

CC: ENTWICKLUNG UND TRENDS



Dipl.-Medieninf. Hai Dang Le
Software Engineer
hhdang.88@gmail.com

2018



AGENDA

1. EINFÜHRUNG IN CLOUD COMPUTING

- a. Marktübersicht
- b. Begriffsdefinition & Eigenschaften
- c. Liefermodelle - Public, Private, Hybrid, Community Cloud
- d. Service-Modelle - IaaS, PaaS, SaaS

AGENDA

2. VERGANGENHEIT, GEGENWART UND ZUKUNFT

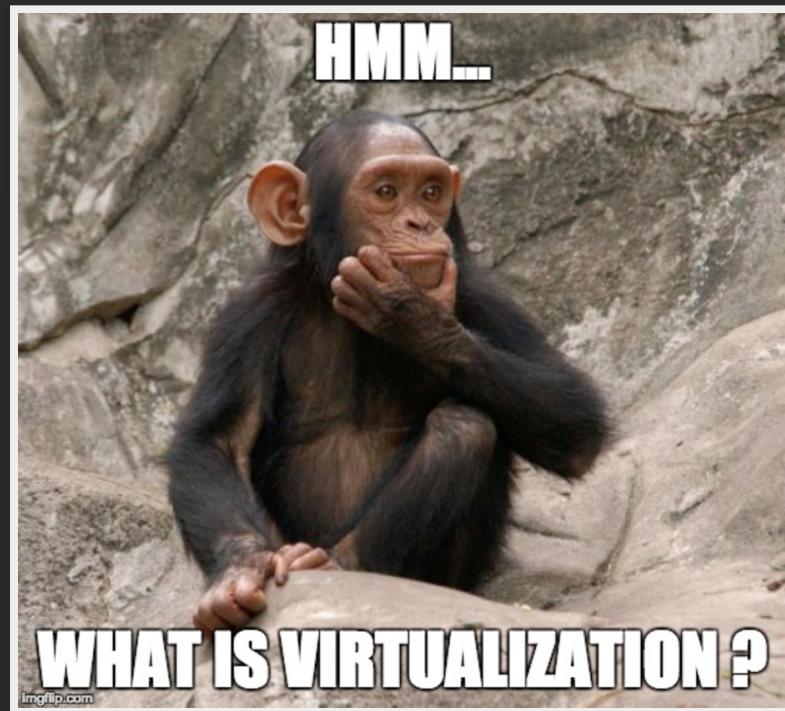
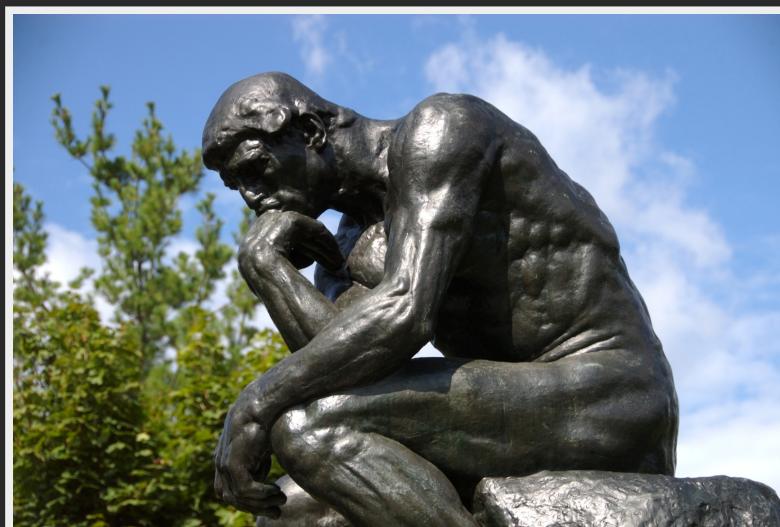
- a. Virtualisierung
- b. Technische Entwicklung
- c. Cloud Trends - Container, Container Orchestration,
Serverless
- d. Auto-Scaling
- e. Hands-On: Docker, Kubernetes, Cloud Foundry

2. VERGANGENHEIT, GEGENWART UND ZUKUNFT

A. CLOUD-ENABLER: VIRTUALISIERUNG

VIRTUALISIERUNG

Frage: Welche Bedeutung hat Virtualisierung für Cloud Computing ?



VIRTUALISIERUNG

essentiell u.a. für:

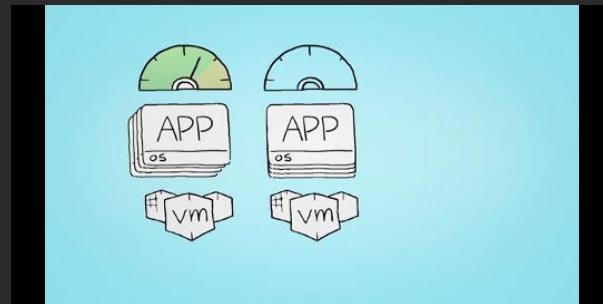
- Partitionierung von Ressourcen
- Skalierung / "Elastic Runtime"
- Sicherheit
- Multi-Tenancy
- Serverauslastung / Loadbalancing

VIRTUALISIERUNG: HANDSON DEMO

- Virtual Box
- Docker

60 min.

VIRTUALISIERUNG



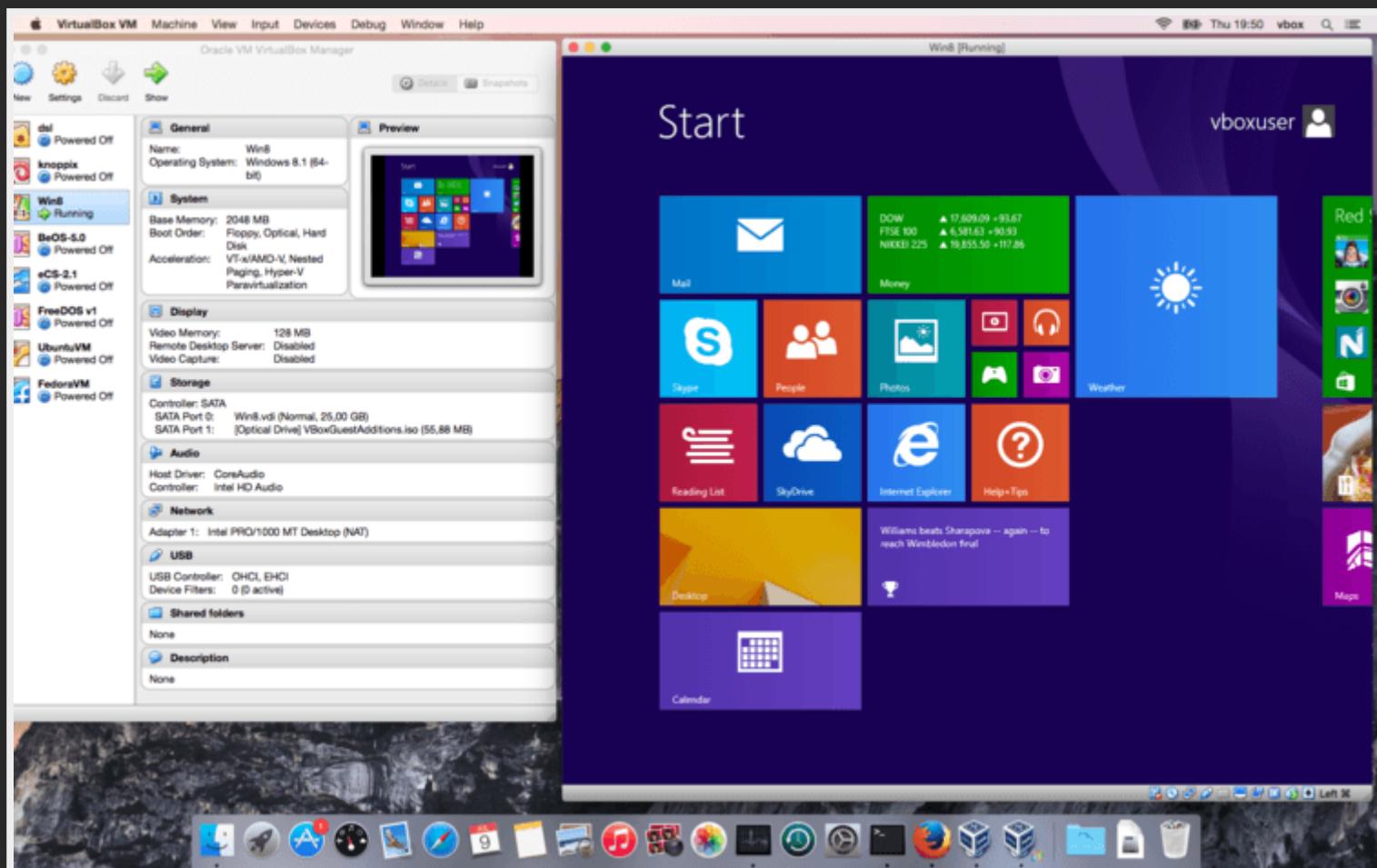
- VMWare

VIRTUALISIERUNG

"... refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources."

- Wikipedia, April.2017

BEISPIEL: SYSTEMVIRTUALISIERUNG



- <https://www.virtualbox.org/>

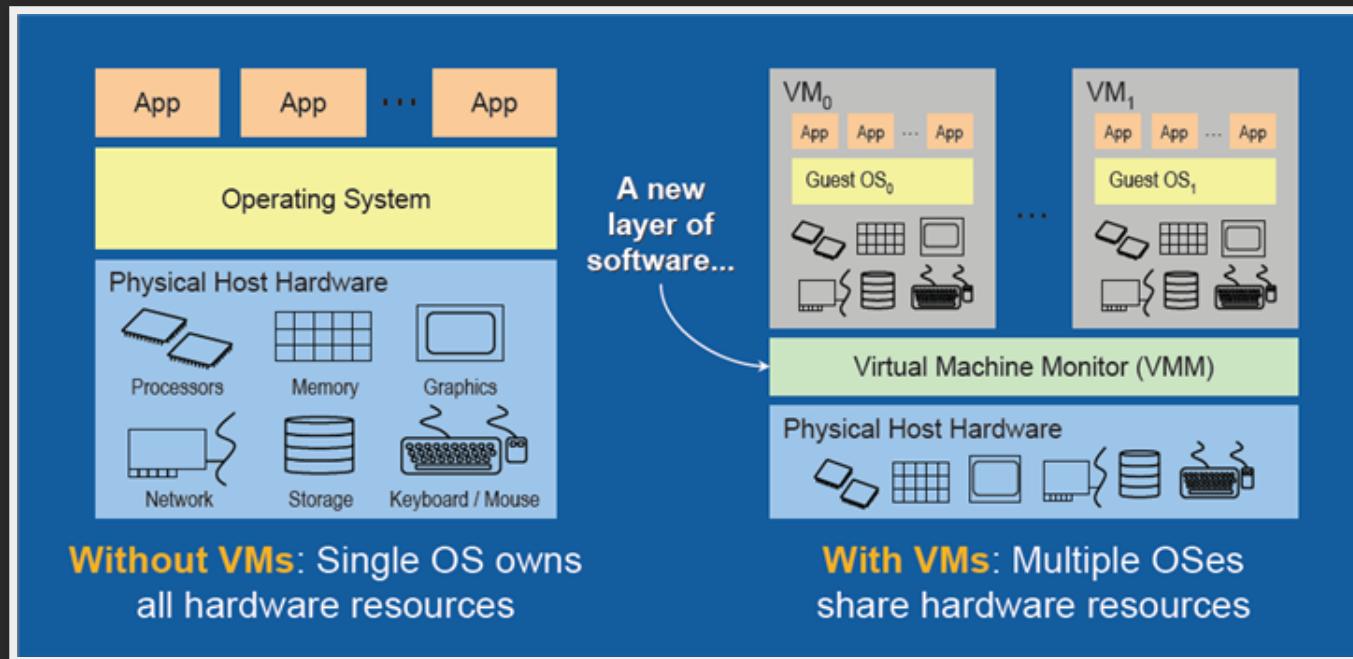
SYSTEMVIRTUALISIERUNG

- versteht man eine Virtualisierungsschicht zwischen Hardware und OS
- Host-System: Basissystem auf dem die Virtualisierungsschicht läuft (Host-OS, Bare-Metal)
- Gast-System: Gast-Betriebssystem welches virtualisiert wird
- VM: Container in der das Gast-Betriebssystem läuft
- Was wird virtualisiert: CPU, Speicher, Storage, I/O, Netzwerk

SYSTEMVIRTUALISIERUNG

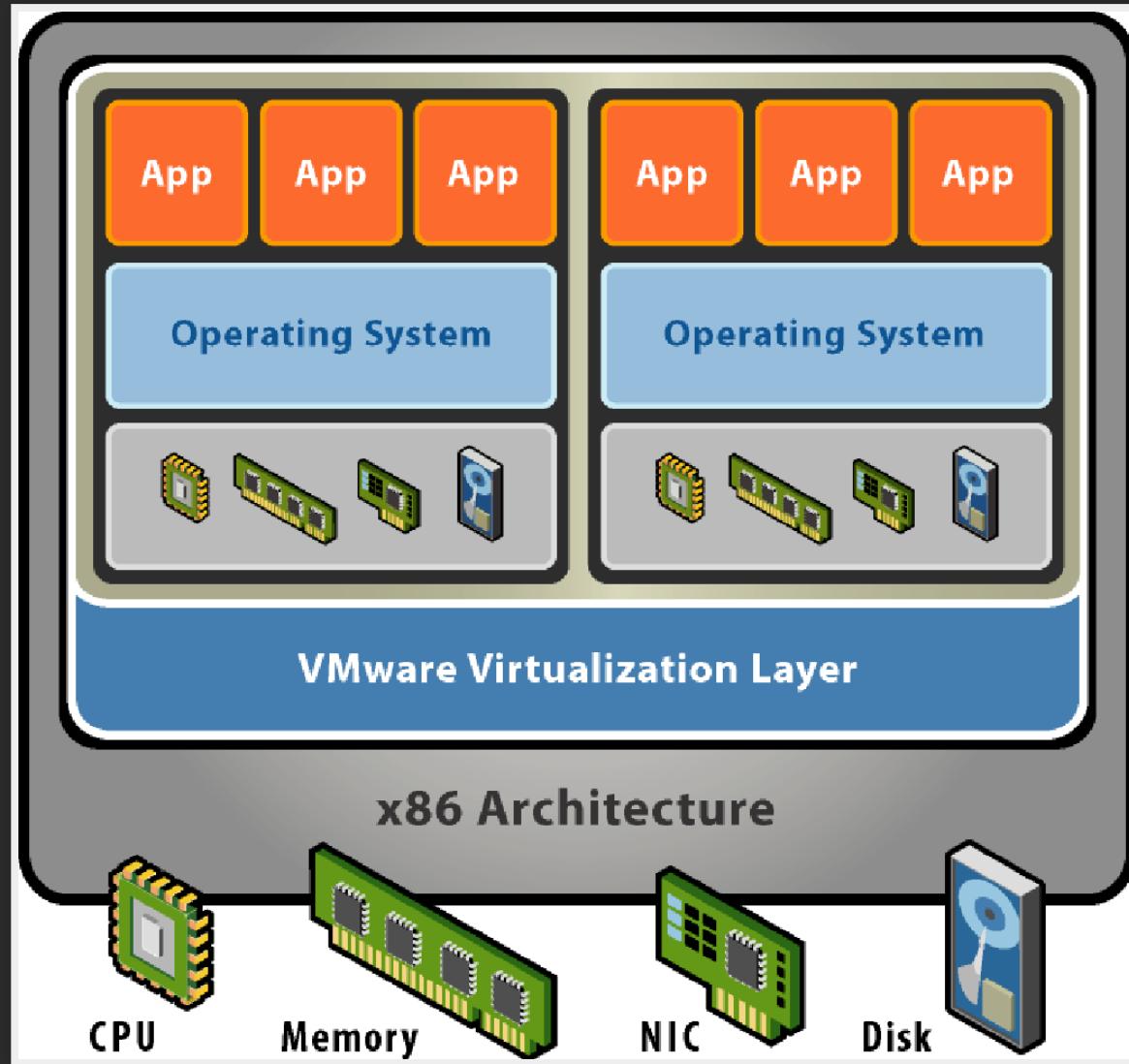
- physische Ressourcen werden partitioniert und abstrahiert bereitgestellt
- Hypervisor oder auch Virtual Machine Monitor (VMM) stellen Laufzeitumgebungen bereit
- Gast-Betriebssystem läuft in dieser Laufzeitumgebung mit virtualisierten Ressourcen (Speicher, Storage, Netzwerk, I/O)
- CPU-Operationen laufen direkt auf der unterliegenden Hardware (keine SW-Interpretation)
- I/O, Speicher- und Netzwerk-Operationen werden durch VMM umgeleitet

SYSTEMVIRTUALISIERUNG



<https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise>

SYSTEMVIRTUALISIERUNG



SYSTEMVIRTUALISIERUNG

- Kerneigenschaften:
 - Partitionierung der Ressourcen - die verfügbaren Ressourcen werden separiert den VMs bereitgestellt
 - Enkapsulierung - VM (-Zustand) wird durch ein Image-file abgebildet
 - Isolation - die VMs laufen unabhängig und voneinander getrennt (VMs können sich nicht beeinflussen)

SYSTEMVIRTUALISIERUNG: VERWENDUNG (BEISPIELE)

- Rechenzentrum
 - Steigerung der Auslastung durch mehrere VMs je phys. Server (z.B. 1 VM je App, Web- oder Application-Server oder Kunde)
 - Lastdistribution zwischen Servern: VMs werden durch Image-Files migriert
- Software Entwicklung
 - Testing: verschiedene Betriebssysteme, Konfigurationen, Software-Versionen werden durch Image-Files abgebildet
 - Distribution: Software Releases werden in ein Image-File verpackt und verteilt

EMULATION

DOSBox - Emulator



EMULATION

- Emulationssoftware, oder auch "Emulator", bildet ein System software-technisch ab
- ermöglicht Systeme zu simulieren welche nicht für die Host-Architektur ausgelegt sind
- im Gegensatz zur Virtualisierung läuft das zu emulierende System nicht direkt auf der CPU des Hosts ab
- Programm-Code welches im emulierten System läuft wird interpretiert

VIRTUALISIERUNG VS EMULATION

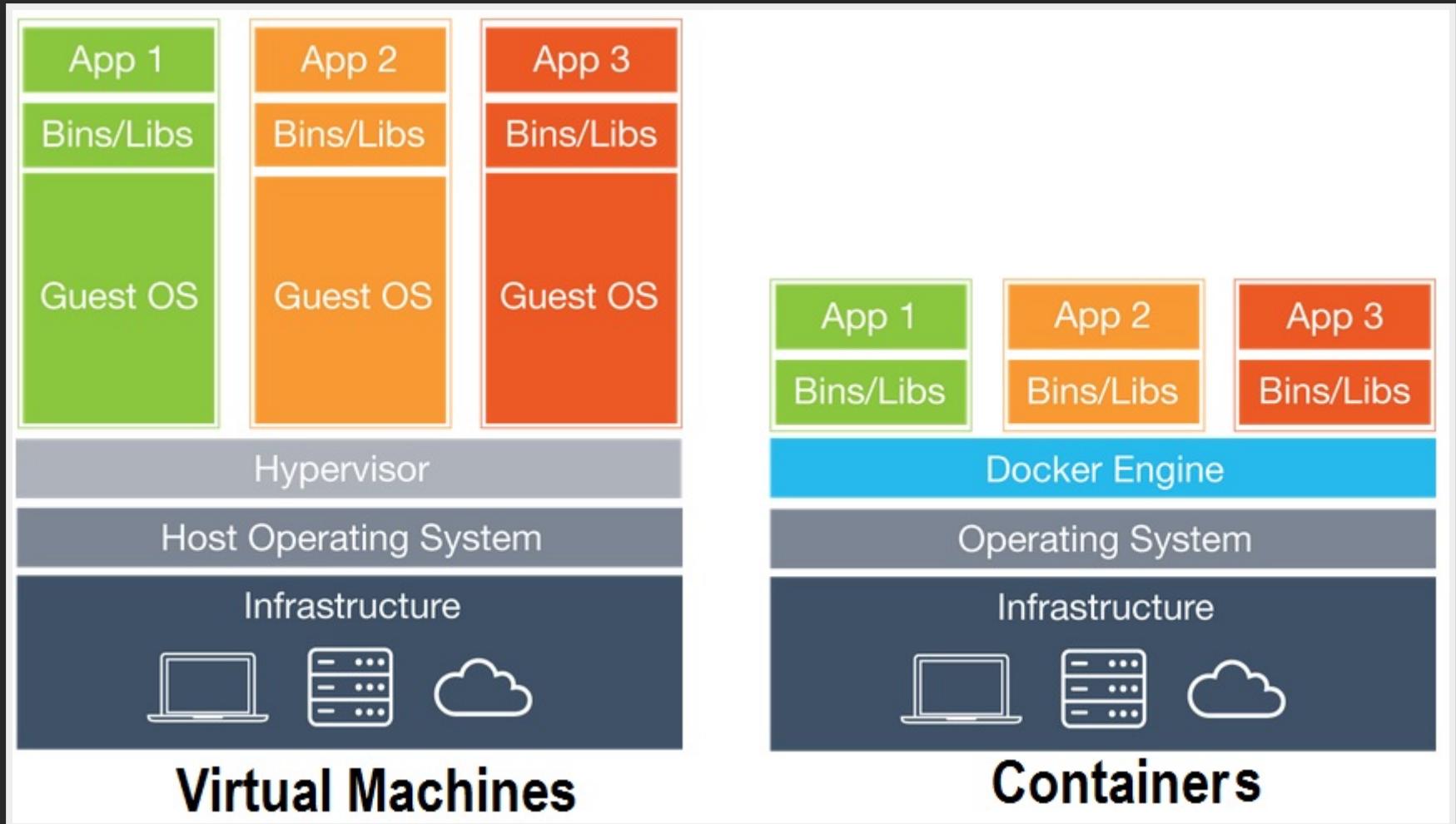
- Virtualisierung ist performanter, mit weniger Overhead
- Virtualisierung abstrahiert Laufzeitumgebungen (CPU, Speicher, I/O)
- Emulation simuliert ein System über die Emulationssoftware
- Emulation ist flexibler aber deutlich langsamer als Virtualisierung (Emulation-Overhead)

EMULATION: VERWENDUNG (BEISPIELE)

- Mobile Software Development (Android, iPhone - Emulatoren)
- Embedded Software Entwicklung (Steuergeräte in Autos, Haushaltsgeräte etc.)
- zum Abspielen alter Medien von nicht mehr unterstützter Hardware (SNES Emulator)
- Virtualisierung: zum simulieren von nicht vorhandener Hardware/inkompatibler Hardware

BETRIEBSSYSTEM-VIRTUALISIERUNG

BETRIEBSSYSTEM-VIRTUALISIERUNG VS SYSTEMVIRTUALISIERUNG



CONTAINER: VORTEILE

- effizient die Container laufen direkt auf dem Host-Betriebssystem bzw. teilen sich denselben Kernel
- das Gast-Betriebssystem muss nicht extra geladen werden
- Container-Images sind kleiner als VM-Images (lassen sich in Image-Repositories teilen)

CONTAINER: NACHTEILE

- Hosts können keine Fremdsysteme virtualisieren, wenn Sie nicht den gleichen Kernel haben
- Sicherheit: nicht so hoch bei VMs
 - Container teilen sich Kernel-Funktionen
 - keine vollständige Isolation: Fehler im Container können Einfluss auf geteilte Ressourcen haben
 - Ausbruch aus dem Container könnte direkt den Host kompromitieren

BEDEUTUNG VON VMS FÜR CLOUD COMPUTING

- bieten Sicherheit durch Isolierung
- Multi-Tenancy: Kunden teilen sich denselben Server, aber arbeiten getrennt voneinander (durch Isolation)
- ermöglicht dass VM-Instanzen zwischen Servern "wandern" (durch Enkapsulierung in VM-Images)
- flexiblere und bessere Auslastung von physischen Servern
- Grundbaustein für IaaS

BEDEUTUNG VON CONTAINERN FÜR CLOUD COMPUTING

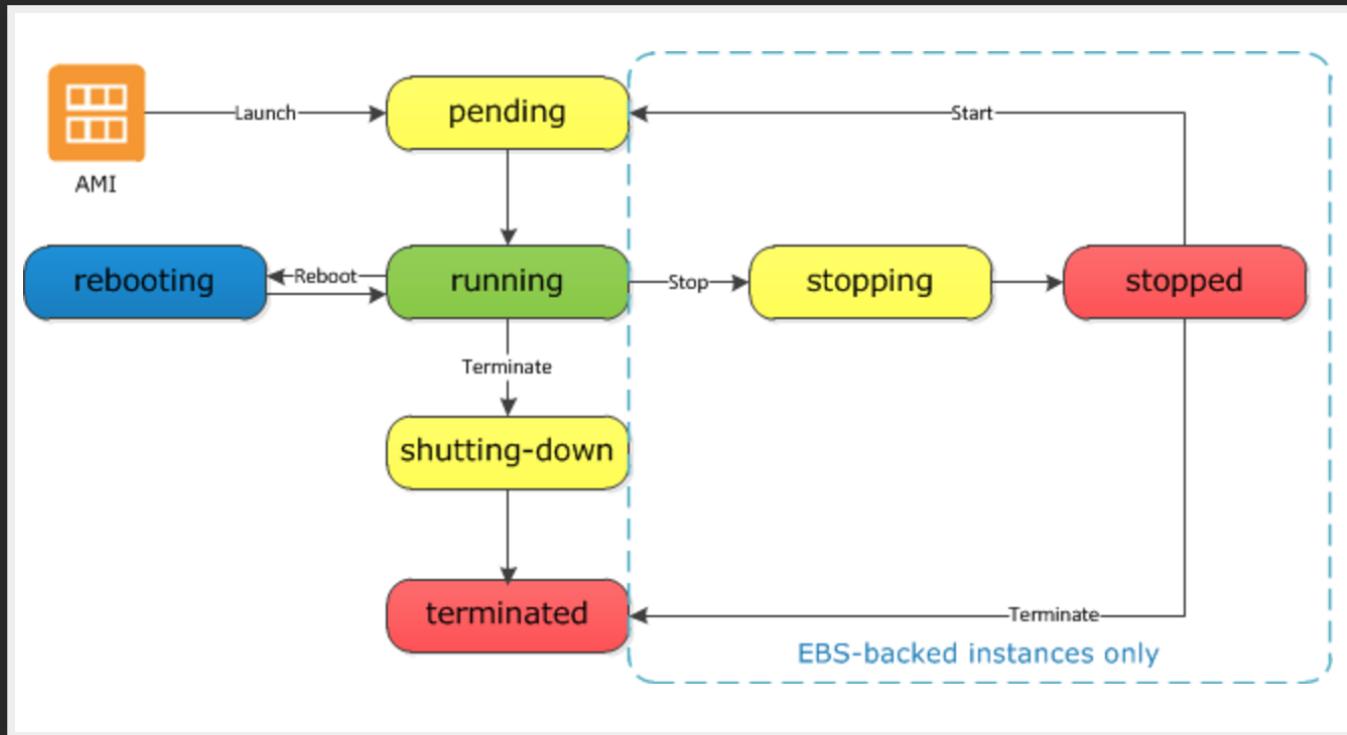
- effizienter als VMs
 - schnellere Skalierbarkeit durch schnelleres Boot-up
 - Container können effizienter zwischen VM-Instanzen "wandern" (leichtgewichtigere Container-Images)
 - eignen sich besser um Software Komponenten zu kapseln (Apps, DB, Software-LB, Web-Server etc.)
 - essentielle Grundlage für PaaS, CaaS und auch Serverless (OpenWhisk)

VM-EINSATZ BEI IAAS (AWS-ELASTIC COMPUTE CLOUD)

- Amazon Machine Image (AMI) bilden konfigurierten Stand einer VM-Instanz ab
- beliebig viele VM-Instanzen können durch ein AMI gestartet werden
- VM-Instanzen sind in EC2 volatil, d.h. der Lebenszyklus einer VM wird durch EC2 bestimmt
- Scale-Up: zusätzliche VM-Instanzen gestartet
- Scale-Down: VM-Instanzen werden terminiert

<https://aws.amazon.com/de/ec2/>

AWS EC2 - INSTANCE LIFECYCLE



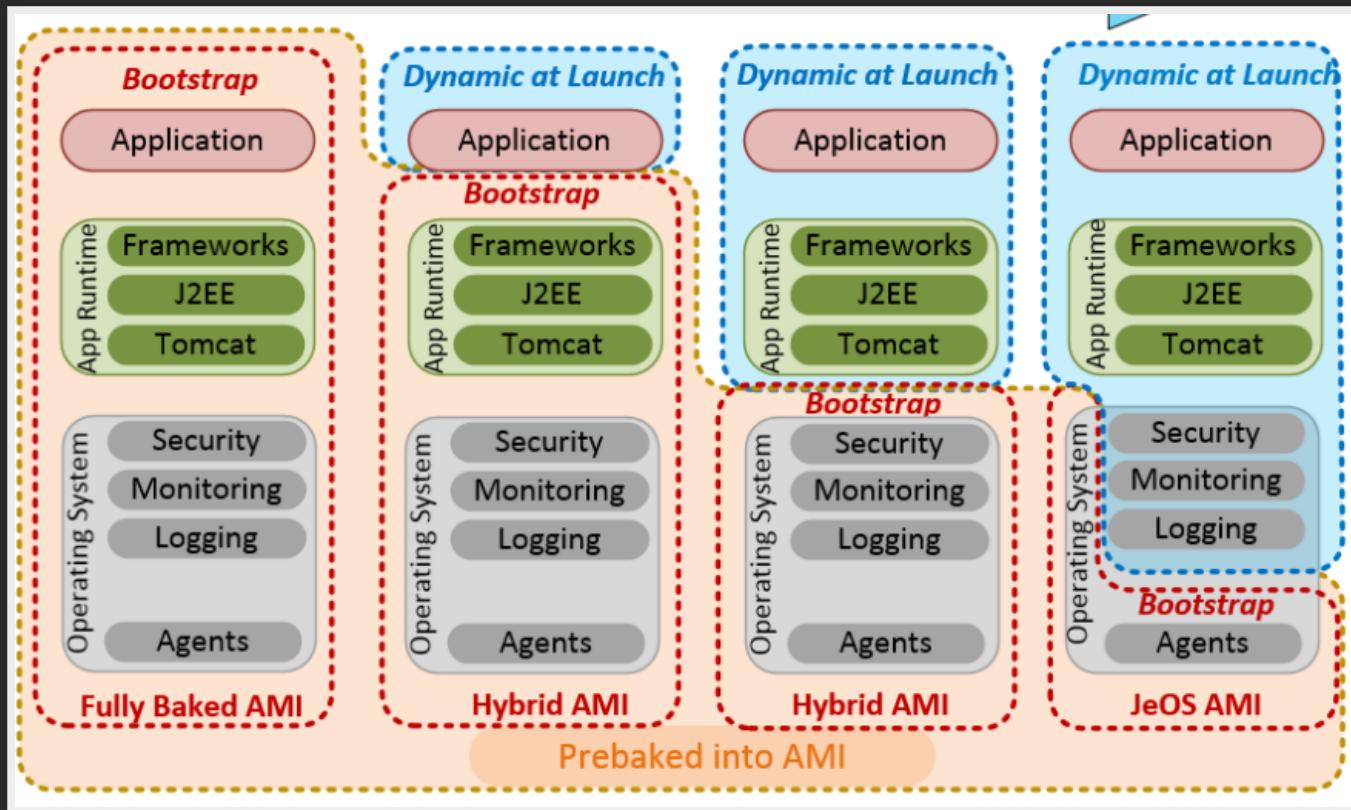
- <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html>

AWS EC2 - LIFECYCLE OPERATIONS

Characteristic	Reboot	Stop/start (Amazon EBS-backed instances only)	Terminate
Host computer	The instance stays on the same host computer	The instance runs on a new host computer	None
Private and public IPv4 addresses	These addresses stay the same	EC2-Classic: The instance gets new private and public IPv4 addresses EC2-VPC: The instance keeps its private IPv4 address. The instance gets a new public IPv4 address, unless it has an Elastic IP address (EIP), which doesn't change during a stop/start.	None
Elastic IP addresses (IPv4)	The Elastic IP remains associated with the instance	EC2-Classic: The Elastic IP is disassociated from the instance EC2-VPC: The Elastic IP remains associated with the instance	The Elastic IP is disassociated from the instance
IPv6 address (EC2-VPC only)	The address stays the same	The instance keeps its IPv6 address	None
Instance store volumes	The data is preserved	The data is erased	The data is erased
Root device volume	The volume is preserved	The volume is preserved	The volume is deleted by default
Billing	The instance billing hour doesn't change.	You stop incurring charges for an instance as soon as its state changes to stopping . Each time an instance transitions from stopped to running , we start a new instance billing hour.	You stop incurring charges for an instance as soon as its state changes to shutting-down .

- <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html>

AWS EC2 - AMI



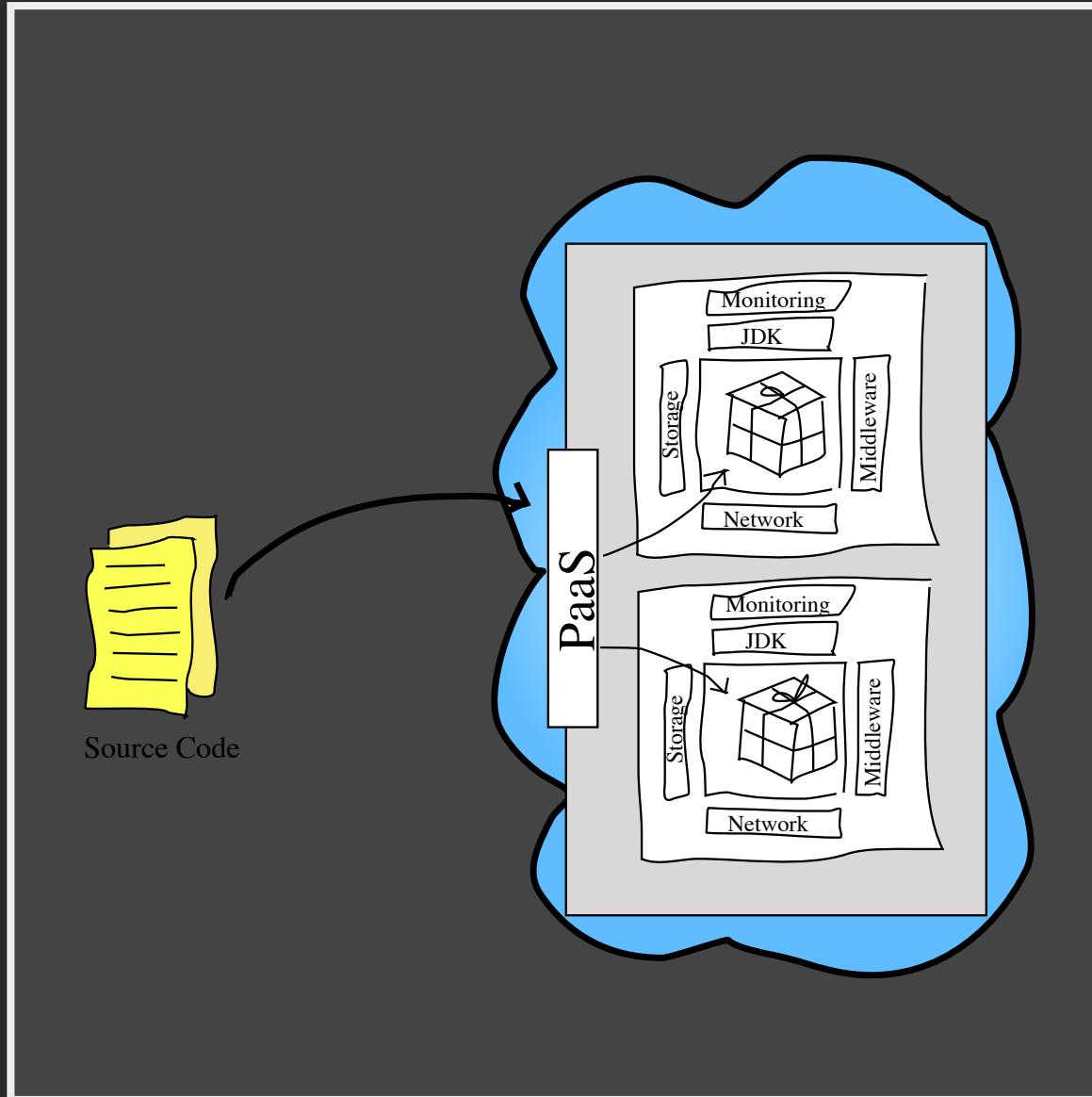
CONTAINER-EINSATZ BEI PAAS (CLOUD FOUNDRY)

- Container-Images sind "Templates"
- es können beliebig viele Container-Instanzen aus Images erzeugt werden
- Grundbaustein für PaaS
- Lebenszyklus von Container wird von der PaaS-Umgebung bestimmt (d.h. Container-Zustand sondern **volatil**)
- schnellere Skalierung durch leichtgewichtigere Container

CLOUD FOUNDRY

- "Bring-Your-Own-Code"-Prinzip: Container-Image wird durch ein "Buildpack"-script gebaut
- Container-Updates werden durch die Plattform automatisch durchgeführt, bspw. Security-Patching, Runtime-Updates
- Plattform startet/pausiert/rebootet/terminiert Container eigenständig
- Container = "Cell"

"BRING-YOUR-OWN-CODE"

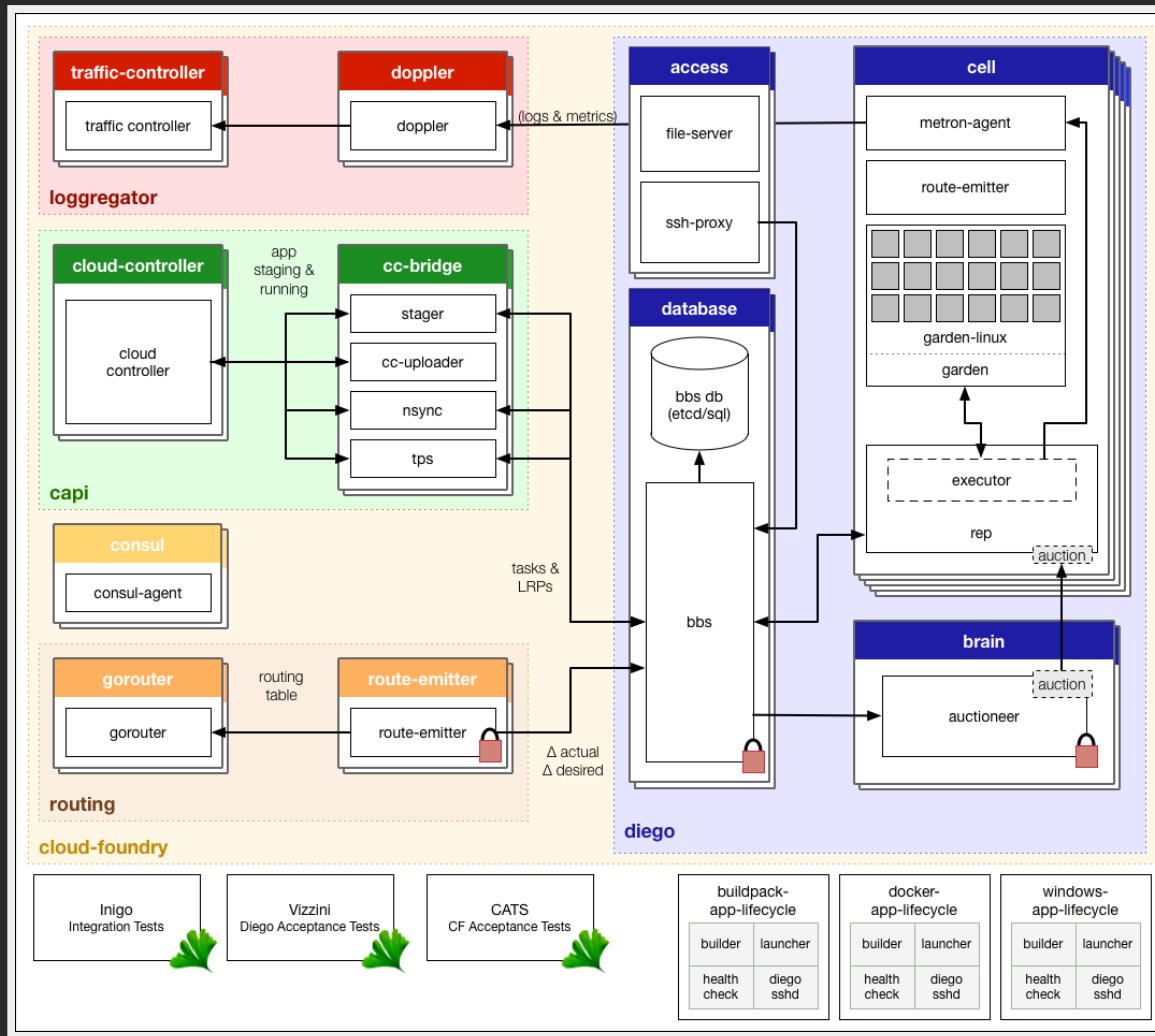


CF CONTAINER VS DOCKER CONTAINER

Feature	Warden	Docker
1. Resource isolation and control	<ul style="list-style-type: none">• CPU shares• memory + swap• network bandwidth• disk size quota	<ul style="list-style-type: none">• CPU shares• CPU sets• memory• memory swap• block device bandwidth
2. Dynamic resource management	Warden containers support this feature, but Cloud Foundry doesn't use it.	Not supported.
3. Image management	Only whole images can be reused to create new containers.	Layered—allows for reusing separate layers.
4. Linking containers	No	Yes
5. Exposing ports	Single port per container (Multiple ports will be available in Garden/Diego.)	Multiple ports per container

<https://www.cloudfoundry.org/cloud-foundry-containers-difference-warden-docker-garden/>

CLOUD FOUNDRY ARCHITECTURE



<https://docs.cloudfoundry.org/concepts/diego/diego-architecture.html>

2. VERGANGENHEIT, GEGENWART UND ZUKUNFT

B. CLOUD HISTORIE

DIGITALISIERUNG ...

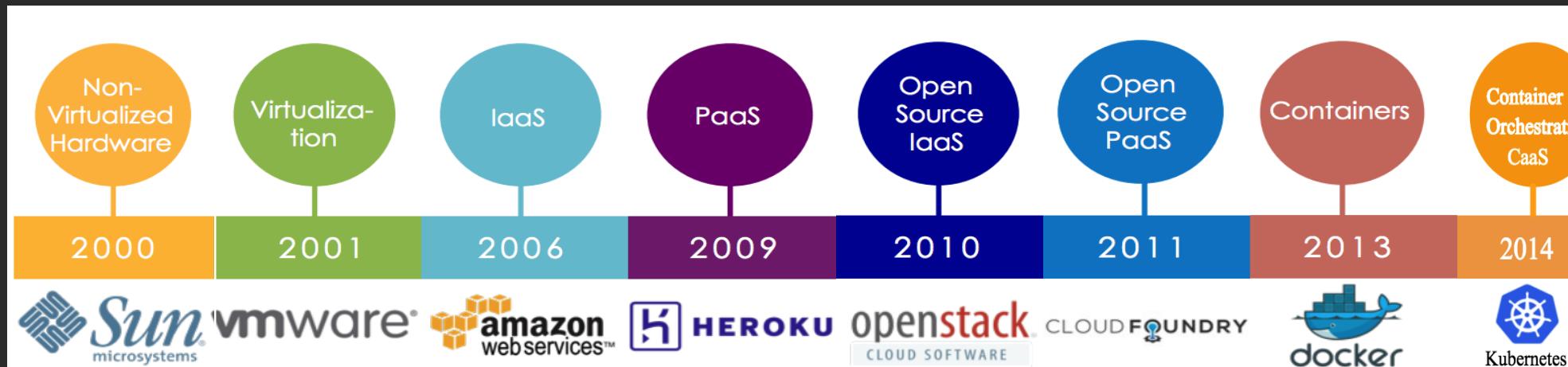
- Traditionelle Unternehmen verkaufen physische Produkte und Dienstleistungen
- Arbeitsprozesse basieren auf analogen Medien, keine Vernetzung der Informationsquellen
- Digitale Vertriebskanäle nicht vorhanden, keine schnelle globale Verbreitung möglich

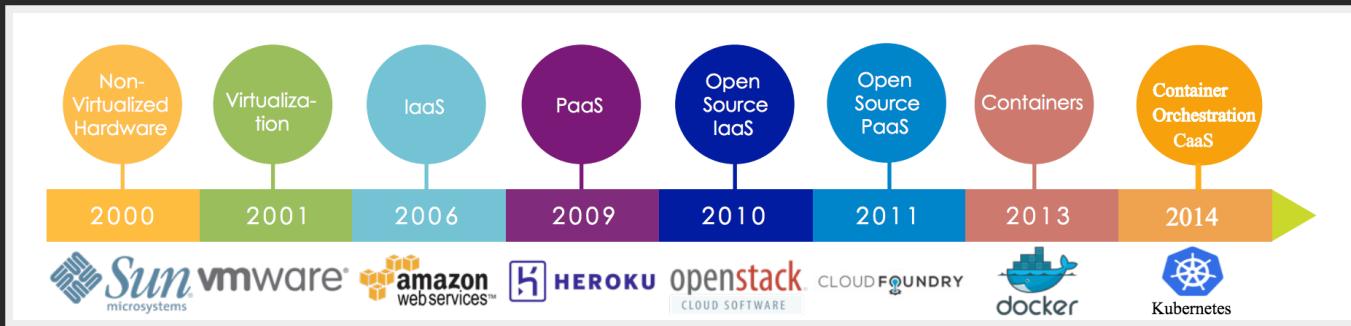
- Traditionelle analoge Prozesse und auf physischen Dingen basierende Geschäftsmodelle werden digitalisiert
- durch Möglichkeiten der Digitalisierung entstehen Disruptionen:
 - Netflix, Uber, Amazon, Spotify usw.
- Folgen: aus traditionellen Unternehmen ("Hardware Hersteller") werden IT-Service Unternehmen
- Cloud Computing ist ein Enabler für digitale Disruption, da diese die Skalierung der IT mit dem Geschäftsmodell ermöglicht

DIE VORTEILE VON CLOUD COMPUTING (WIEDERHOLUNG)

- Reduzierung von Total Cost of Ownership (TCO)
- Elastische Skalierung durch Automatisierung und Virtualisierung
- Agilität durch On-Demand Bereitstellung Infrastruktur und Services
- Kostenreduktion und Flexibilität durch dynamisches Kostenmodell (Pay-per-Use)
- Schnellere Time to Market, z.B. durch On-Demand Dienste, höhere Entwicklungsgeschwindigkeit
- Geografische Abdeckung des Cloud-Providers

TECHNISCHE ENTWICKLUNG & TRENDS IN CLOUD COMPUTING





- verändert, Original von [CNCF Keynote - A Brief History Of The Cloud](#)

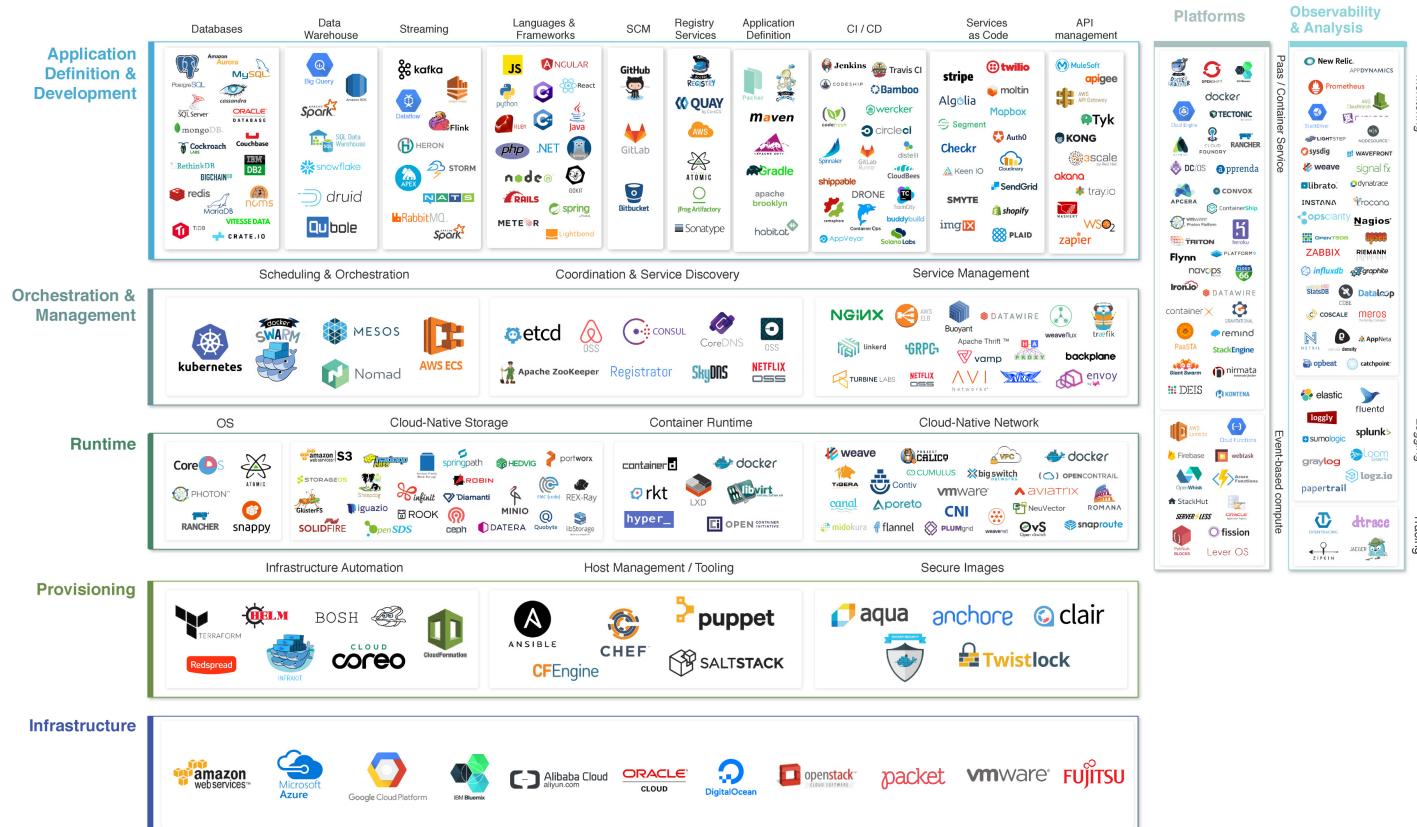
CLOUD NATIVE COMPUTING FOUNDATION

- 2015 gegründet, Teil der Linux-Foundation
- Open Source Konsortium um Open Source Cloud Computing insbesondere Container-Technologien zu promoten und zu steuern
- organisiert jährlich Konferenzen in USA, Europa und Asien
- identifiziert Technologien die für Cloud Computing relevant sind

CNCF LANDSCAPE 2017

Cloud Native Landscape

v0.9.4



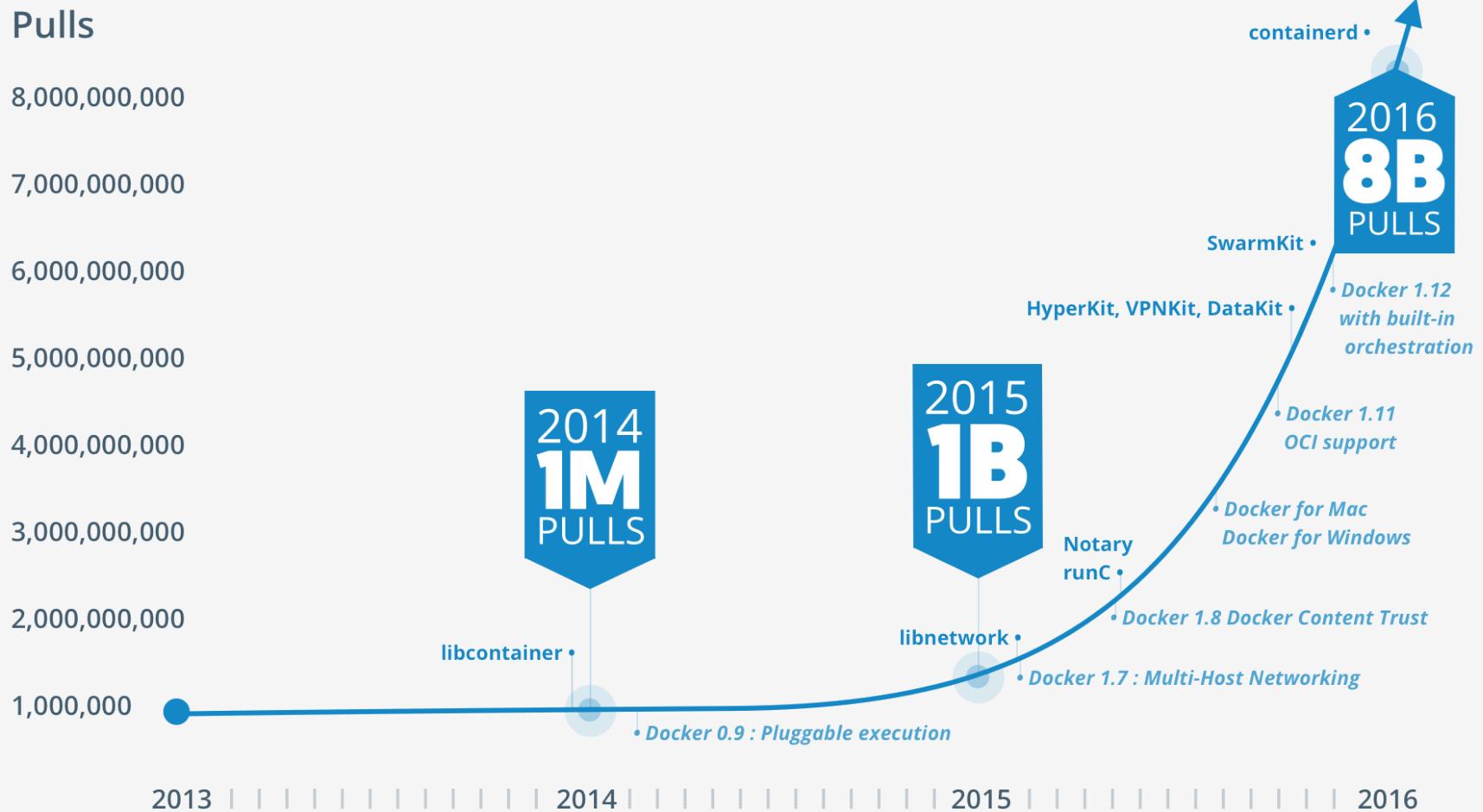
 <http://github.com/cncf/landscape>

 @dankohn1 @lennypruss @sraney

2. VERGANGENHEIT, GEGENWART UND ZUKUNFT

C. TECHNOLOGISCHE TRENDS

CONTAINER TECHNOLOGIE: DOCKER



CONTAINER TECHNOLOGIE: DOCKER

65%

use Docker to deliver development agility.

48%

use Docker to control app environments.

41%

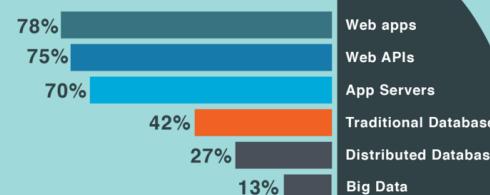
use Docker to achieve app portability.

90%

use Docker for apps in development.



Docker Workloads



58%

use Docker for apps in production.



90%

plan dev environments around Docker.



80%

plan DevOps around Docker.



CONTAINER TECHNOLOGIE: DOCKER

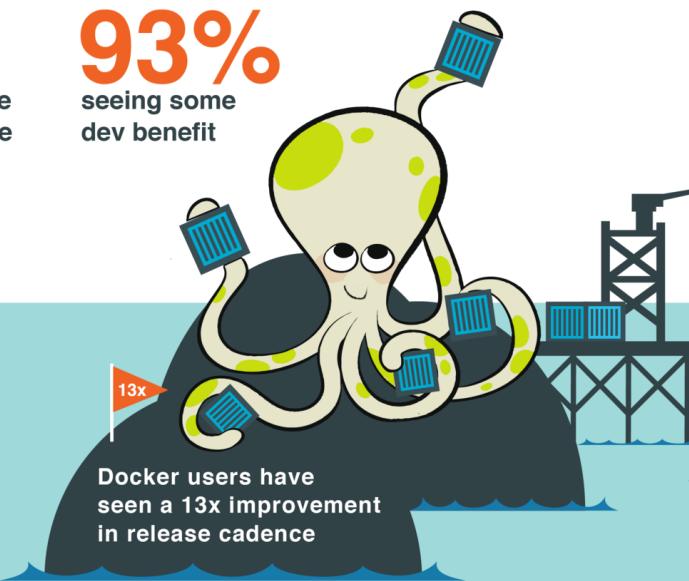
45%

of Docker users have been able to increase the frequency of software releases



93%

seeing some dev benefit

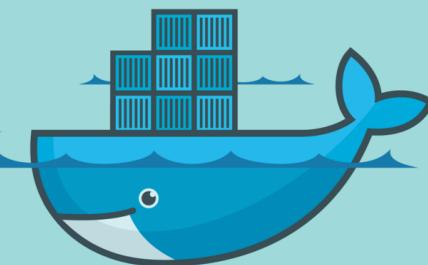


57%

Docker users have seen improvements in operational environment management

85%

seeing some ops benefit



70%

of Docker users say
'Docker has dramatically transformed... etc



62%

have seen improved MTTR on software issues.



CONTAINER TECHNOLOGIE: DOCKER

80%

say Docker is part
of cloud strategy

60%

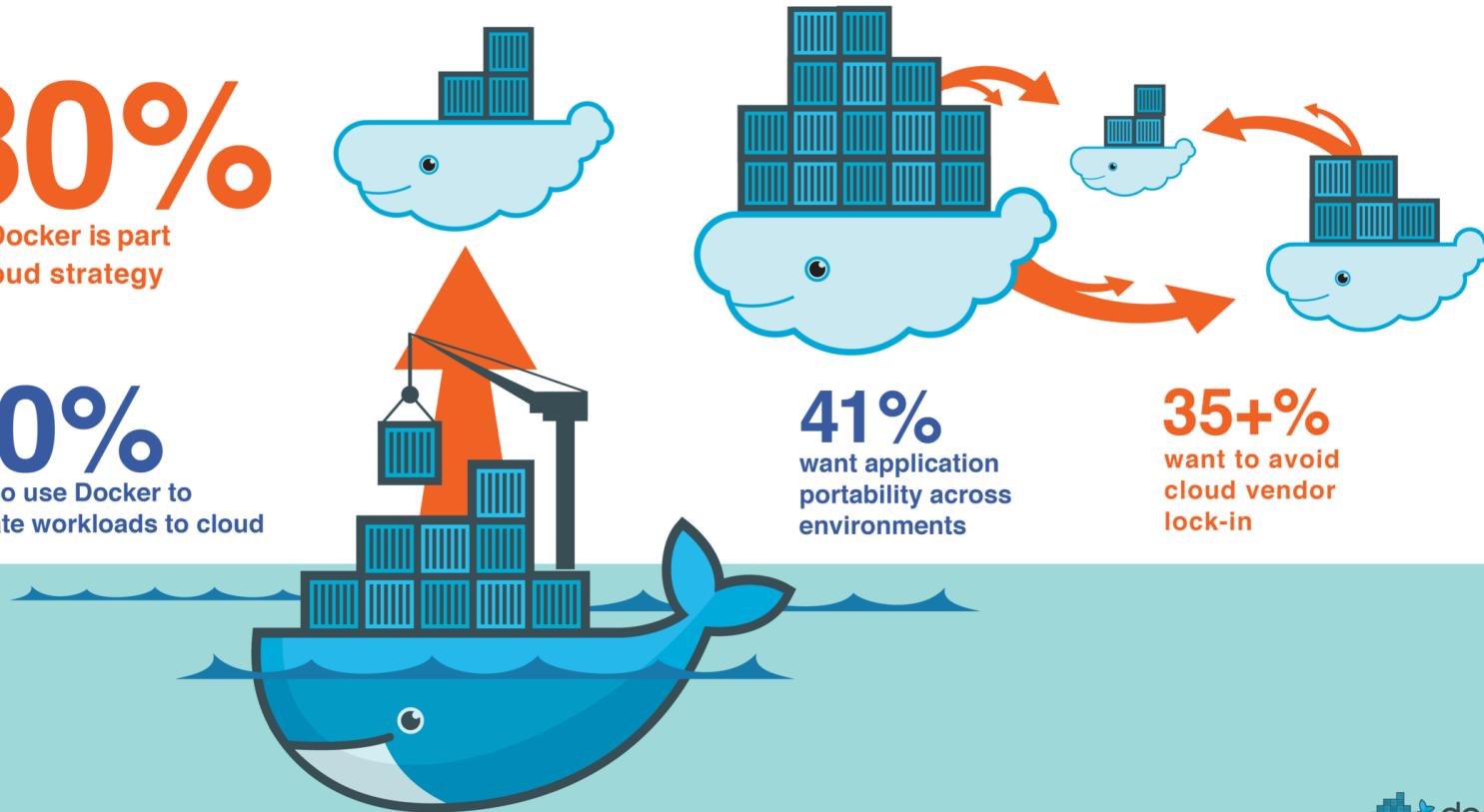
plan to use Docker to
migrate workloads to cloud

41%

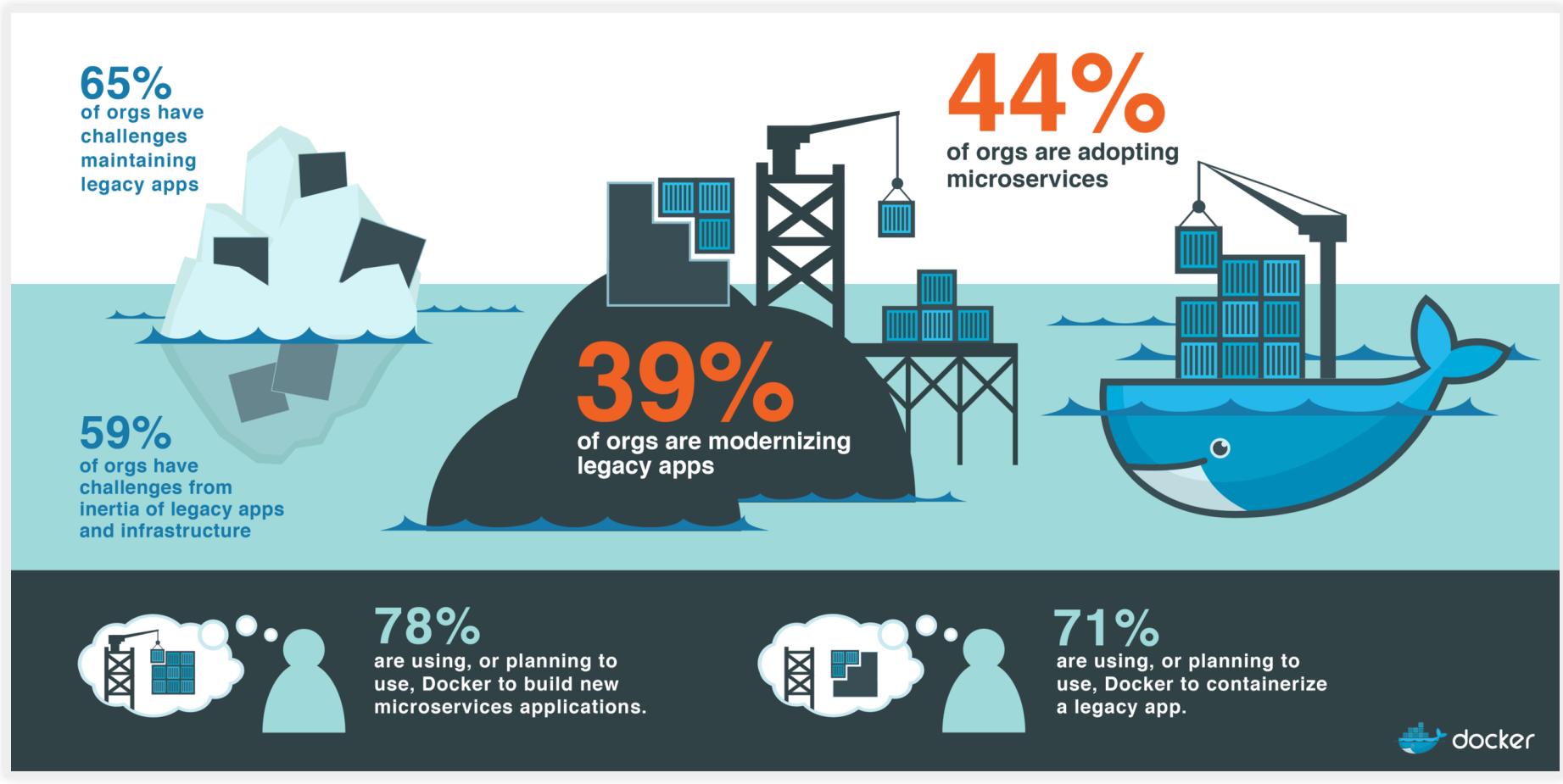
want application
portability across
environments

35+%

want to avoid
cloud vendor
lock-in



CONTAINER TECHNOLOGIE: DOCKER

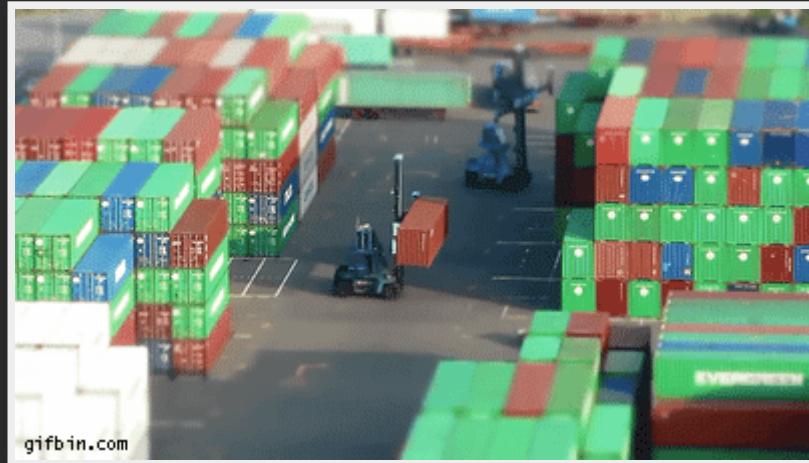


DOCKER DEMO (TEIL 2)

App-Containerisierung:



CONTAINER ORCHESTRIERUNG



CONTAINER ORCHESTRIERUNG

- Framework verantwortlich für Container Life-cycle, Scaling, Networking, Loadbalancing, Virtual Storage
- Container kommunizieren über ein privates Netzwerk miteinander
- Kommunikation nach extern über externes Netzwerk
- "Bring-your-own-Container"-Prinzip: die Plattform startet beliebige Container-Images
- Container Images werden in Image-Repositories hochgeladen (public/private)

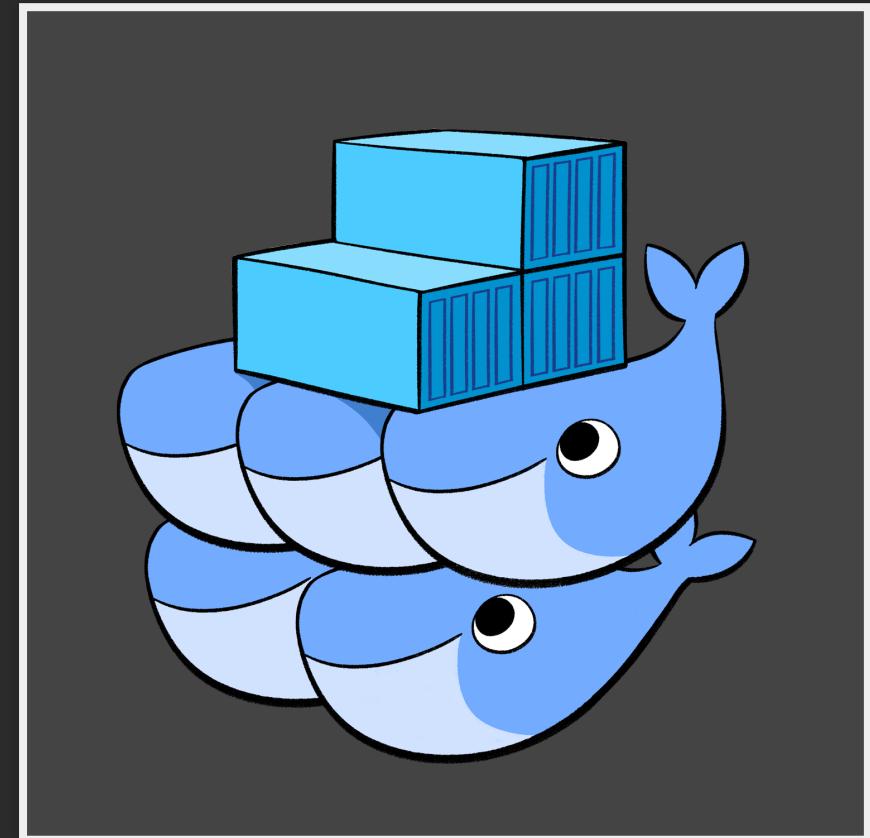
Bedeutung für Cloud Computing

- Basis für Container-as-a-Service (CaaS)
 - flexibler als PaaS, einfacher als IaaS
 - CaaS als Alternative zur PaaS und IaaS
 - DevOps-Orientiert
- Kein Vendor Lock-in
 - Kubernetes ist open-source, integrierbar in private / public cloud

Beispiel:

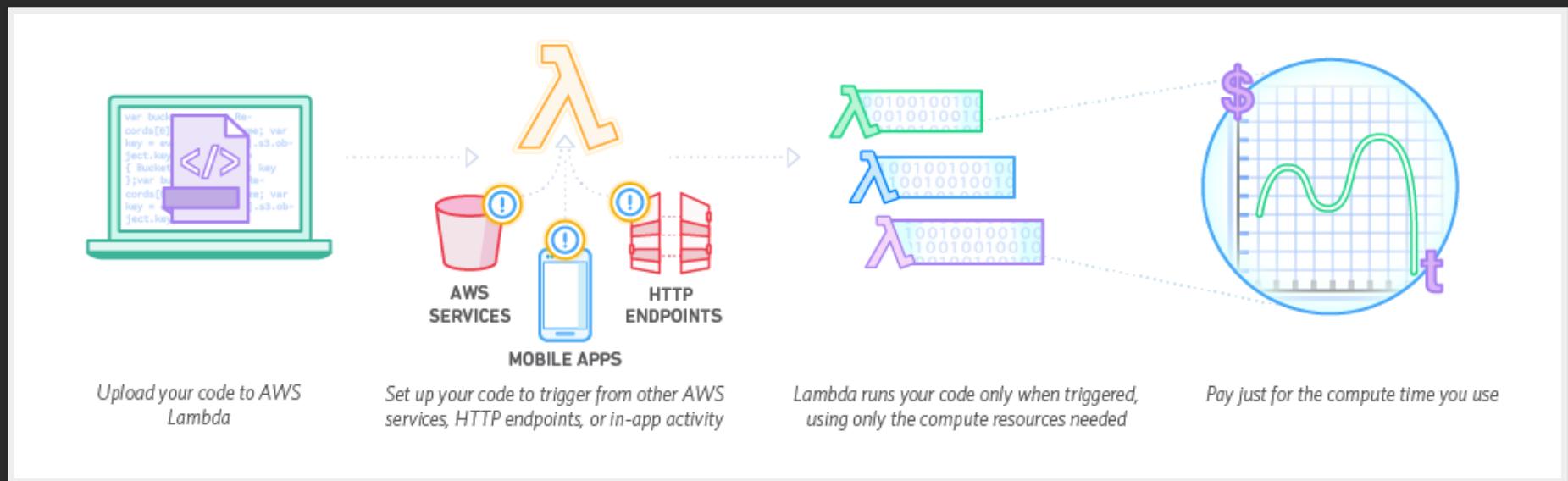


Kubernetes



Docker Swarm

SERVERLESS



- Serverless != Ohne Server
- kurzlebige Funktionen (stateless)
- In Serverless-Funktionen können On-Demand Rechenoperationen implementiert werden, die nicht ständig laufen müssen
- Lambda-Funktionen werden in der Cloud On-Demand erstellt, ausgeführt und beendet
- Die Kosten werden im Pay-per-Use Prinzip in Millisekunden abgerechnet

Bedeutung für Cloud Computing

- Rechenkosten können genauer abgerechnet werden (Pay-per-Use auf Millisekunden - Basis)
- Rechenkosten entstehen wirklich nur bei Bedarf
- Leichtgewichtige Funktionen können schnell entwickelt und in den Betrieb überführt werden
- Management und Betrieb von Funktionen liegen in der Verantwortung des Cloud-Providers

Beispiele:

AWS Lambda, Apache OpenWhisk, Google Cloud Functions

MICROSERVICES

- modernes / populäres Architekturmuster in der Cloud
- eignen sich für "App-Containerisierung"
- eignen sich für die horizontale Skalierung in der Cloud
- eignen sich für "Continuous Delivery"

PHILOSOPHIE

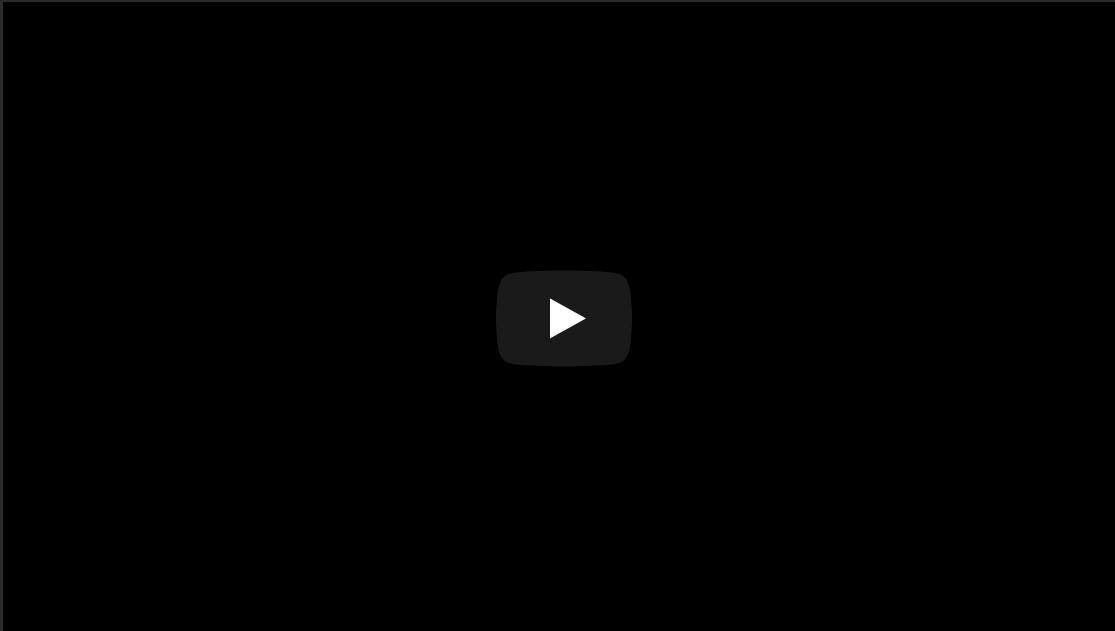
- Entkopplung von Funktionalitäten in kleine eigenständig lebende Funktionen
- Dekomposition von Applikationen in abgegrenzte, isolierte Module
- Überschaubarkeit, Einfachheit von Komponenten
- Offene Schnittstellen, keine proprietäre Protokolle, meistens Http + JSON/XML
- Programmiersprachenunabhängigkeit von Komponenten
- Entkopplung von Entwicklungsteams

VORTEILE

- unabhängige Skalierbarkeit von Microservices
- bessere Wartbarkeit, einfache Neuimplementierung
- versteckte Abhängigkeiten werden über die API erkennbar
- Sprachenunabhängig, Teams können nach ihren Wünschen den Technologiestack auswählen
- unabhängige Entwicklungszyklen, Parallelisierung der Entwicklung
- bei Überlastung (DDos) können sekundäre Services zugunsten von primären Services abgeschaltet werden

NACHTEILE

- Performance Overhead durch Netzwerk-Kommunikation
- Erhöhte Komplexität von Tests: Unit/Komponenten-Tests werden Integrationstests
- Erhöhte Komplexität von Monitoring: zentralisiertes Monitoring, Splunk, AppDynamics etc.
- Deployment Prozess muss ggf. zwischen Teams abgestimmt werden (API Breaking Changes)
- Einführung von generellen Problemen von verteilten Anwendungen: z.B. Lastverteilung, Zeitsynchronisation, distributed Locking



<https://www.youtube.com/embed/CKL3fV5UR8w>