

IT-Systemengineering & -Operations

Virtualisierung im DC (4)

OS Virtualisierung

Bruno Joho



Lernziele

- Der Studierende kennt die technischen Grundlagen wie OS Virtualisierung implementiert ist.
- Der Studierende kennt Vor-/Nachteile der OS Virtualisierung.
- Der Studierende kann sein erlerntes anwenden auf den Unterhalt und Betrieb eines Datacenters.

Inhalt



- Hypervisor vs. Container
- Operating System Virtualisierung
- Sketch
- Beispiele Linux OS Virtualisierung
- Isolation
- Docker

Server Virtualisierungsarten

- **HW Partitionierung**

nur noch auf high end Computer anzutreffen (IBM z und p Systems, Sun/Oracle Sparc). Ist teuer, erfüllt aber gewisse Sicherheitsnormen (zB: EAL5 für LPAR).



- **HW Virtualisierung (Hypervisor)**

Virtuelle Maschinen nach Poppek & Goldberg.

- **OS Virtualisierung**

Gegenstand dieser Vorlesung

Hypervisor vs. OS Virtualisierung

„Hypervisors are the living proof of operating system's incompetence"

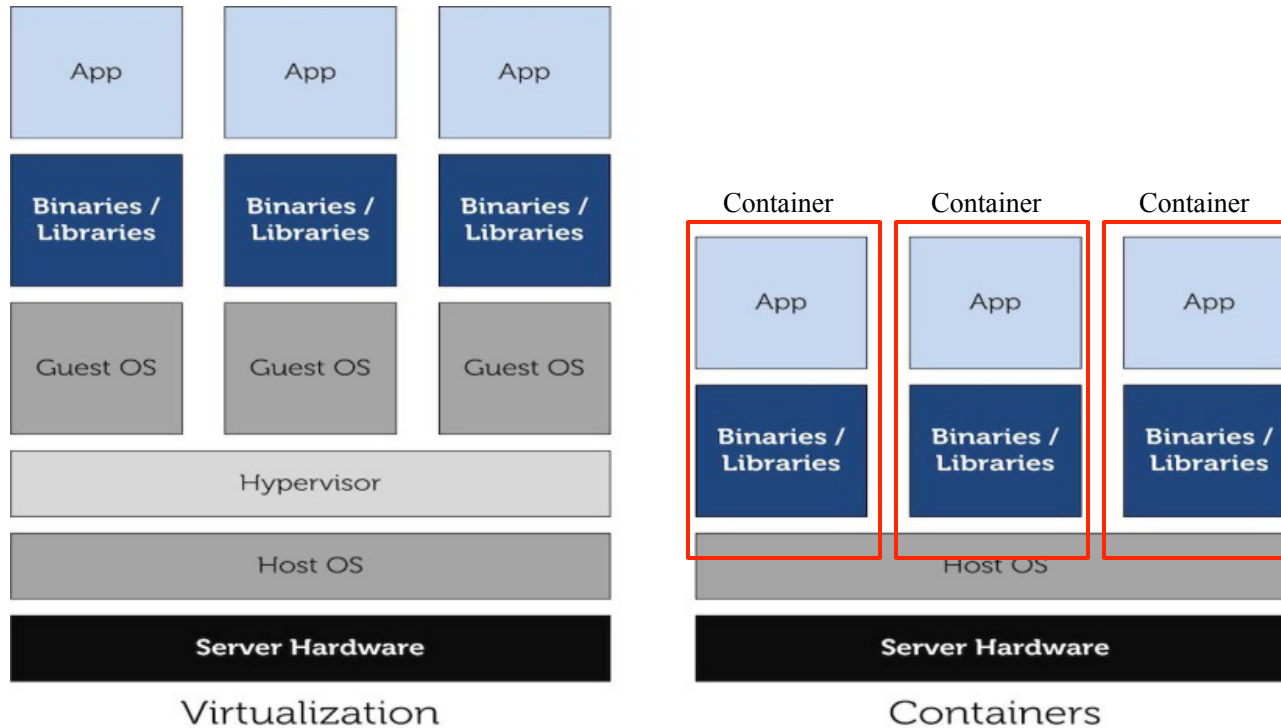
- Hypervisor wurden geschaffen um Mängel im OS auszugleichen.
- Die OS Virtualisierung ist die Antwort der OS Hersteller.

Container

Eine Instanz eines virtuellen OS wird Container (Linux) oder Zone (Solaris) genannt.

Eine virtuelle OS Instanz ohne eingeschaltetes Ressourcen Management ist nicht für die Produktion geeignet (vergleiche Übung OS Virtualisierung).

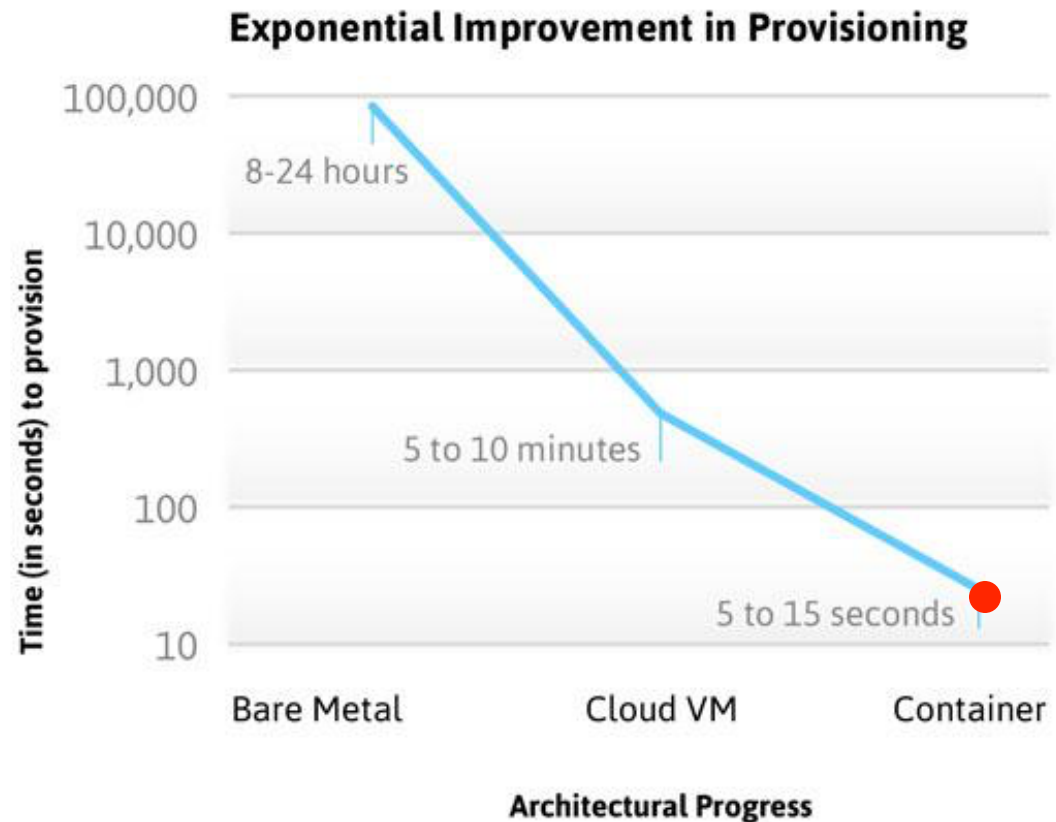
Hypervisor vs. Container




Container verwenden den Kernel vom Host OS

Provisionierung im Vergleich

VM's sind
schwergewichtig,
dagegen sind Container
super leichtgewichtig.



Inhalt

- Hypervisor vs. Container
- ▪ Operating System Virtualisierung
- Sketch
- Beispiele Linux OS Virtualisierung
- Isolation
- Docker

Operating System Virtualisierung

- verwendet keinen Hypervisor.
- also kein „Trap and Emulate“.
- braucht daher auch nur ganz wenige zusätzliche Prozessorzyklen (zB: mapping von UID zu vUID).
- verwendet nur einen Kern*.

* Solaris kennt zusätzlich die sog. Kernel-Zones was eine Ausnahme darstellt, weil so mehrere Kerne unabhängig voneinander verwendet werden können.

Was kann die OS Virtualisierung?

- Stellt eine feingranulare Kontrolle der individuellen Prozesse und Applikationen zur Verfügung.
- Erlaubt die Isolation von Applikationen.
- Transparente Migrationen von Applikationen* möglich. Im Gegensatz zum Hypervisor der nur ganze OS Instanzen migriert.

* Vergleichen Sie hierzu Docker

Was kann OS Virtualisierung nicht?

- Keine „fremden“ OS möglich. Nur gleiche OS können virtualisiert werden. Beispiel: kein Windows Programme möglich auf Linux Kern.
- Zonen müssen auf dem selben Patch Level laufen wie OS/Kern.
- Anforderungen an SysAdmin steigen (bei Hypervisor auch)
- Ressource Management muss eingeschaltet werden.

Welche OS beherrschen OS Virtualisierung?

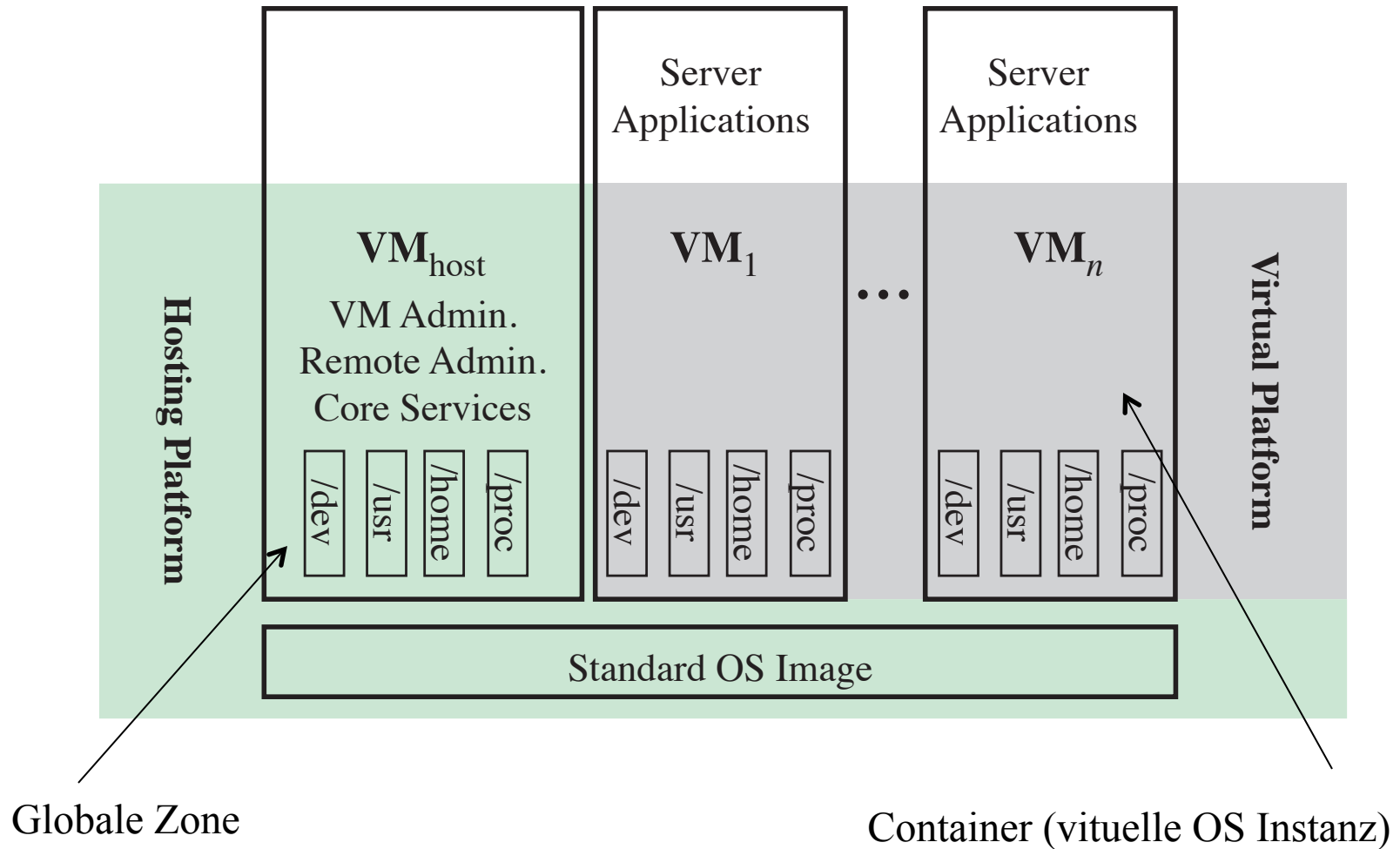
- BSD → Jails (hier ist die Idee entstanden...)
- Solaris → Zonen
- Linux → Container (LXC) – im mainstream Kern [3]
- Microsoft → Container* (Windows Server 2016)

Werden in Zukunft verschwinden:

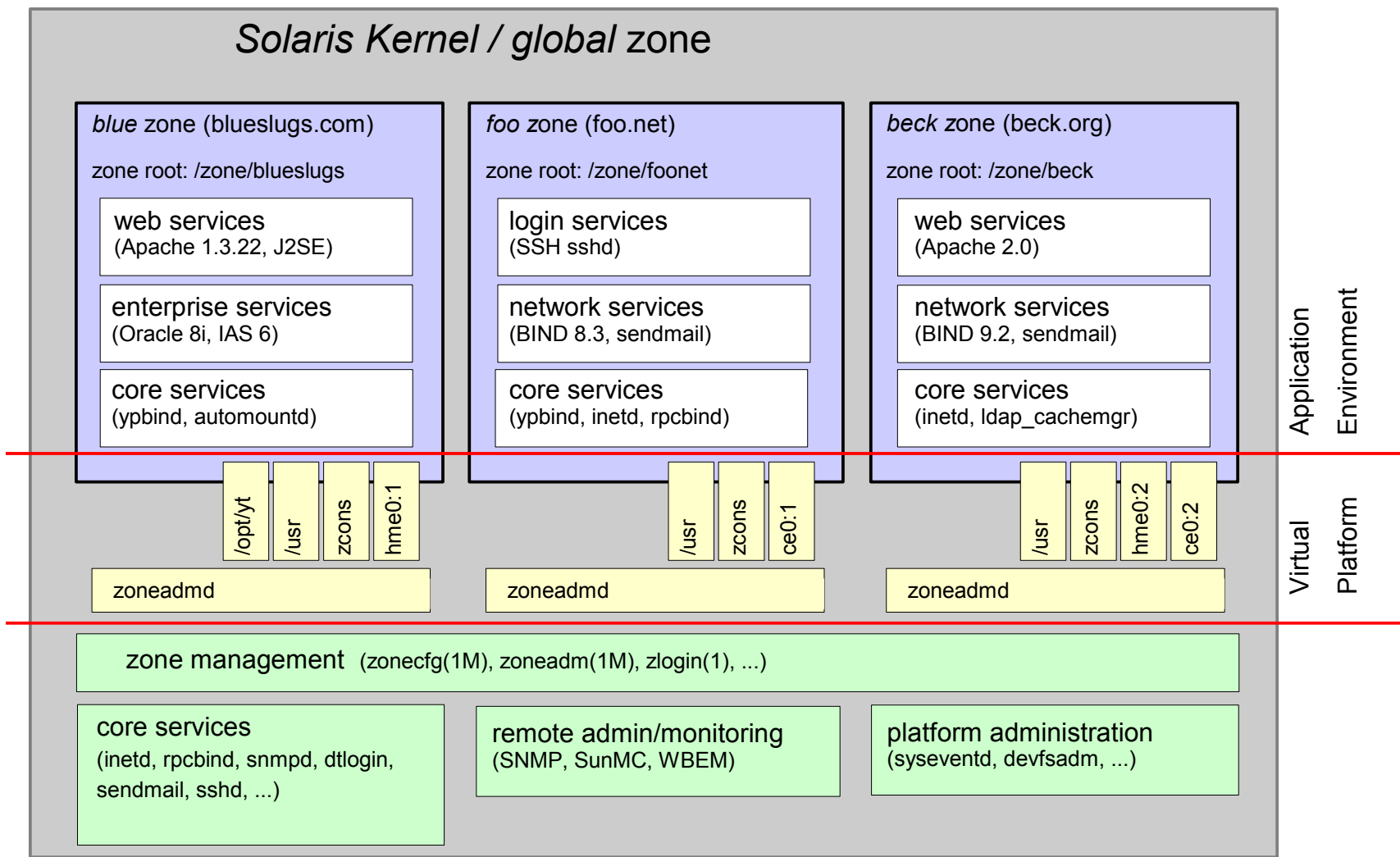
- Linux → OpenVZ - Kern Patch nötig.
- Linux → Vserver - Kern Patch nötig.

* Siehe auch [1], [2]


Linux Container Architektur



Solaris Zonen Architektur



Inhalt

- Hypervisor vs. Container
- Operating System Virtualisierung
- ▪ Sketch
- Beispiele Linux OS Virtualisierung
- Isolation
- Docker

Sketch


Auftrag: skizzieren Sie den Entwurf eines virtuellen Operating Systems.

Bedingungen: mehrere Projekte möchten z.B. eine Mail Server Instanz isoliert mit allen OS Administrationstools unabhängig voneinander laufen lassen können.

Es gibt keine Sicherheitsverletzungen und keine gegenseitige Beeinflussung der Prozesse. Der Benutzer sowie Prozesse haben nur die Sicht auf ihre virtuelle OS Instanz.

Wo würden Sie die wichtigsten Änderungen eines bestehenden Betriebssystems vornehmen?

Inhalt

- Hypervisor vs. Container
- Operating System Virtualisierung
- Sketch
-  ▪ Beispiele Linux OS Virtualisierung
- Isolation
- Docker

Beispiel: Linux Container

- Linux Container (LXC) ist Open Source, eine schnelle und leichtgewichtige Art der Implementierung eines virtuellen OS auf einem Linux Server.
- Nur eine Instanz vom Linux Kern ist involviert.
- LXC beinhaltet ein paar wenige einfache Modifikationen im Kern.
- Der Linux Kern unterstützt eine Anzahl separater virtuellen Linux Servern.
- Der Kern verwaltet alle System Ressourcen und Tasks, inklusive Prozess Scheduling, Memory, Disk Platz und Prozessor Zeiten.


Beispiel: Linux Container

Wir brauchen also mindestens 3 wichtige Erweiterungen:

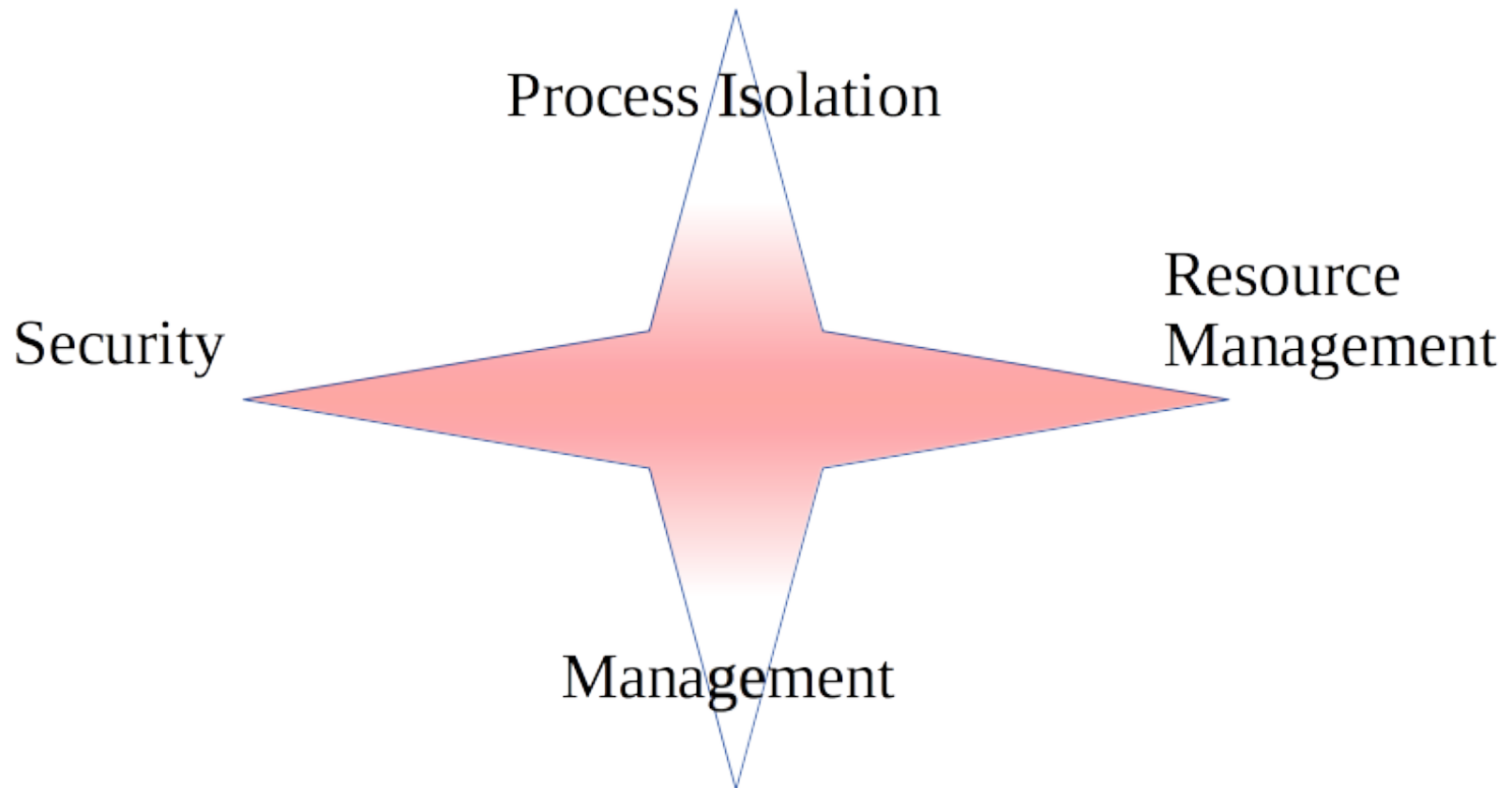
- Filesystemgrenzen für Prozesse → chroot
- Ressourcen sharing zwischen Prozess Gruppen → cgroups
- Namespaces → ns

Neben *chroot*, *namespaces* und *cgroups*, hat der Linux Kern alles was es braucht für die Unterstützung eines modernen Container Systems. Das ist genau das was LXC ausmacht – eine Ansammlung von Utensilien die mit *chroot*, *namespaces* und *cgroups* interagieren.

Inhalt

- Hypervisor vs. Container
- Operating System Virtualisierung
- Sketch
- Beispiele Linux OS Virtualisierung
-  ▪ Isolation
 - chroot
 - cgroups
 - namespace
- Docker

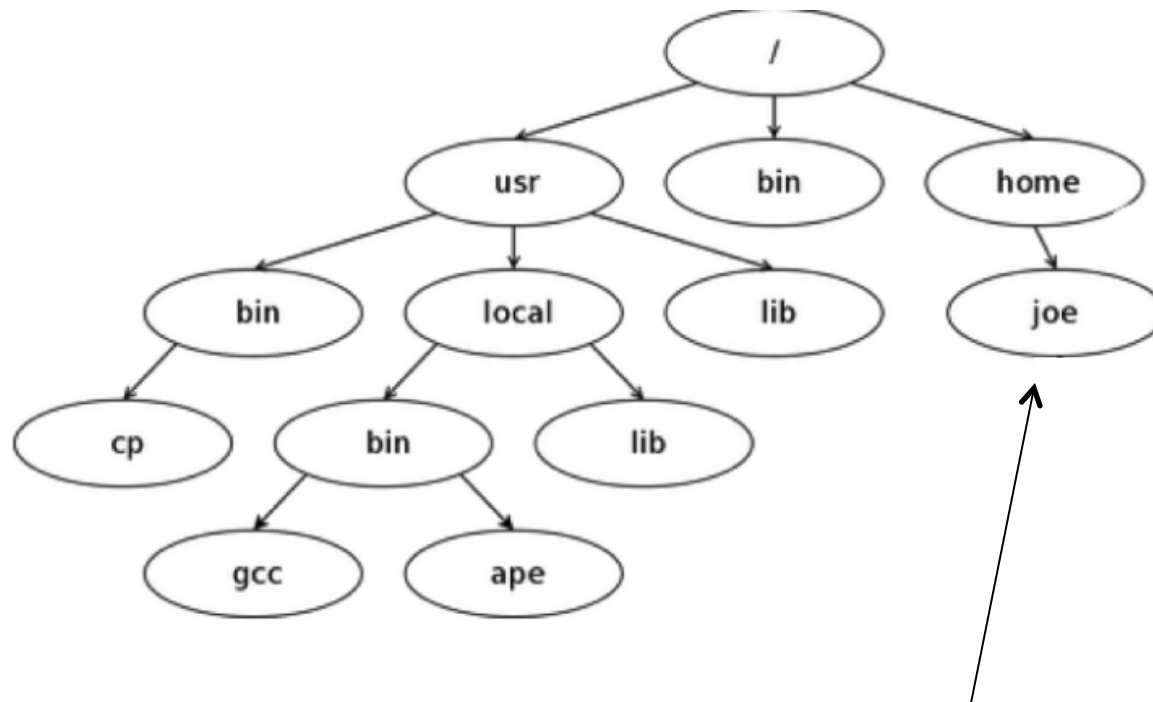
Key Elemente von einem Linux Container



Isolation im Linux File System mit chroot (1)

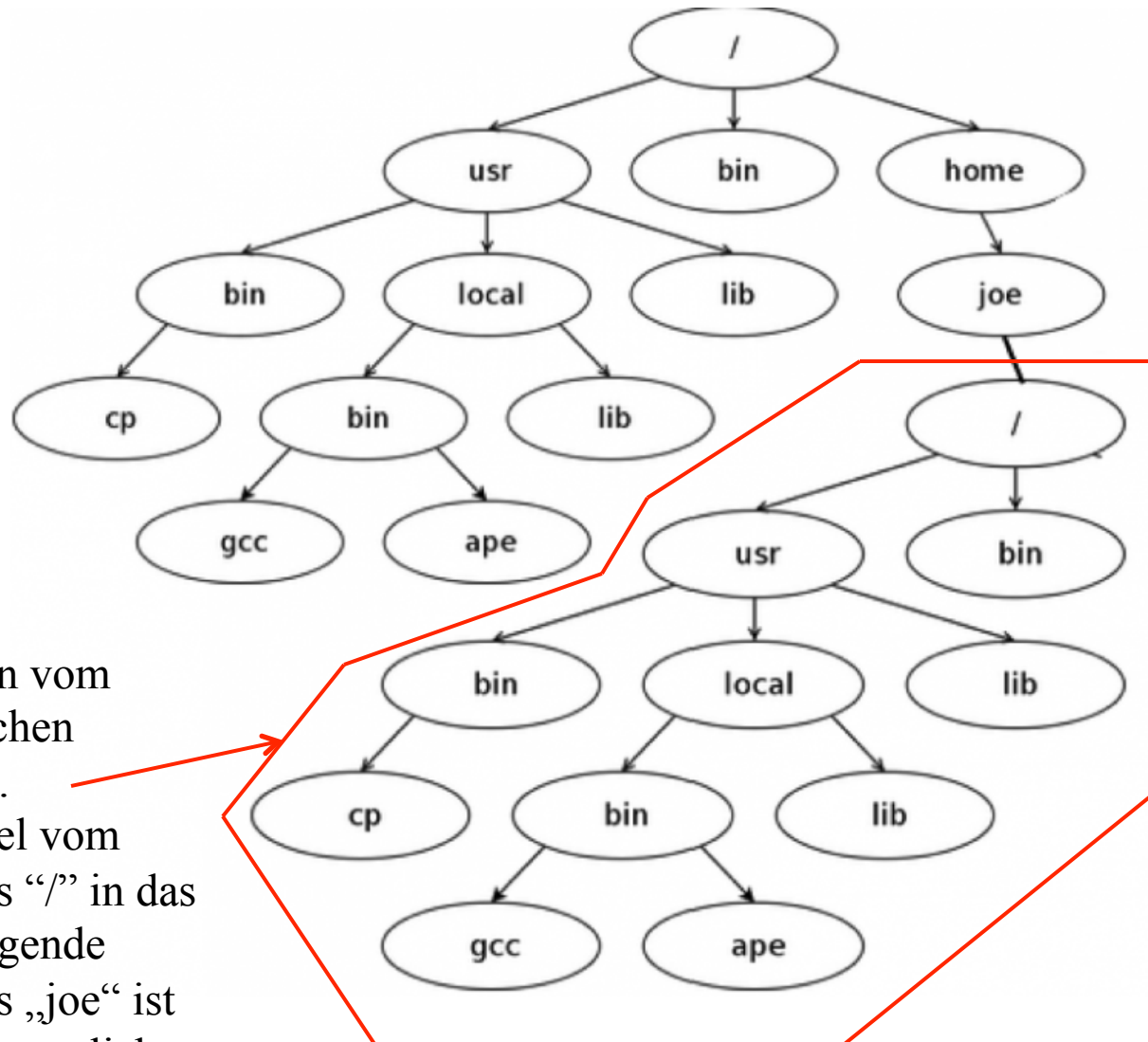
`chroot` ist eine Operation die das root Directory für den aktuell laufenden Prozess und dessen Kinder ändert. Ein Programm das in diesem modifizierten Umfeldes läuft kann auf keine Dateien und Kommandos mehr ausserhalb dieses Verzeichnis Baumes zugreifen. Das modifizierte Umfeld nennt man *chroot jail*.

Isolation im Linux File System mit chroot (1)



Eine neue FS Struktur mit dem Root Directory unterhalb “joe” soll angelegt werden.
Dazu verwenden wir das Dienstprogramm chroot

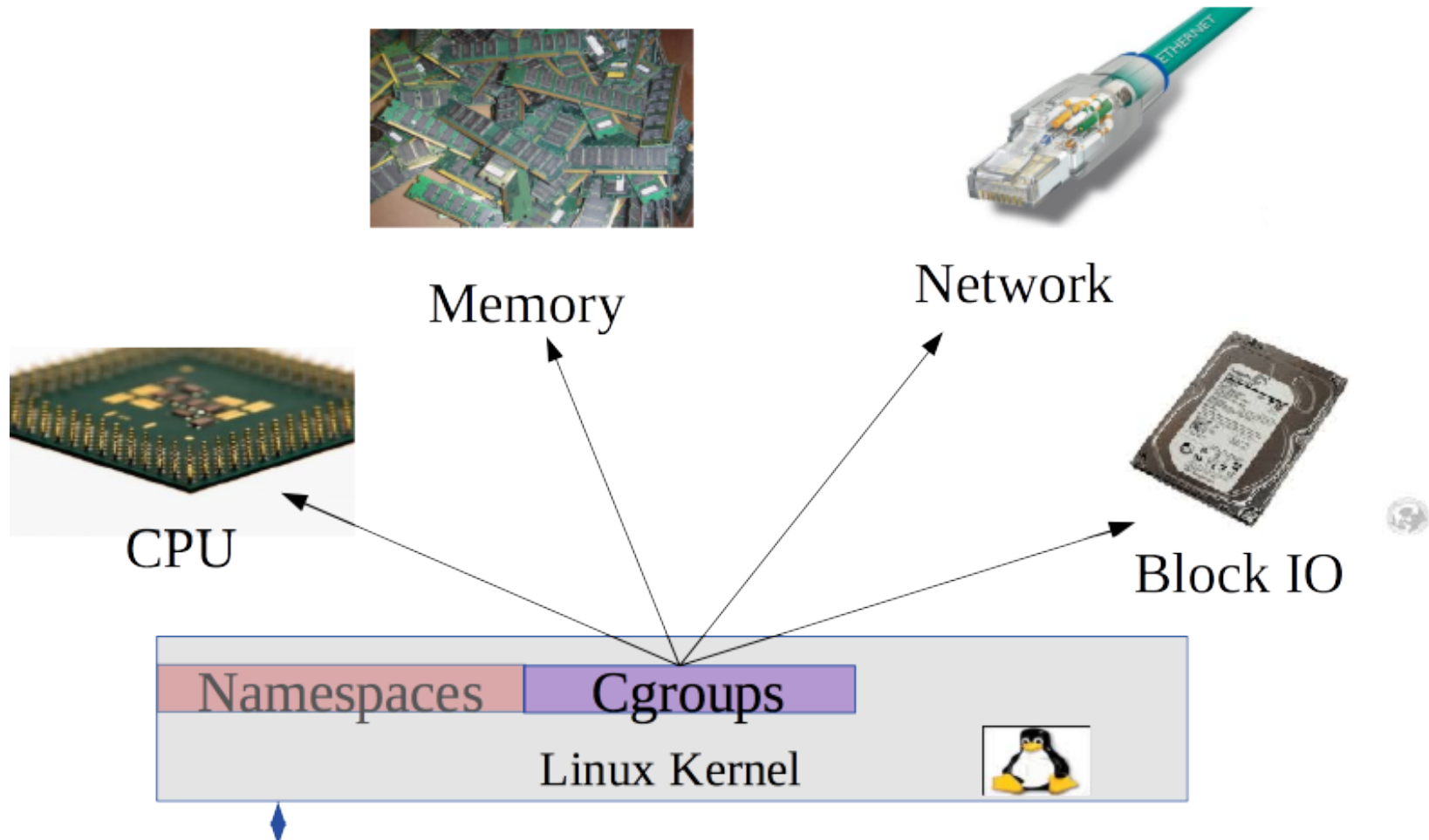
Isolation im Linux File System mit chroot (2)



Segregation vom ursprünglichen Filesystem.
Ein Wechsel vom Verzeichnis “/” in das darüber liegende Verzeichnis „joe“ ist nicht mehr möglich.

- cgroups sind ein Kernel Feature
- Erlauben das allozieren von Systemressourcen wie zB: CPU Zeit, System Memory, Netzwerkbandbreite oder eine Kombination davon über eine Benutzer definierte Gruppe von Tasks (Prozesse)
- cgroups erlauben dem Administrator eine feingranulare Kontrolle über die Zuordnung, Priorisierung, Verwaltung und Überwachung von Systemressourcen.
- HW Ressourcen können so aufgeteilt werden unter Benutzern und Tasks (Prozesse).
- cgroups sind wie Prozesse hierarchisch organisiert, wobei child cgroups die Attribute ihrer Eltern erben.
- cgroups sind wie Prozesse hierarchisch organisiert, wobei die Kinder die Attribute ihrer Eltern erben. Trotzdem gibt es zwischen den beiden Modellen Unterschiede (vergl. nächste Folie)

Ressourcen Management mit cgroups



Linux/Unix Prozess Modell

- Alle Prozesse im Linux System sind Kinder Prozesse von einem gemeinsamen Vater, dem `init` Prozess, welcher zur Bootzeit ausgeführt wird und andere Prozesse startet (welche ihrerseits wieder Kind-Prozesse erzeugen). Weil alle Prozesse von einem und demselben Vater abstammen ist das Linux/Unix Prozess Modell eine Mono-Hierarchie (oder Einzel Baum).
- Zusätzlich erbt ein Linux/Unix Prozess die Umgebung (zB. die PATH Variable) und bestimmte andere Attribute (zB File Deskriptoren) von seinem Vater Prozess.

cgroups Prozess Modell

- Der fundamentale Unterschied besteht darin, dass viele verschiedene Hierarchien von cgroups simultan in einem System existieren können. Wenn das Linux/Unix Prozess Modell ein „single Tree“ von Prozessen ist, so ist das cgroups Modell ein oder mehrere separate, unabhängige Prozess Bäume.
- Weil jede Hierarchie an ein oder mehrere Subsysteme gebunden sind braucht es mehrere separate Hierarchien. Ein Subsystem repräsentiert eine einzelne Ressource wie zB CPU Zeit oder Memory. Es gibt im Linux einige solcher Subsysteme (vergl. nächste Folie).

Linux Subsysteme

- **blkio** setzt Limiten auf Input/Output Zugriffe auf Blöcke Geräte **cpu** verwendet den Scheduler um cgroups Tasks die CPU Verwendung zu ermöglichen
- **cpuacct** generiert automatische Reports bez. CPU Verwendung. Diese werden von Tasks in einer cgroup verwendet.
- **cpuset** vergibt individuelle CPUs (in einem multicore System) und Memory an Tasks in einer cgroup.
- **devices** erlaubt/verhindert Zugriff von Tasks in einer cgroup auf Geräte.
- **freezer** suspends/resumes Tasks in einer cgroup.
- **memory** setzt Limiten auf Memory Verbrauch von Tasks in einer cgroup. Generiert automatische Reports dieser Memory Ressourcen
- **net_cls** setzt Tags mit einer ID (classid) auf Netzwerkpackete welche dem Linux traffic controller (tc) erlaubt zu Erkennen welche Packete von einer bestimmten cgroup Task stammen.
- **net_prio** setzt dynamisch Prioritäten auf Netzwerk Interfaces.
- **ns** — Das Namespace Subsystem (vergl. nächste Folie).

Isolation im Linux Namensraum durch Namespaces

- Ein Namespace abstrahiert eine globale System Ressource in der Art, dass der zum Namensraum gehörende Prozess meint er besitze diese Ressource einzigartig.
- Änderungen an dieser globale Ressource sind nur sichtbar für die Mitglieder im gleichen Namensraum. Für andere Prozesse sind sie nicht sichtbar.
- Eine offensichtliche Verwendung von Namensräumen ist die Implementation von Containern.

Bekannte Namespaces

Namespaces sind schon bekannt durch:

- Java namespace: `java.util.Date`
- C++: `std::array`
- XML: `xmlns:xhtml=http://www.w3.org/1999/xhtml`
`<xhtml:body>`
- C:\Program Files\System32\

Linux Namespaces

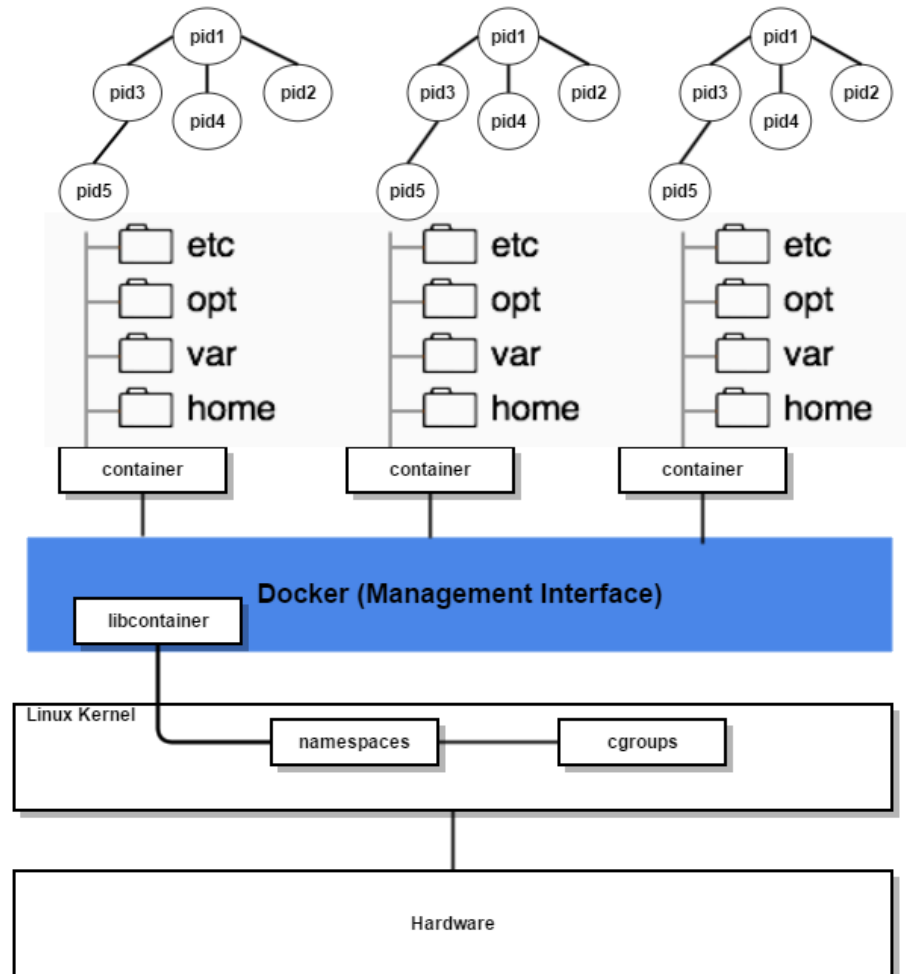
- **Mount namespaces** isolieren die FS mount Points wie sie von Prozessen gesehen werden. Prozesse in verschiedenen Namensräumen können so verschiedene Sichten auf die FS Hierarchie haben.
- **UTS namespaces** isolieren 2 System IDs. Node Name und Domainname. Das erlaubt jedem Container seinen eigenen Namen zu tragen.
- **IPC namespaces** isolieren gewisse Ressourcen für Inter Prozess Kommunikationen.
- **PID namespaces** isolieren den Prozess ID Nummernraum. So können Prozesse in verschiedenen Namensräumen die gleiche ID besitzen (zB jeder Container hat sein `init` PID 1).
- **Network namespaces** isolieren die System Ressourcen welche in Verbindung mit dem Netzwerk stehen. So kann jeder Namensraum sein eigenes Netzwerk Interface haben.
- **User namespaces** isolieren den User und Gruppen ID Nummernraum. So kann ein Prozess innerhalb des Namensraumes volle Root Rechte ausserhalb jedoch keine Privilegien haben.

Inhalt

- Hypervisor vs. Container
- Operating System Virtualisierung
- Sketch
- Beispiele Linux OS Virtualisierung
- Isolation
- Docker



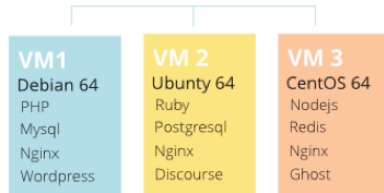
Docker und LXC Container



LXC vs. Docker

LXC

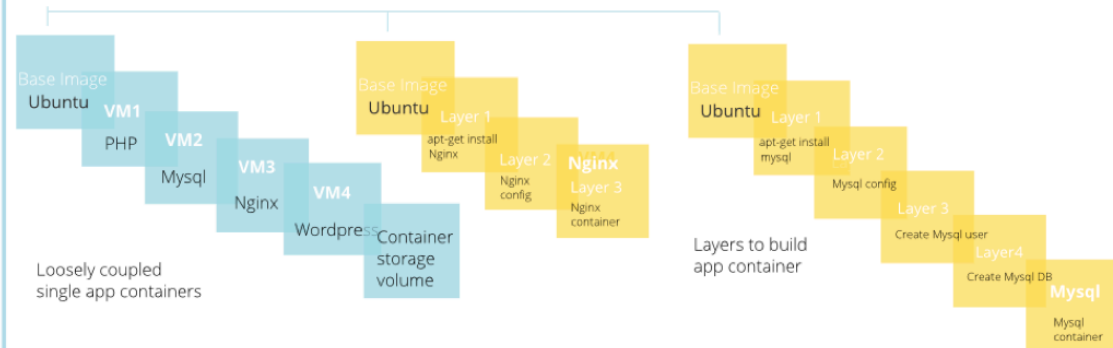
Host



- Filesystem neutral
- Containers are like VMs with a fully functional OS
- Data can be saved in a container or outside
- Build loosely coupled or composite stacks

Docker

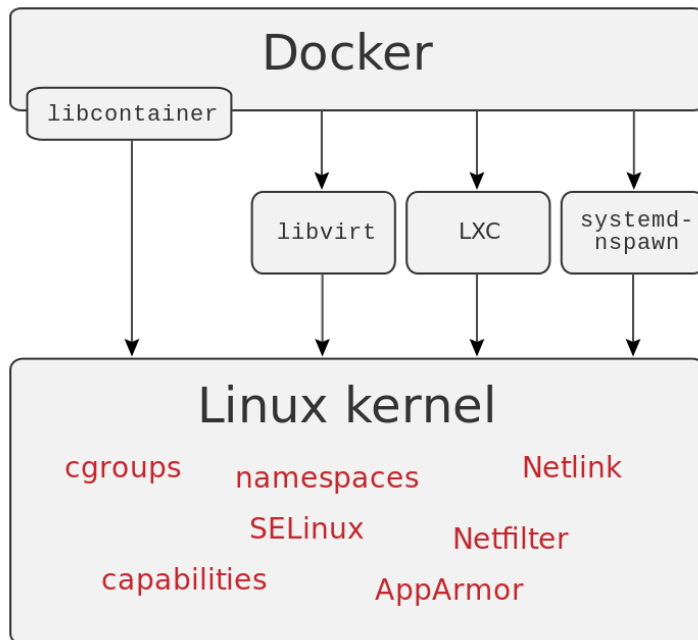
Host



- Containers are made up of read only layers via AUFS/Devicemapper
- Containers are designed to support a single application.
- Instances are ephemeral, persistent data is stored in bind mounts to host or data volume containers

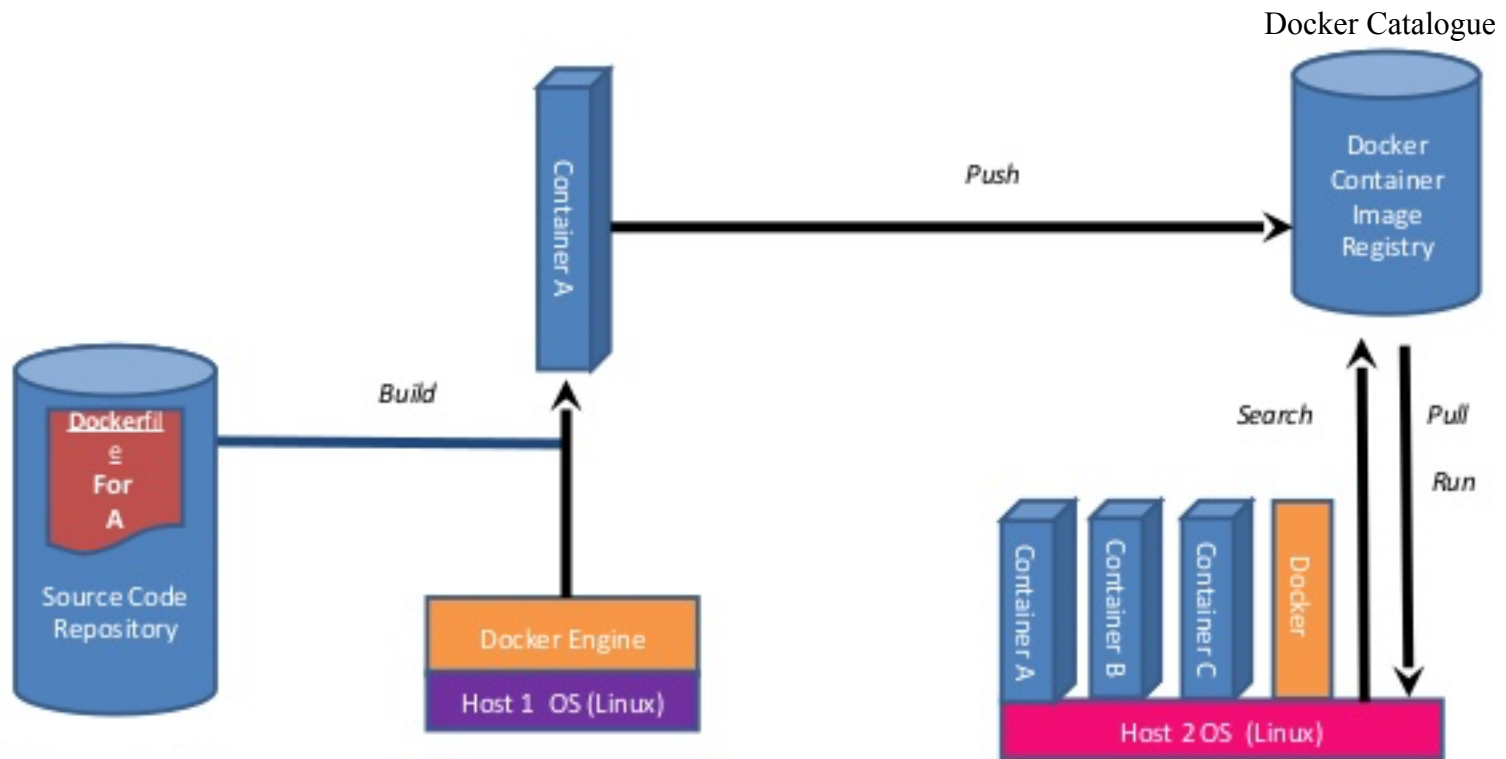
LXC ist eine Container Technologie während Docker eine Applikations-Engine basierend auf Containern ist.

Docker



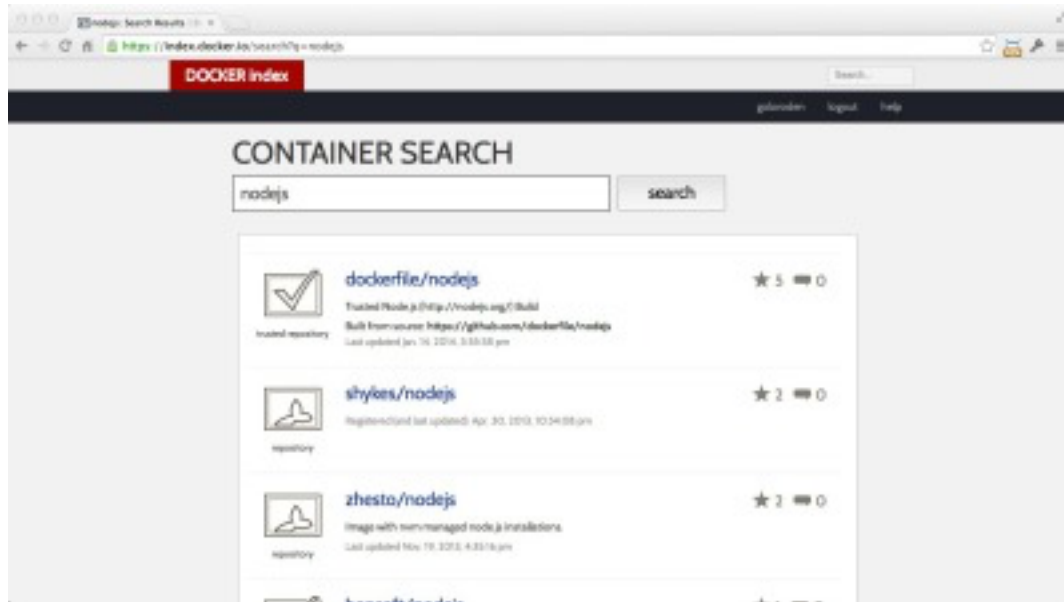
- Docker verwendet die Möglichkeiten von LXC um Applikationen innerhalb eines Container bereitzustellen.
- Solche „Container“ sind read only und hochgradig Portierbar.
- ist ein SW Layer über dem Linux Container.
- Dockt an jedes Linux an und verwendet die vorher besprochenen Kern Erweiterungen (erzeugt eigenen Namespace und cgroups für die Applikation durch die Virtualisierungs-APIs und die mitgebrachte `libcontainer` Library)

Docker Workflow



Stellt Katalog Funktionalität zur Verfügung [4]

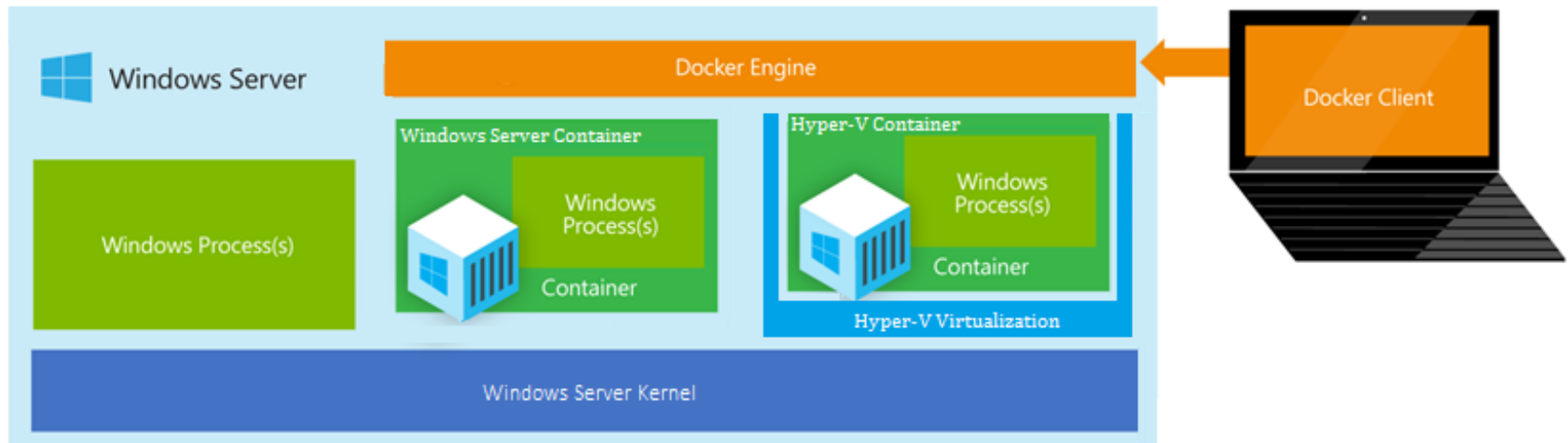
Docker Lebenszyklus



Beachten Sie die
kommende Lab Übung
„Docker“

- Herunterladen eines Images
`$ docker pull ubuntu`
- Suchen nach einem Image
`$ docker search nodejs`
- Mit run wird ein Container (und dessen Applikation) gestartet
`$ docker run ubuntu echo Hello world`
Hello world

Windows Container und Hyper-V Container



Applikationen die für Windows Server Container entwickelt wurden können nun auch ohne Modifikationen als Hyper-V Container deployed werden. Das erlaubt dem SysAdmin flexibel die Dichte, Mobilität und Isolation in einer Multi Plattform und Multi Applikationslandschaft zu wählen.

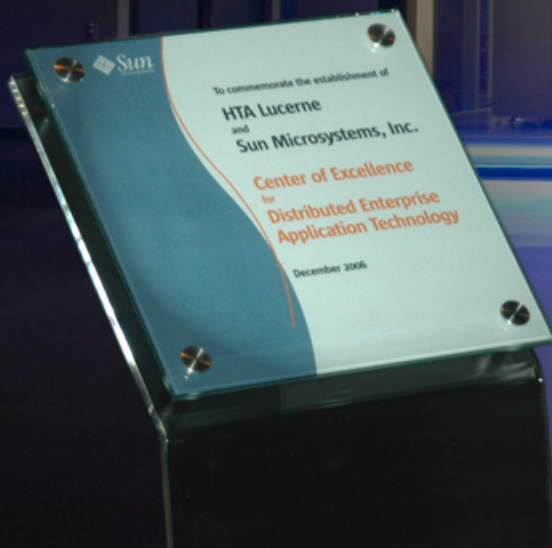
Ressources

- [1] OS-level Virtualization and Its Applications: A Dissertation Presented by Yang Yu
- [2] New Windows Server and Docker: <http://azure.microsoft.com/blog/2014/10/15/new-windows-server-containers-and-azure-support-for-docker/>
- [3] Linux Container: <https://linuxcontainers.org>
- [4] Docker Katalog: <https://registry.hub.docker.com/repos/library/>

Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Engineering & Architecture



Questions?