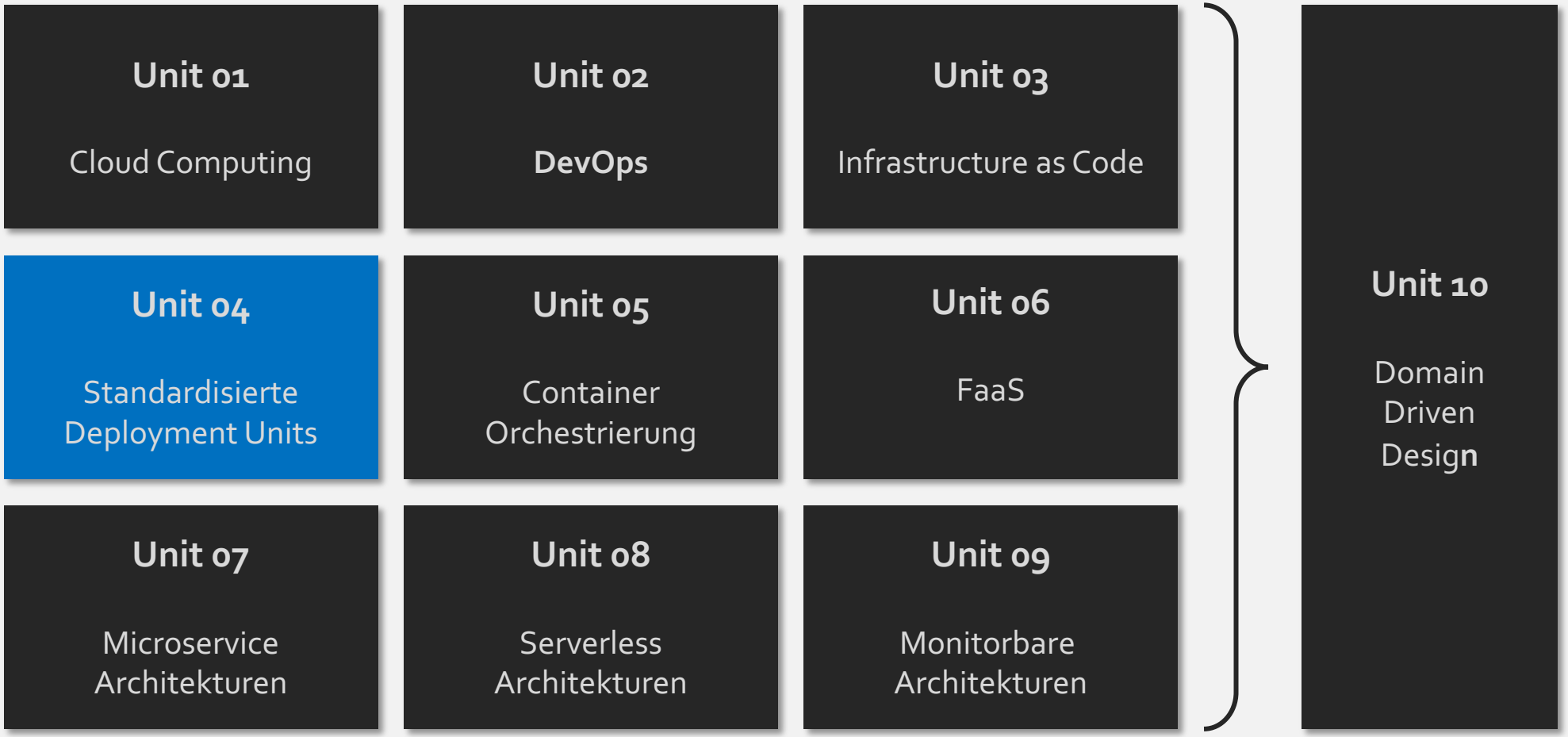
8.4 Bau und Bereitstellung von Container-Images

8.5 Faktoren gut betreibbarer Container
- Codebase
- Abhängigkeiten und Konfigurationen
- Unterstützende Services und Port Binding
- Build-, Release- und Run-Phase
- Horizontale Skalierung über Prozesse
- Umgebungen, Logs und Betrieb

8.6 Zusammenfassung

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

Hintergrund

- Platform as a Service
- Das PaaS-Problem
- Das CaaS-Versprechen

Betriebssystem-Virtualisierung

- OS-Virtualisierung
- Linux-basierte Techniken zur OS-Virtualisierung
- Standardisierung von Deployment-Einheiten => Container

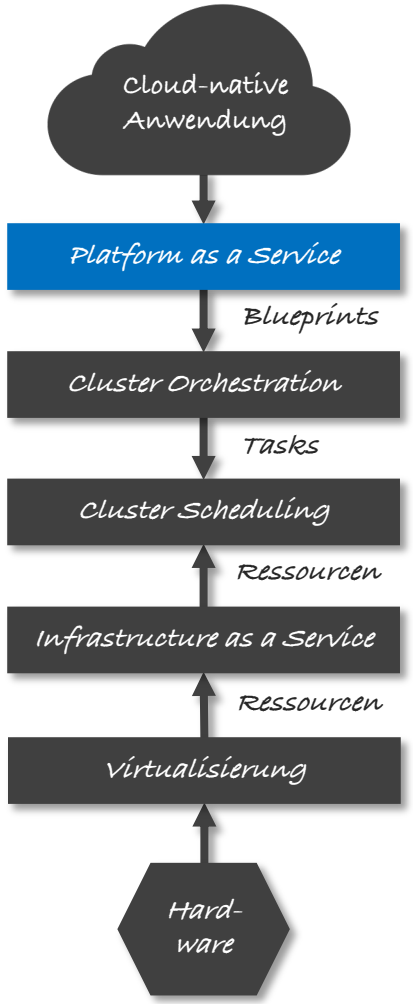
Laufzeitumgebungen für Container

- Container Laufzeitumgebungen und Standards
- Docker
- Image Building und Registries

Container-Pattern

- Container (Anti-)Pattern
- 12-Factor Apps

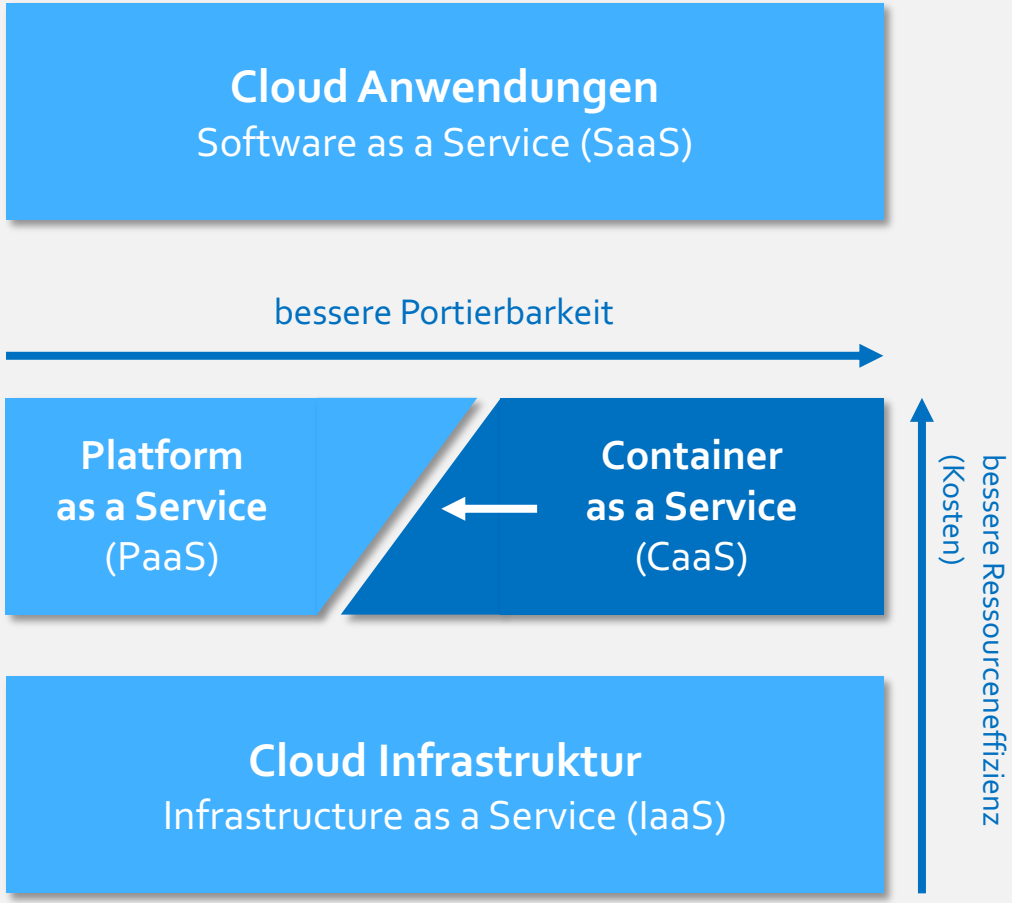
CLOUD NATIVE COMPUTING



VON PAAS ZU CAAS

Container as a Service

Bei CaaS handelt sich um ein Cloud-Computing-Modell, mit dem sich Container-basierte Virtualisierung als Service aus der Cloud nutzen lässt. Anwender von CaaS benötigen keine eigene Infrastruktur für die Container-Virtualisierung und müssen keine eigene Container-Plattform betreiben und managen. Sämtliche Ressourcen für die Virtualisierung wie Rechenleistung, Speicherplatz, Container-Engine und Orchestrierungssoftware stellt der Cloud-Service-Provider zur Verfügung. Es ist möglich, CaaS als Service aus einer Public Cloud oder aus einer Private Cloud zu beziehen.
Container as a Service liegt konzeptionell zwar zwischen den Modellen Infrastructure as a Service (IaaS) und Platform as a Service (PaaS) verdrängt aber im Cloud-native Umfeld allmählich aufgrund besserer Portierbarkeit und besserer Standardisierung das PaaS Modell.



CLOUD NATIVE COMPUTING

CONTAINER

Die grundsätzliche Idee

Anstatt „mundgeblasener" Speziallösungen

Standardisiert ausbringbare und ausführbare Einheiten

- **Container** = Verpackung für Ops Components
- **Standard-Schnittstellen** für Standard-Betriebsprozeduren
- **Einfach zu transportieren**
- **Schnell zu starten**
- **Wenig Performance-Overhead**

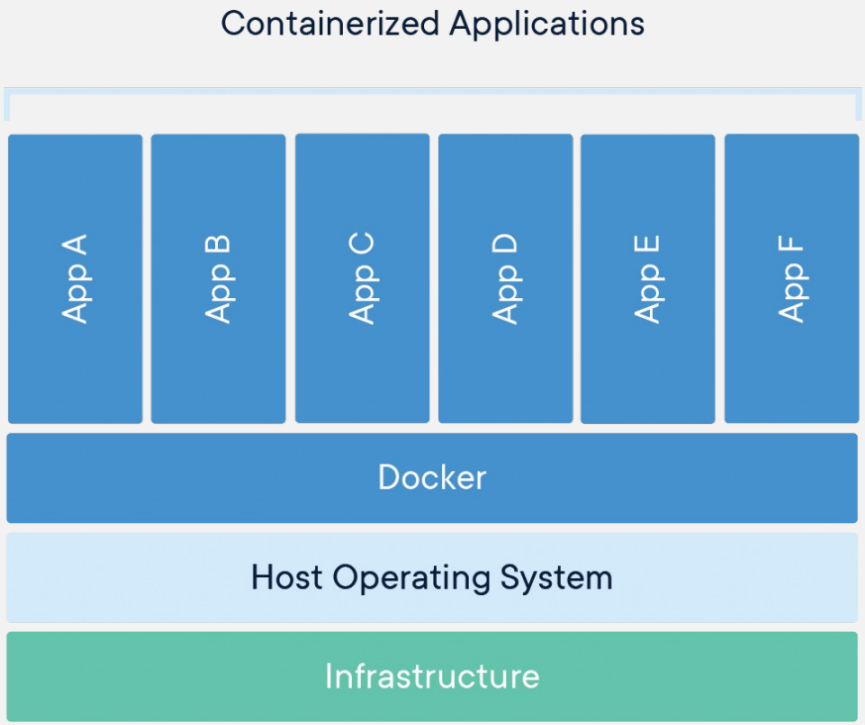
CLOUD NATIVE COMPUTING

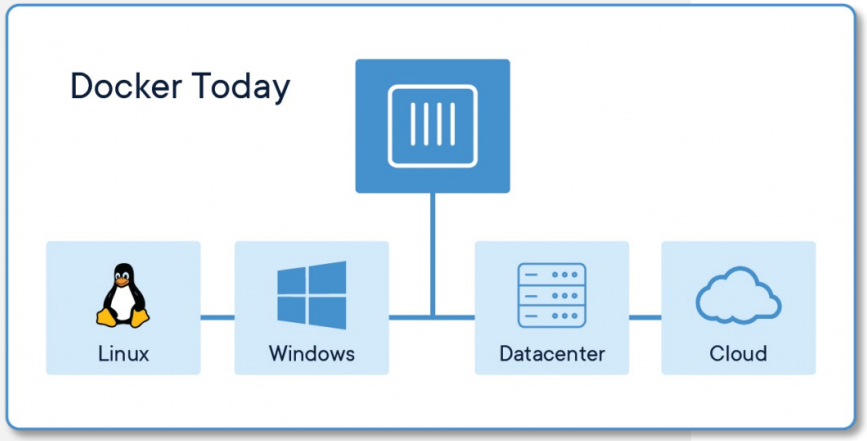
Bei einem Container ist egal was er beinhaltet.

Er kann standardisiert auf Schiffen, Zügen oder Lastern transportiert und verladen werden.

WAS IST EIN CONTAINER?

Eine standardisiert betreibbare Software-Komponente





VIRTUALISIERUNGSARTEN

Wiederholung

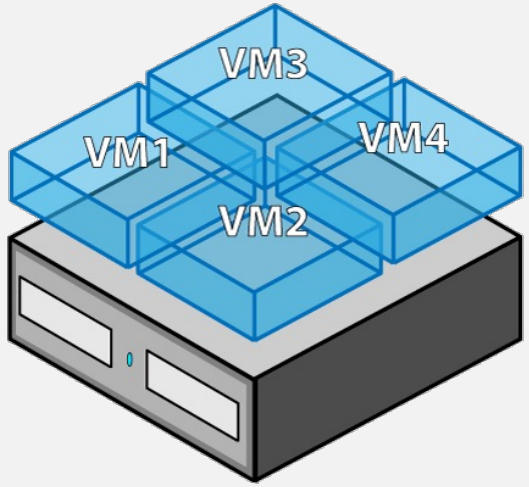
Unter Virtualisierung werden grundsätzlich verschiedene Konzepte und Technologien verstanden.

Virtualisierung von Hardware-Infrastruktur

1. Emulation
2. Voll-Virtualisierung (Typ-2 Virtualisierung)
3. Para-Virtualisierung (Typ-1 Virtualisierung)

Virtualisierung von Software-Infrastruktur

- Betriebssystem-Virtualisierung (Containerization)
- Anwendendungs-Virtualisierung (Runtime, z.B. die Java Virtual Machine)

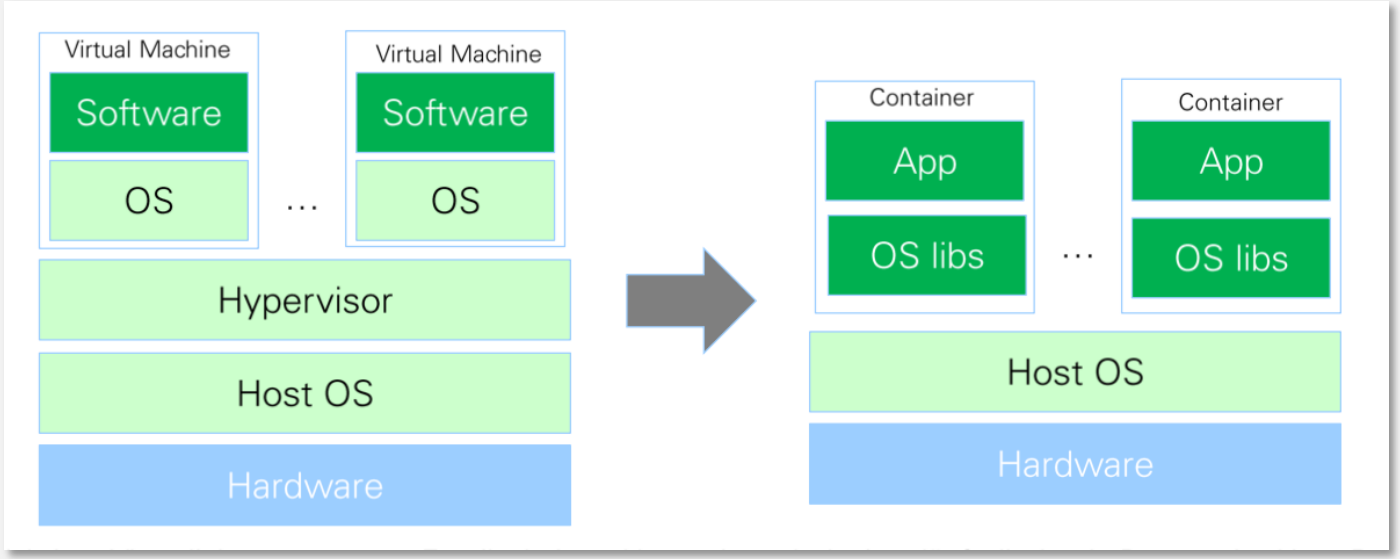


BETRIEBSSYSTEM-VIRTUALISIERUNG

Wiederholung

Es gibt keinen Hypervisor. Jede Applikation läuft direkt als Prozess im Host-Betriebssystem.

Dieser Prozess ist jedoch mittels OS-Mechanismen isoliert.

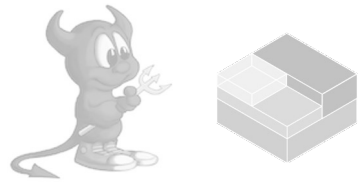


- Free BSD Jails (2000)
- Solaris Zones (2005)
- Linux OpenVZ (2005)
- Linux LXC (2008)
- und mehr …

Docker V0.0.1 release: 2013

- Isolation des Prozesses durch Namespaces (bzgl. CPU, RAM, Disk I/O) und Containments
- Isoliertes Dateisystem
- Eigene Netzwerkschnittstelle
- Startup-Zeit = Startdauer für den ersten Prozess (kein Boot des OS erforderlich!)

Leistungsverlust: CPU-/RAM-Overhead in der Regel nicht messbar (~0%)

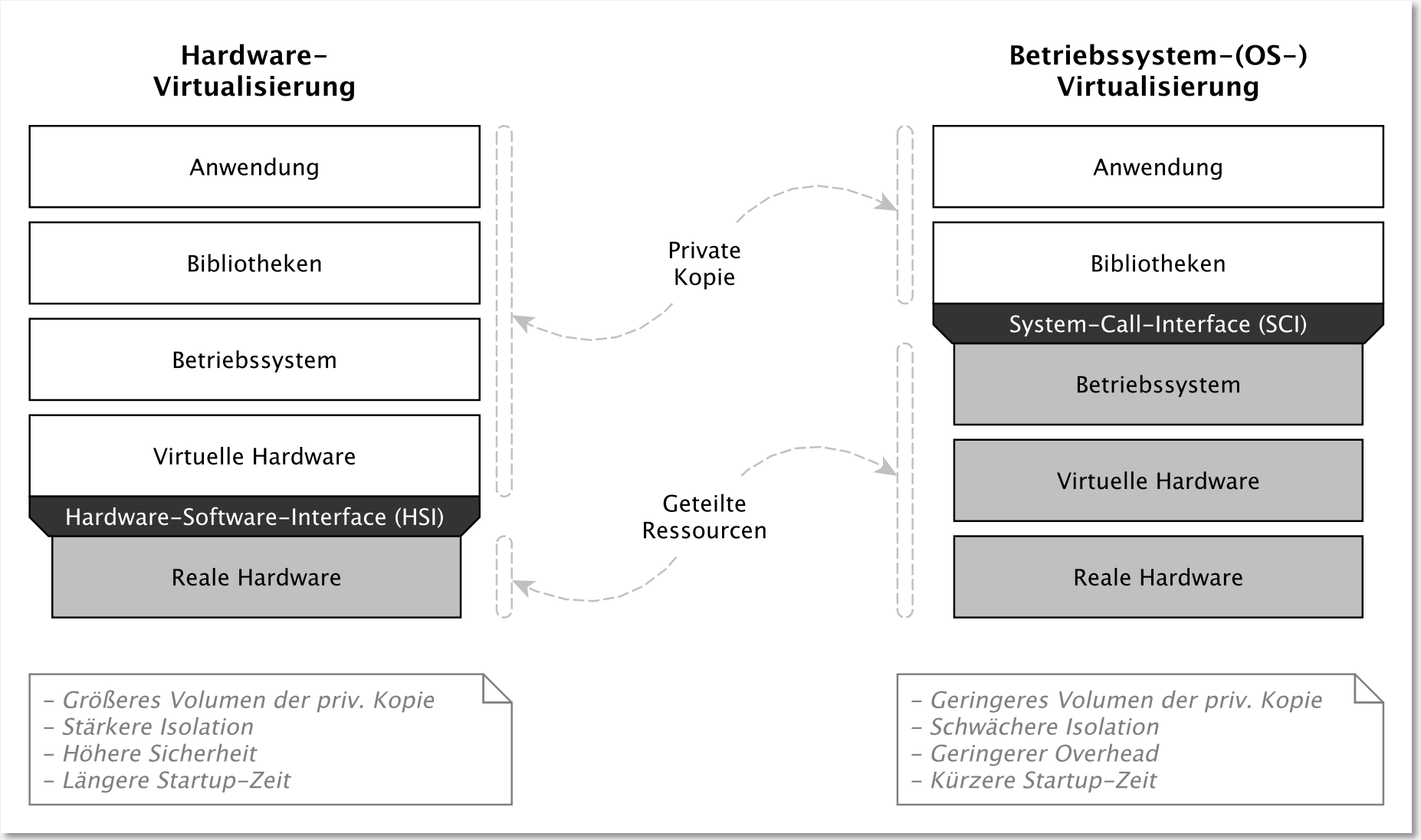


OpenVZ

solaris

HARDWARE- VS. OS-VIRTUALISIERUNG

Wiederholung



OS-VIRTUALISIERUNG

Aus dem Blickwinkel des Linux-Kernels

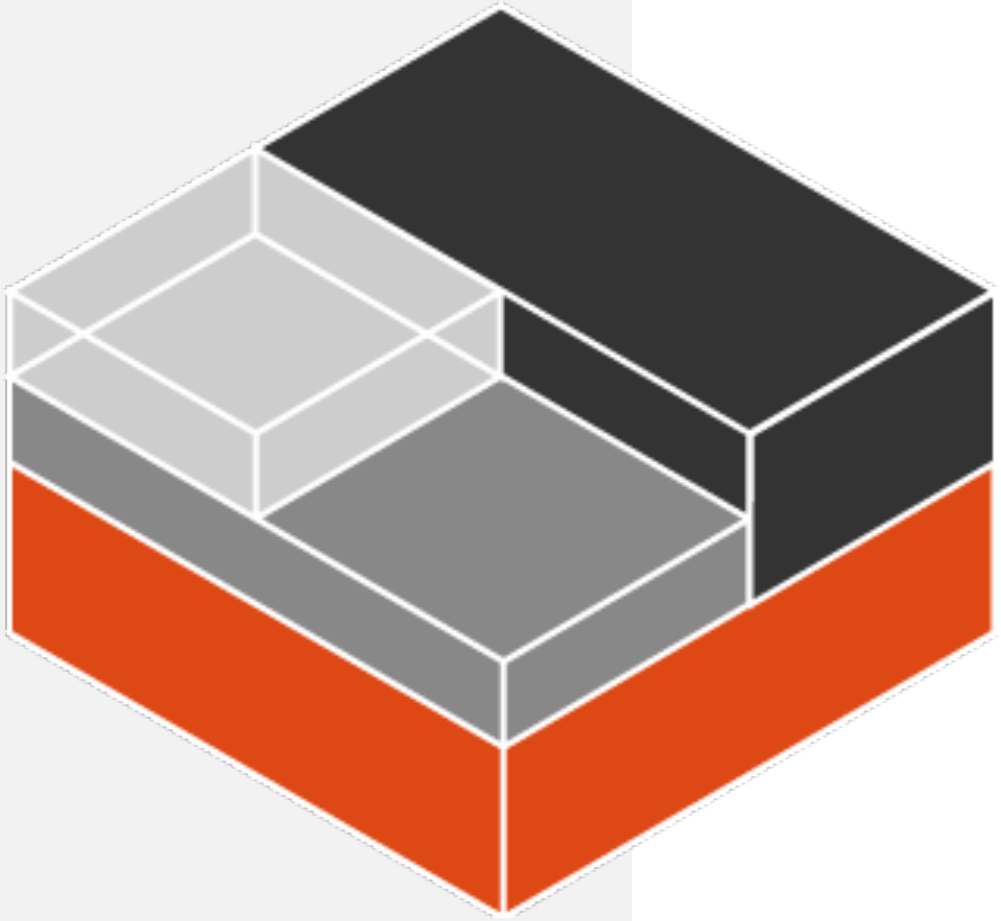
CLOUD NATIVE COMPUTING

Control Groups
Gruppieren von Prozessen (z.B. alle Prozesse einer App, eines Dienstes, etc.) in Gruppen

Kernel Namespaces
Bereitstellung globaler Systemressourcen (z.B. Netzwerk, Volume mounts) als eigene isoliert erscheinende Ressource für alle Prozesse in einem Namespace.

Process Capabilities
Capabilities sind spezielle Attribute im Linux-Kernel, die Prozessen und ausführbaren Binärdateien bestimmte Berechtigungen gewähren, die normalerweise nur Prozessen des Root-Users vorbehalten sind.

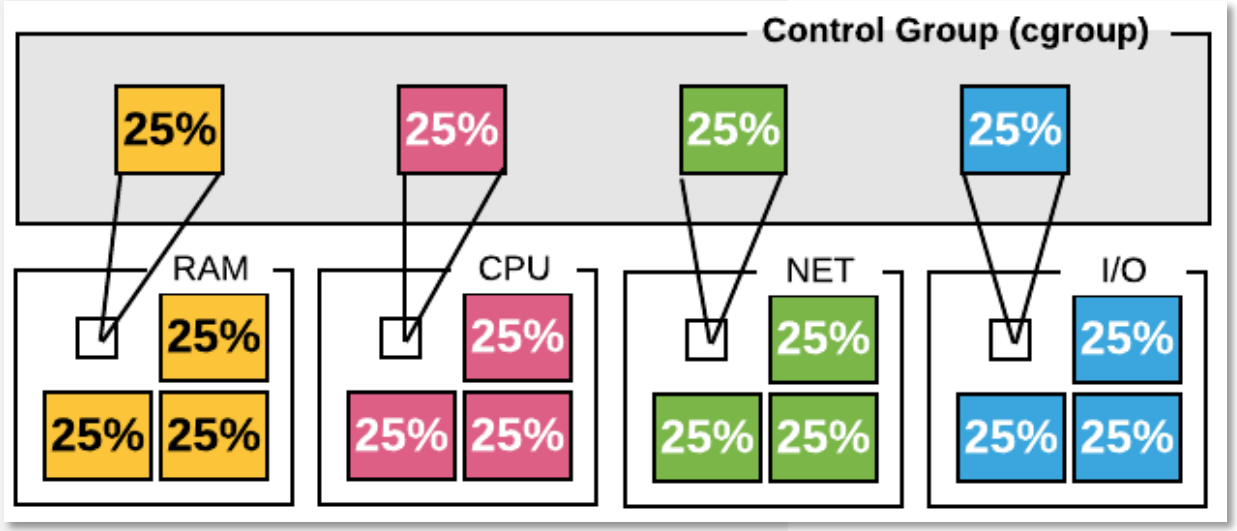
Union File System (inkl. Copy on Write)
Union-Dateisysteme gruppieren Verzeichnisse und Dateien des globalen Dateisystems in Layern. Tiefere Layer werden dabei mit höheren Layern vereinigt (daher Union, engl. Vereinigung).



OS-VIRTUALISIERUNG

Linux Control Groups (Isolation durch Grenzen)

Control Groups sind eine Methode des Linux Kernels (maßgeblich von Google entwickelt), um Prozesse in Gruppen einzuteilen und Ressourcen-Limits durchzusetzen. Prozessgruppen können geschachtelt (weitere Prozessgruppen enthalten) sein.



- **Ressourcenbeschränkung:** Prozessgruppen können so eingestellt werden, dass sie ein konfiguriertes Speicherlimit nicht überschreiten können.
- **Priorisierung:** Gruppen können einen festgelegten Anteil an der CPU-Auslastung oder dem I/O-Durchsatz zugewiesen werden.
- **Beobachtung:** Der Kernel misst wie viel Ressourcen bestimmte Gruppen verbrauchen. Dies kann beispielsweise für Abrechnungszwecke oder einfach nur Reporting oder Monitoring verwendet werden.
- **Kontrolle:** Der Kernel kann Prozessgruppen bei Überschreiten definierter Grenzen einfrieren (inkl. Checkpointing) und Neustarts von Prozessgruppen veranlassen.

OS-VIRTUALISIERUNG

Linux Kernel Namespaces (Isolation durch Sichtbarkeit)

Ein Namespace umschließt eine globale Systemressource mit einer Abstraktion, die den Prozessen im Namespace den Eindruck vermittelt, dass sie über eine eigene isolierte Instanz der globalen Ressource verfügen. Änderungen an der globalen Ressource sind nur für die Prozesse sichtbar, die Mitglieder des Namespace sind, für andere Prozesse jedoch nicht.

Eine Verwendung von Namespaces besteht darin, Container zu implementieren und diese voneinander zu isolieren.

Namespace	Isolierung
Cgroup	Cgroup root directory
IPC	System V IPC (Inter Process Communication), POSIX message queues
Network	Network devices, stacks, ports, etc.
Mount	Mount points
PID	Process IDs
Time	Boot and monotonic clocks
User	User and group IDs
UTS	Hostname and domain name

OS-VIRTUALISIERUNG

Linux Process Capabilities (Isolation mittels Rechten)

Bei der Durchführung von Berechtigungsprüfungen unterscheiden herkömmliche UNIX-Implementierungen zwei Kategorien von Prozessen: privilegierte Prozesse (Superuser oder Root) und nicht privilegierte Prozesse (alle anderen Benutzer). Privilegierte Prozesse umgehen alle Kernel-Berechtigungsprüfungen, während nicht privilegierte Prozesse einer vollständigen Berechtigungsprüfung auf der Grundlage der Anmeldeinformationen des Prozesses unterliegen.

Ab Kernel 2.2 unterteilt Linux die traditionell mit Superuser verbundenen Berechtigungen in verschiedene etwa 40 Einheiten, die als Capabilities bezeichnet werden und pro Thread unabhängig voneinander aktiviert und deaktiviert werden können.

Jeder Linux Prozess hat hierzu sogenannte Capability-Sätze, die es ermöglichen Prozessen „Root-Teilrechte" zuzuweisen (z.B. Netzwerkmanagement):

- **Effective** (E): Aktivitierte Capabilities definieren, welche Berechtigungen der Prozess tatsächlich hat.
- **Permitted** (P): Erlaubte, aber aktuell nicht genutzte Capabilities. Dies ermöglicht es Prozessen Capabilities an- und abzuschalten (bzw. zu entziehen).
- **Inheritable** (I): Welche Capabilities dürfen an Threads und Child-Prozesse weitergegeben werden.

OS-VIRTUALISIERUNG

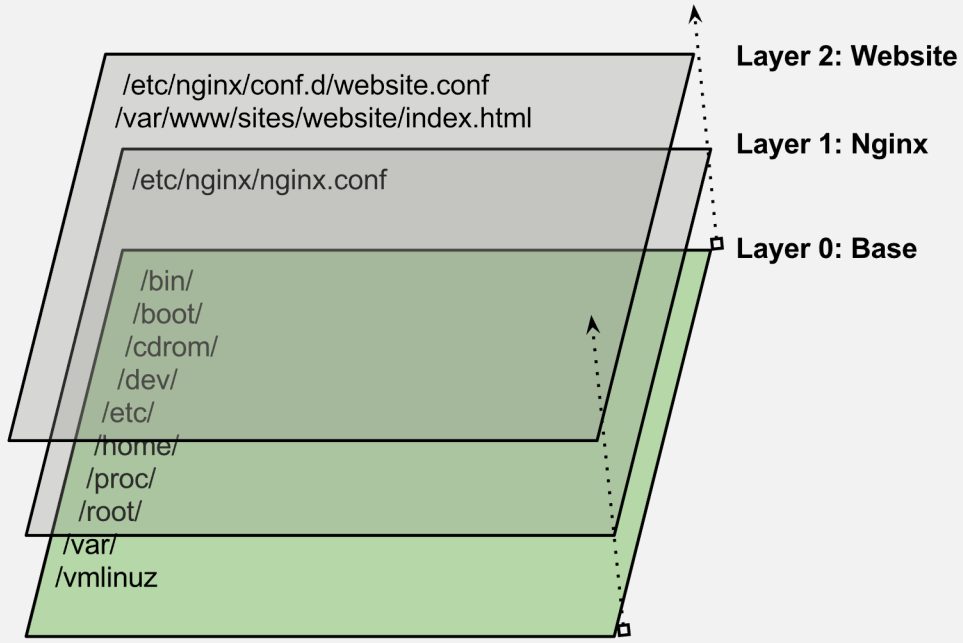
Union Filesystems (Isolation des geteilten Dateisystems)

Union Filesystems werden in Betriebssystemen dazu verwendet, Prozessen eigene Namensräume innerhalb von Dateisystemen zuzuweisen.

Dadurch können die Dateien verschiedener Dateisysteme zu einem einzigen logischen Dateisystem vereinigt werden.

Dateien, die zwar in getrennten Dateisystemen aber im gleichen Verzeichnis liegen, werden dadurch im selben Verzeichnis angezeigt.

Dabei werden den einzelnen beteiligten Layern Prioritäten zugeordnet, so dass eine eindeutige Zuordnung auch im Falle gleicher Dateinamen gewährleistet ist (höherer Layer überschreibt tieferen Layer).



Isolation durch Copy-on-Write:

Jeder Container blendet mit Start erforderliche Dateien der Basis als In-Memory-Kopien ein und isoliert so sein Dateisystem vom (persistenten) Host-Dateisystem. Werden diese Dateien durch den Container geändert, erfolgt dies nur In-Memory (nicht auf dem persistenten Storage). Auf diese Weise kann ein Container nicht das Dateisystems des Hosts ändern. Er ist „stateless" („vergisst" geschriebene Dateien nach einem Neustart) und isoliert sich so vom Dateisystem des Hosts.

AUSBLICK

CLOUD NATIVE COMPUTING

Hintergrund

- Platform as a Service
- Das PaaS-Problem
- Das CaaS-Versprechen

Betriebssystem-Virtualisierung

- OS-Virtualisierung
- Linux-basierte Techniken zur OS-Virtualisierung
- Standardisierung von Deployment-Einheiten => Container

Laufzeitumgebungen für Container

- Container Laufzeitumgebungen und Standards
- Docker
- Image Building und Registries

Container-Pattern

- Container (Anti-)Pattern
- 12-Factor Apps

KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de