

# **VL - CONNECTED MOBILITY - EINFÜHRUNG IN CLOUD COMPUTING**

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

**2018**

# AGENDA

## 1. CONNECTED MOBILITY - EINFÜHRUNG

- a. Begriffsdefinition
- b. Marktübersicht: Players & Use Cases
- c. 'Automotive meets Web'
- d. Workshop Session

# AGENDA

## 2. EIN KLEINER ABSTECHER - CLOUD COMPUTING

- a. Was ist Cloud Computing?
- b. Cloud Trends
- c. Workshop Session
- d. Cloud Architektur-Muster
- e. Skalierung
- f. Workshop Session

## 2. CLOUD COMPUTING

A. WAS IST CLOUD COMPUTING?

# WAS IST CLOUD COMPUTING?



# CLOUD COMPUTING ...

"beschreibt die **Bereitstellung von IT-Infrastruktur wie beispielsweise Speicherplatz, Rechenleistung oder Anwendungssoftware als Dienstleistung über das Internet.**"

- Wikipedia

[https://de.wikipedia.org/wiki/Cloud\\_Computing](https://de.wikipedia.org/wiki/Cloud_Computing), 10.April.2017

*"... Cloud Computing die Bereitstellung von Computingdiensten (Server, Speicher, Datenbanken, Netzwerkkomponenten, Software, Analyseoptionen und mehr) über das Internet („die Cloud“) ..." "... Clouddiensteanbieter stellen die Cloud Computing-Dienste üblicherweise basierend auf der jeweiligen Nutzung in Rechnung."*

- Microsoft Azure,

<https://azure.microsoft.com/de-de/overview/what-is-cloud-computing/>, 10.April.2017

## ALLGEMEIN ANERKANNTÉ DEFINITION

*"... a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*

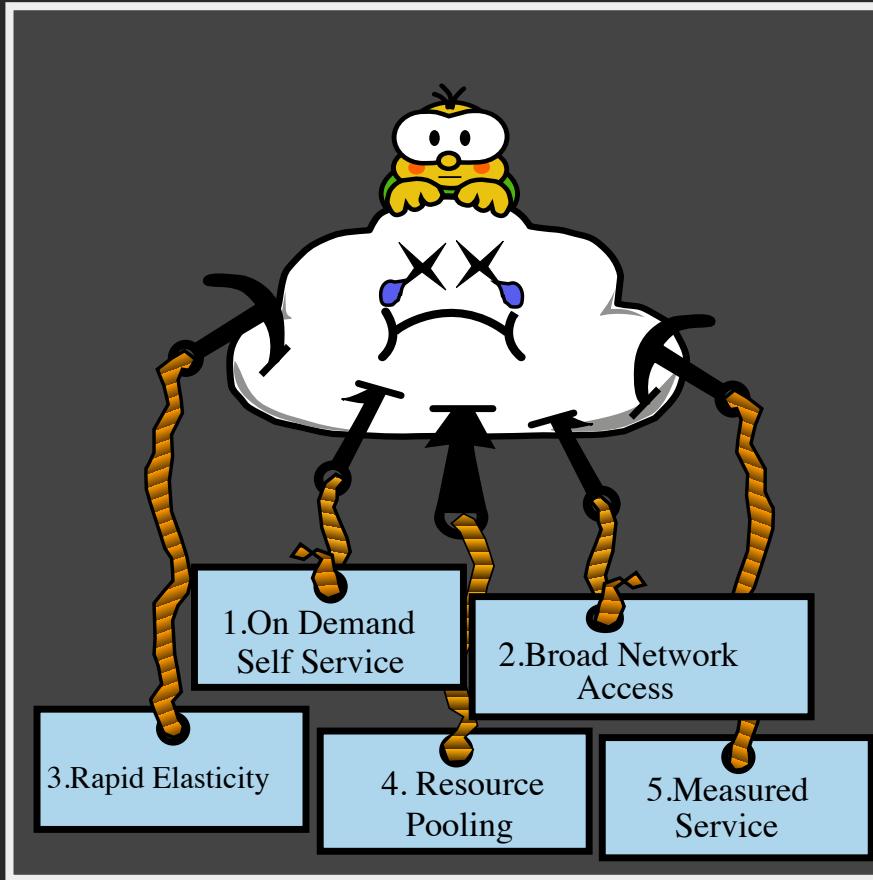
- National Institute of Standards and Technology  
(NIST), 2009

<http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>

# ESSENTIELLE EIGENSCHAFTEN (NIST)

1. Kunden können sich zu jeder Zeit selbst Services zuordnen/bestellen ohne dass der Betreiber etwas tun muss

3. Dienste können je nach Auslastung elastisch skalieren



2. Dienste der Cloud werden über das Netzwerk (Internet/Intranet) bereitgestellt

4. Ressourcen des Cloud-Betreibers sind in einem Pool abstrahiert und können erweitert werden

5. Dienste sind messbar, bspw. nach Ressourcenverbrauch, Nutzungsdauer, Auslastung. Auswertungen sind für Cloud Betreiber und Kunden verfügbar

# WEITERE TYPISCHE EIGENSCHAFTEN

- Geografische Verteilung
- Multi-Mandantenfähigkeit
- Hohe Ausfallsicherheit
- Sicherheit: Sicherheitsstandards, Verschlüsselung & Anonymisierung

# CLOUD COMPUTING VS IT OUTSOURCING

Worin unterscheidet sich Cloud Computing vom  
(klassischen) IT Outsourcing ?

Beispiel:

- Rechenzentrumsbetrieb wird ausgelagert
- Software wird per Spezifikation geschrieben und betrieben

# ALLGEMEINE VORTEILE VON CLOUD COMPUTING

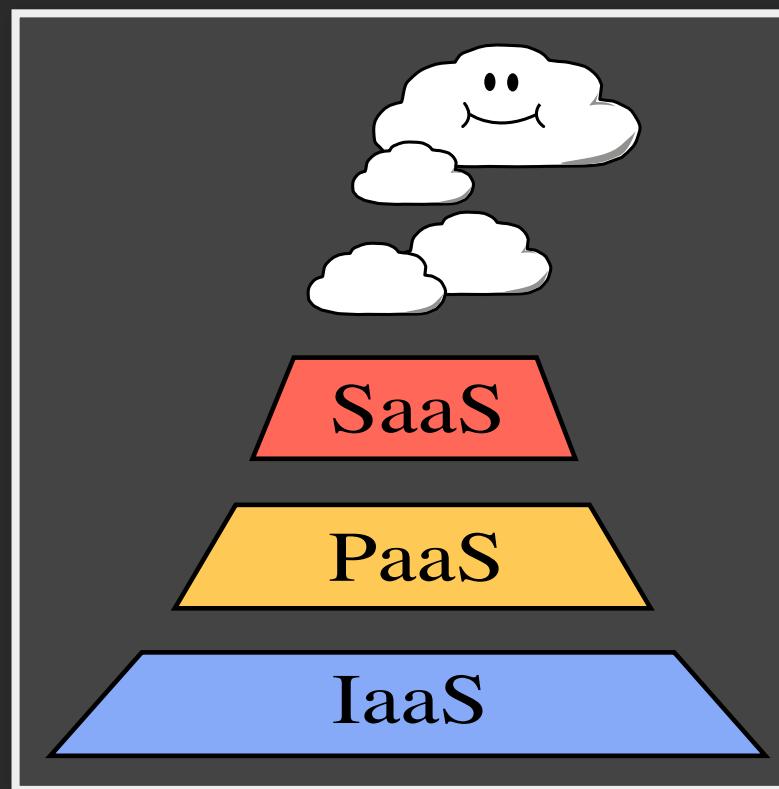
- Kostenreduzierung
- Skalierung, 'scale-as-you-grow'
- höhere Verfügbarkeit, Ausfallsicherheit
- höhere Sicherheit
- schnellere Entwicklungsgeschwindigkeit

# NACHTEILE VON CLOUD COMPUTING

- Kontrollverlust: Compute Location, Storage Location
- Überprüfbarkeit: Sicherheitsstandards, Datenschutz Standards
- Potentielle Abhängigkeit: Vendor Lock-in, Tech Lock-in
- Sicherheit: erhöhte Gefahr durch Erreichbarkeit über das Internet

# SERVICE MODELLE

NIST Definition: 3 Service Modelle, hierarchisch gegliedert



# NOCH MEHR ... "AS A SERVICE"

... mittlerweile gibt es unzählige Formen von Cloud-Dienstleistungen die '... as-a-Service' genannt werden

- Container-as-a-Service (CaaS)
- Mobile Backend-as-a-Service (mBaaS)
- Security-as-a-Service (SecaaS) ...
- Sammelbegriff: Anything-as-a-Service (XaaS)

**NOCH MEHR XAAS ! :)**

- Cat-as-a-Service (**CataaS**)
- Ransomware-as-a-Service (**RaaS**)
- Crime-as-a-Service (**Caas**)
- Game-as-a-Service (Cloud Gaming, **GaaS**)

# INFRASTRUCTURE AS A SERVICE (IAAS)

- Ist die unterste Schicht des NIST-Service Modells
- Der Cloud-Provider kümmert sich um den Rechenzentrumsbetrieb, Hardware, Netzwerk und Speicher
- Rechenressourcen, Speicher, Netzwerkressourcen werden virtualisiert bereitgestellt
- Durch Virtualisierung kann der Kunde On Demand VM Instanzen bestellen

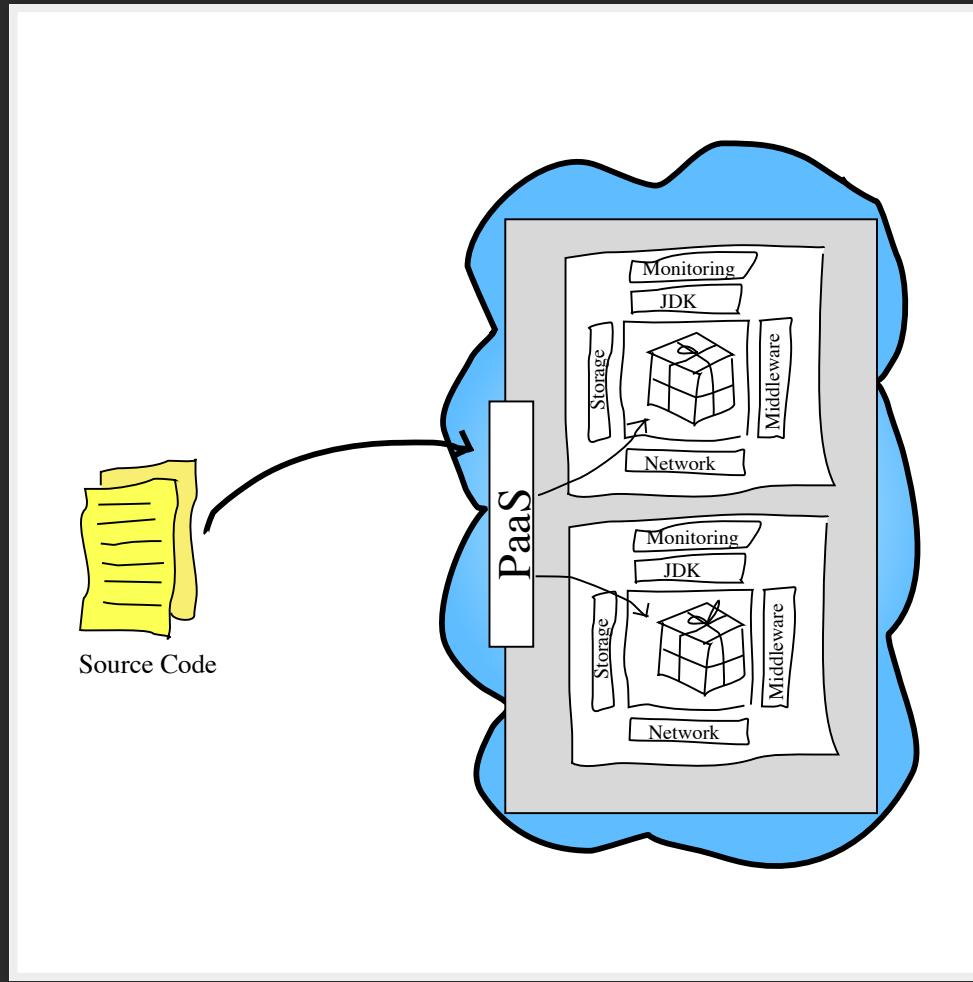
# IAAS - VORTEILE / NACHTEILE

- + freie Kontrolle über VMs, Betriebssystem, Software
- + Kosten nach Nutzung
- + kein Vendor Lock-in
- VM Betrieb in eigener Verantwortung
- langsame Entwicklungsgeschwindigkeit

# PLATFORM AS A SERVICE (PAAS)

- mittlere Schicht im NIST Service-Modell
- Der Cloud-Provider stellt die Infrastruktur und eine Cloud-Plattform bereit.
- Die Plattform stellt vorgefertigte Laufzeitumgebungen bereit auf denen die Kundensoftware läuft.
- Es müssen keine VMs gepflegt und betrieben werden, stattdessen kann sich der Kunde um seine Software kümmern.

# PAAS



# PAAS - VORTEILE/NACHTEILE

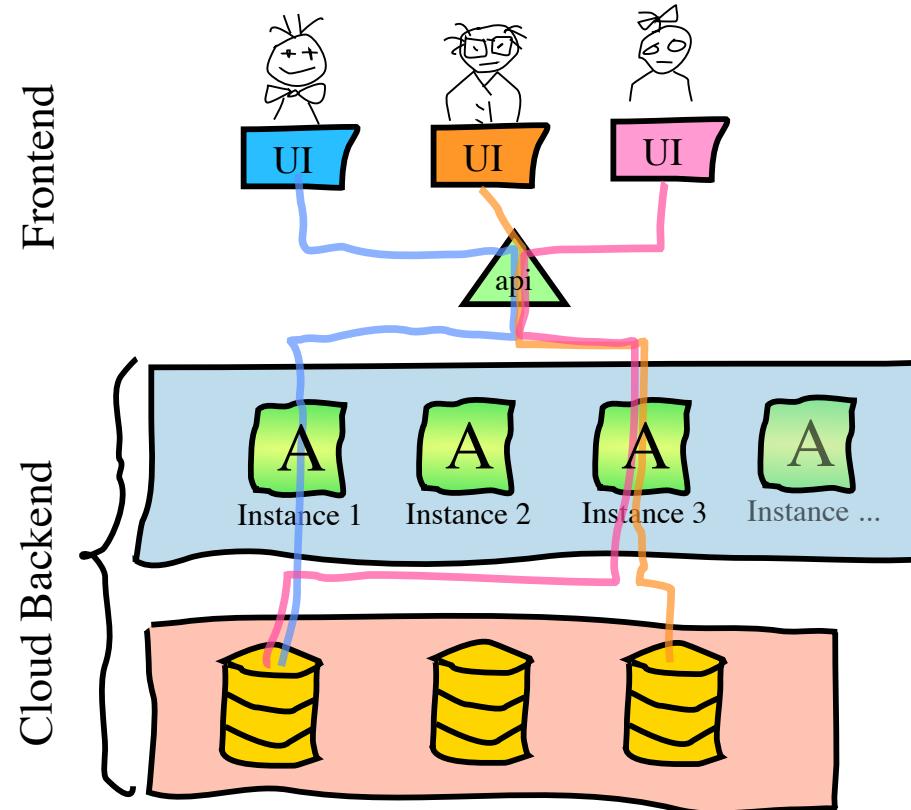
- + upload von Source-Code in die Cloud, schnellere Entwicklungsgeschwindigkeit
- + schnellere Time-to-Market
- + geringerer Aufwand beim Kompetenzaufbau, fördert "DevOps"
  
- Vendor lock-in möglich
- ggfs. veraltete Laufzeitumgebungen (Abhängigkeit vom Patching des Plattform Betreibers)

# SOFTWARE AS A SERVICE (SAAS)

- oberste Schicht im NIST Service Modell.
- Software wird als On-Demand Funktionalität zu jeder Zeit bereitgestellt.
- Der Cloud-Provider verantwortet den gesamten Stack: von Infrastruktur bis zur Applikation.
- Der Kunde ist lediglich nur noch für seine Daten verantwortlich.

# SAAS

## Multi Tenancy



## BEISPIELE

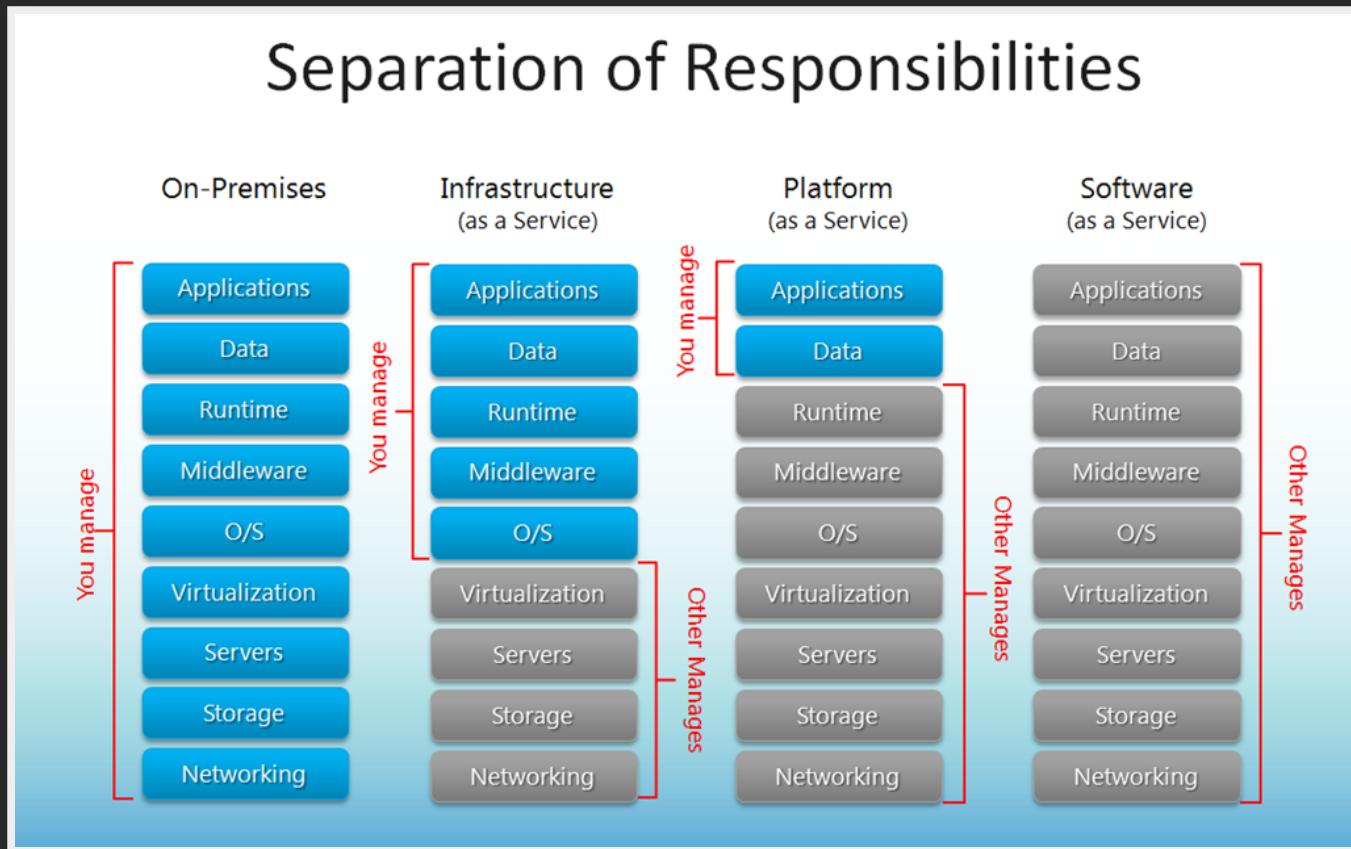
- Office 365, Gmail, Google Docs
- Netflix, Spotify
- Salesforce.com

# SAAS - VORTEILE/NACHTEILE

- + Fokus auf das Kerngeschäft
- + keine Verantwortung für Infrastruktur und Software
- + Mobilität - die Software ist von Überall erreichbar
  
- Vendor Lock-in wenn die Software exklusiv beim Cloud-Provider liegen
- nur Standardsoftware erhältlich, nur eingeschränktes customizing

# VERANTWORTLICHKEIT

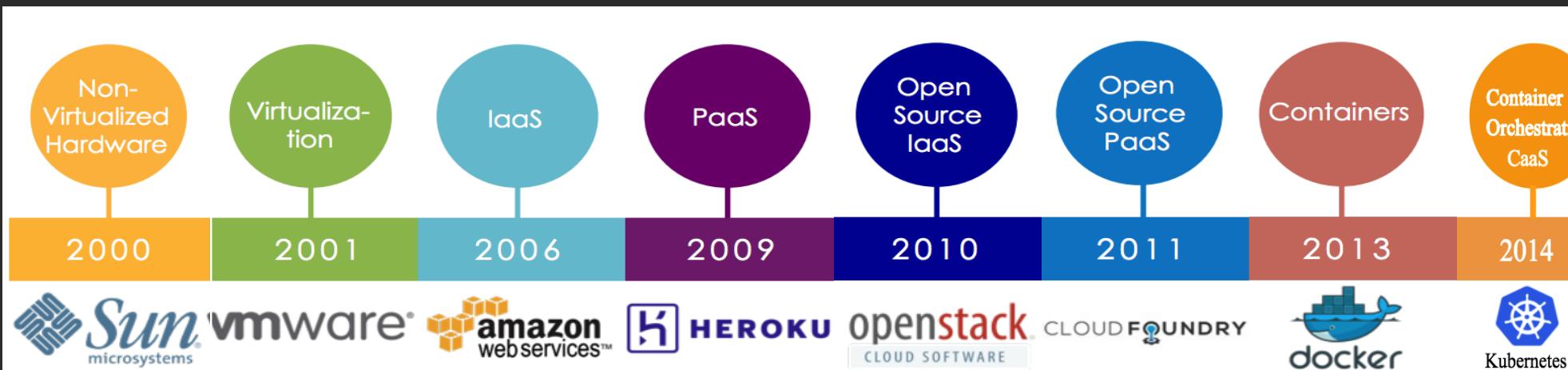
## Separation of Responsibilities

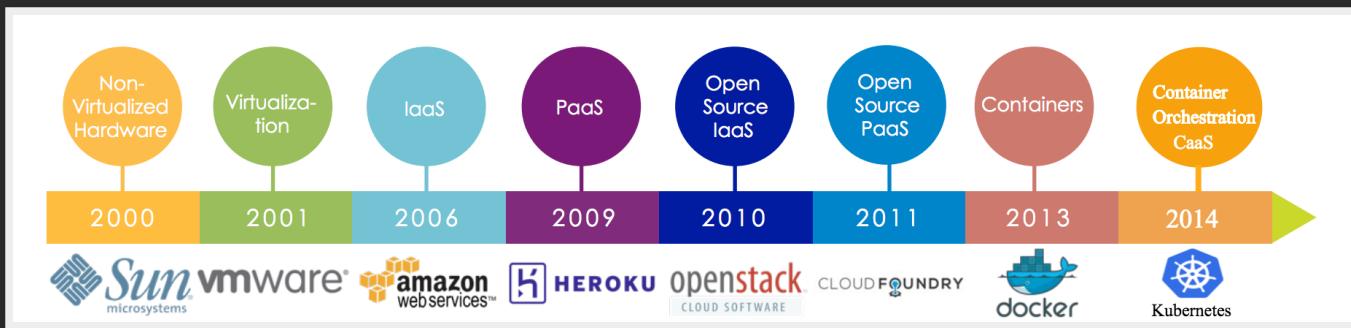


# 2. CLOUD COMPUTING

## B. CLOUD COMPUTING TRENDS

# TECHNISCHE ENTWICKLUNG & TRENDS IN CLOUD COMPUTING





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

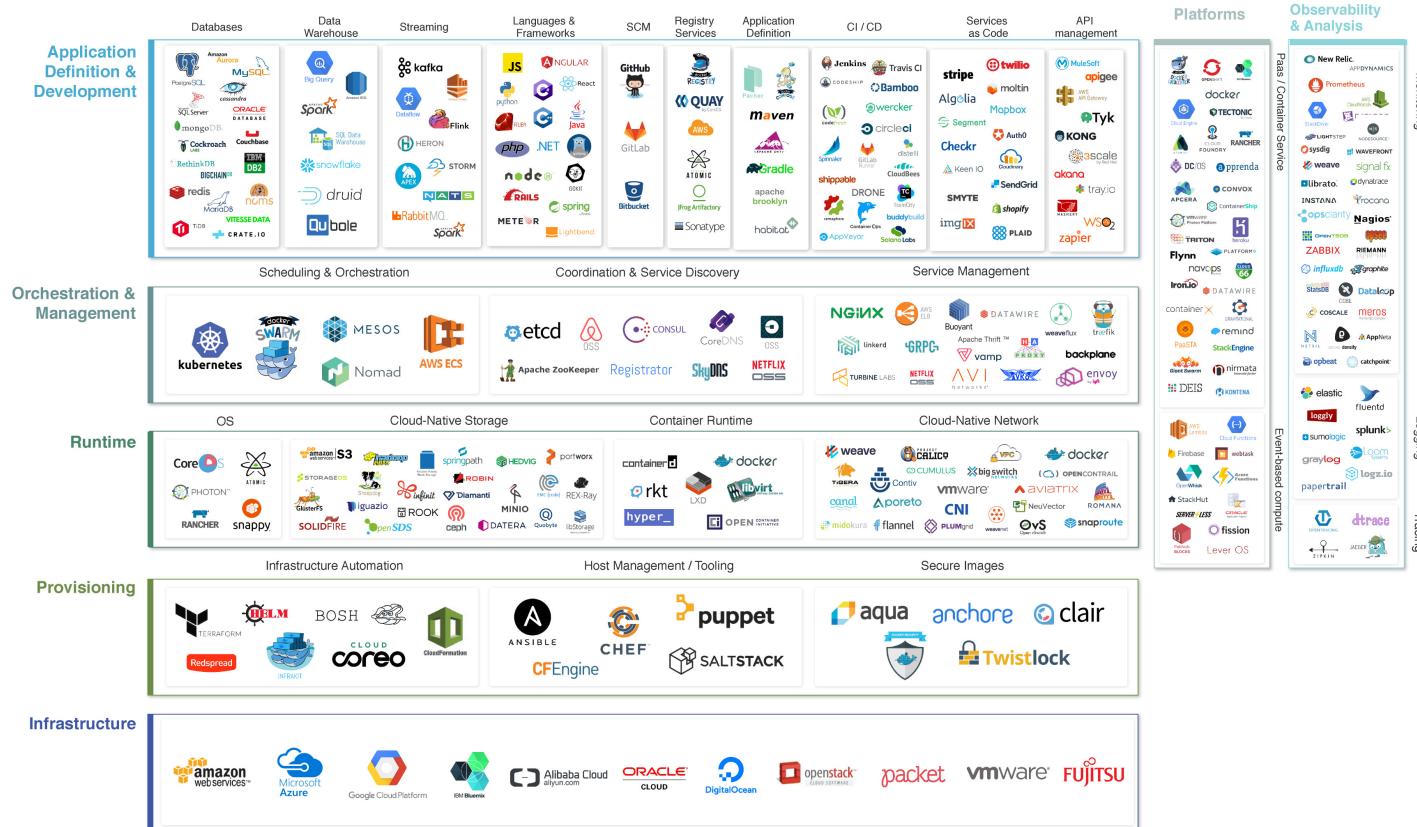
## CLOUD NATIVE COMPUTING FOUNDATION

- 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

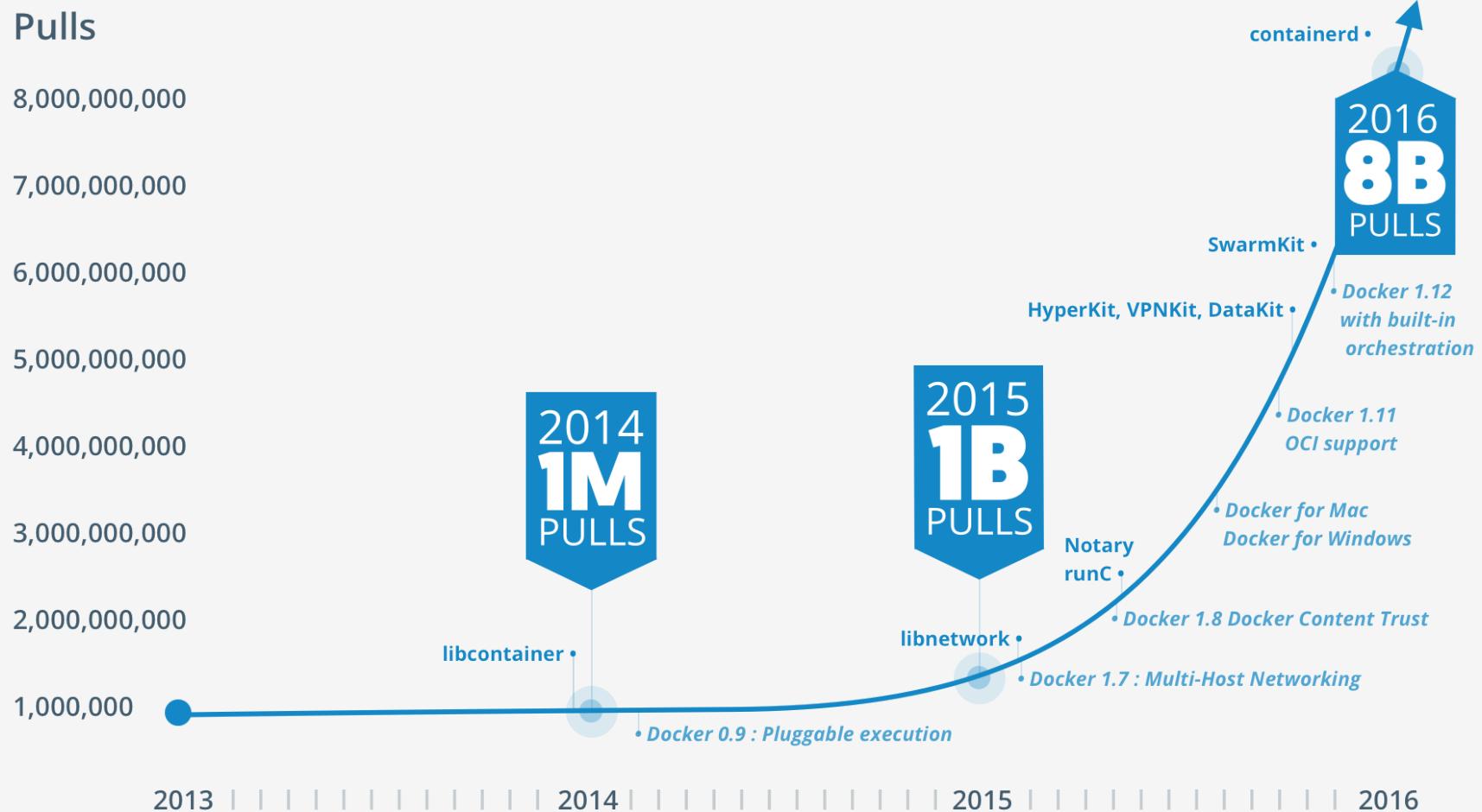
v0.9.4



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

 @dankohn1 @lennypruss @sraney

# CONTAINER TECHNOLOGIE: DOCKER



# CONTAINER TECHNOLOGIE: DOCKER

**65%**

use Docker to deliver development agility.

**48%**

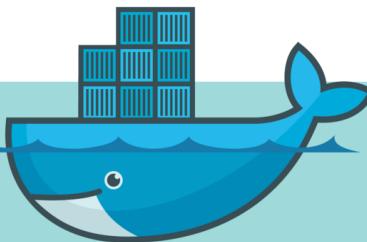
use Docker to control app environments.

**41%**

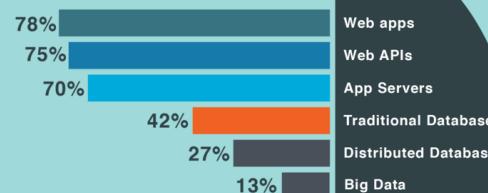
use Docker to achieve app portability.

**90%**

use Docker for apps in development.



Docker Workloads



**58%**

use Docker for apps in production.



**90%**

plan dev environments around Docker.



**80%**

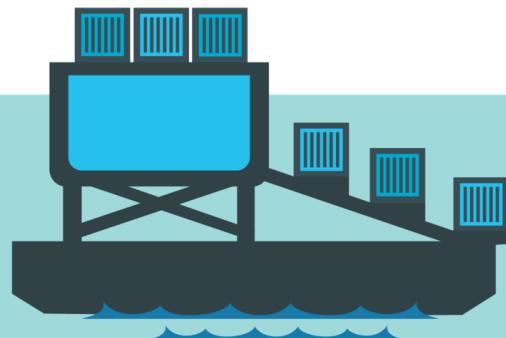
plan DevOps around Docker.



# CONTAINER TECHNOLOGIE: DOCKER

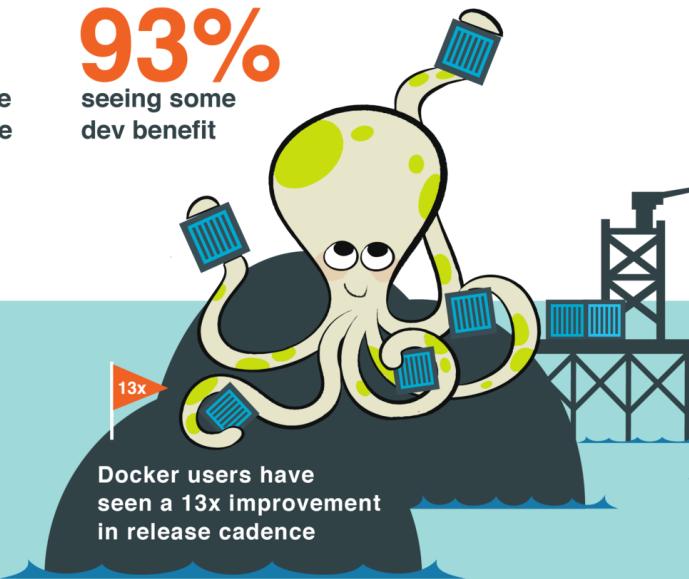
**45%**

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%**

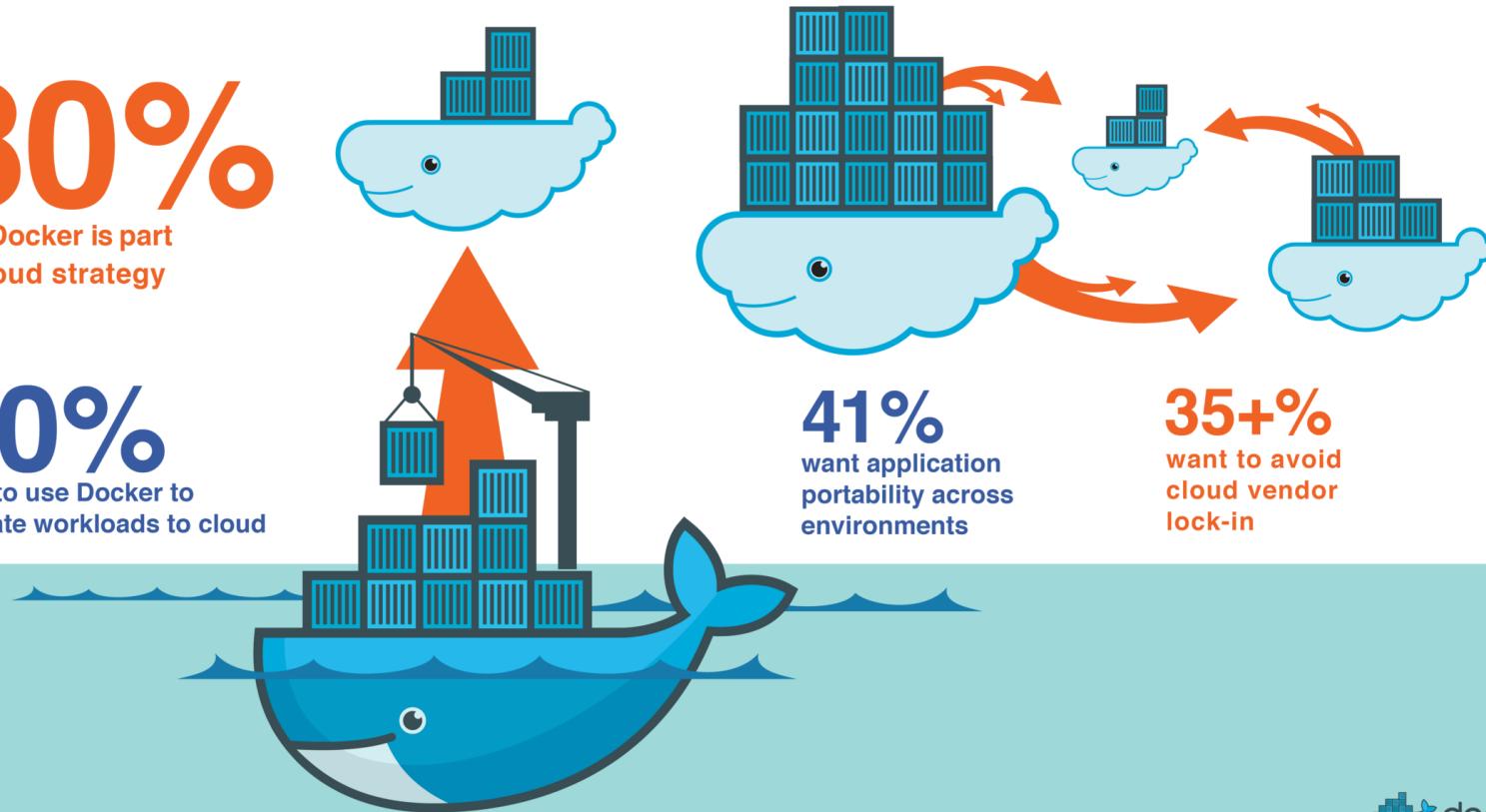
plan to use Docker to  
migrate workloads to cloud

**41%**

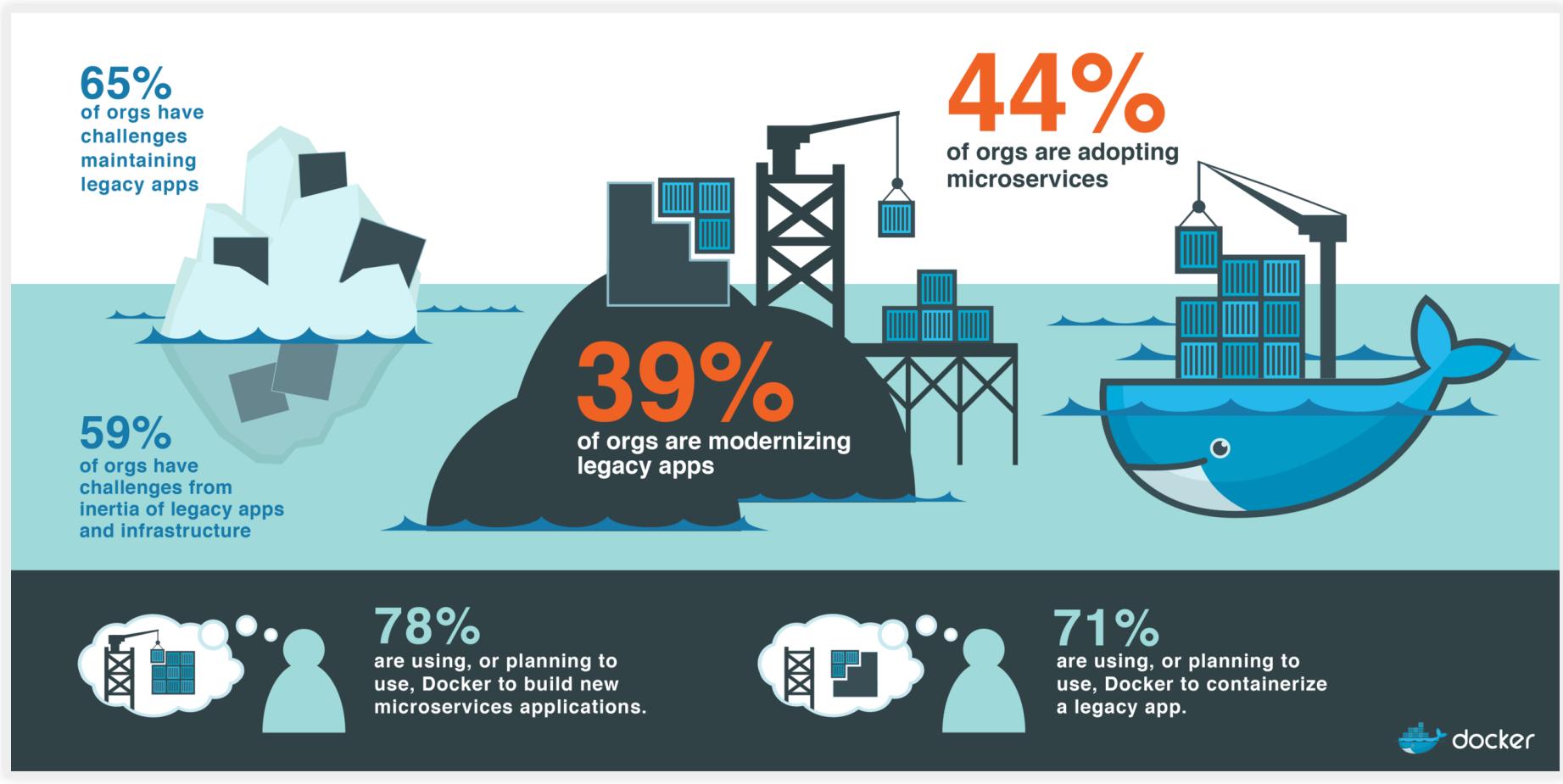
want application  
portability across  
environments

**35+%**

want to avoid  
cloud vendor  
lock-in



# CONTAINER TECHNOLOGIE: DOCKER

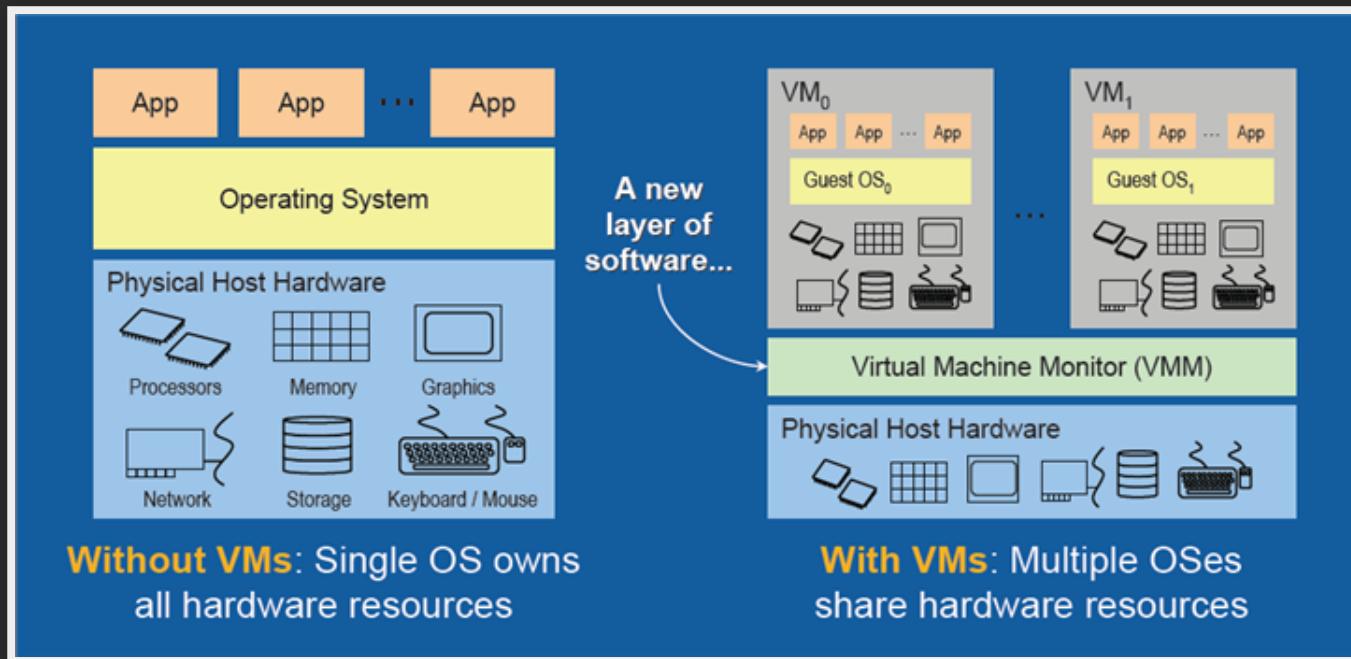


# WIE FUNKTIONIERT DOCKER ?

Betriebssystem-level Virtualisierung

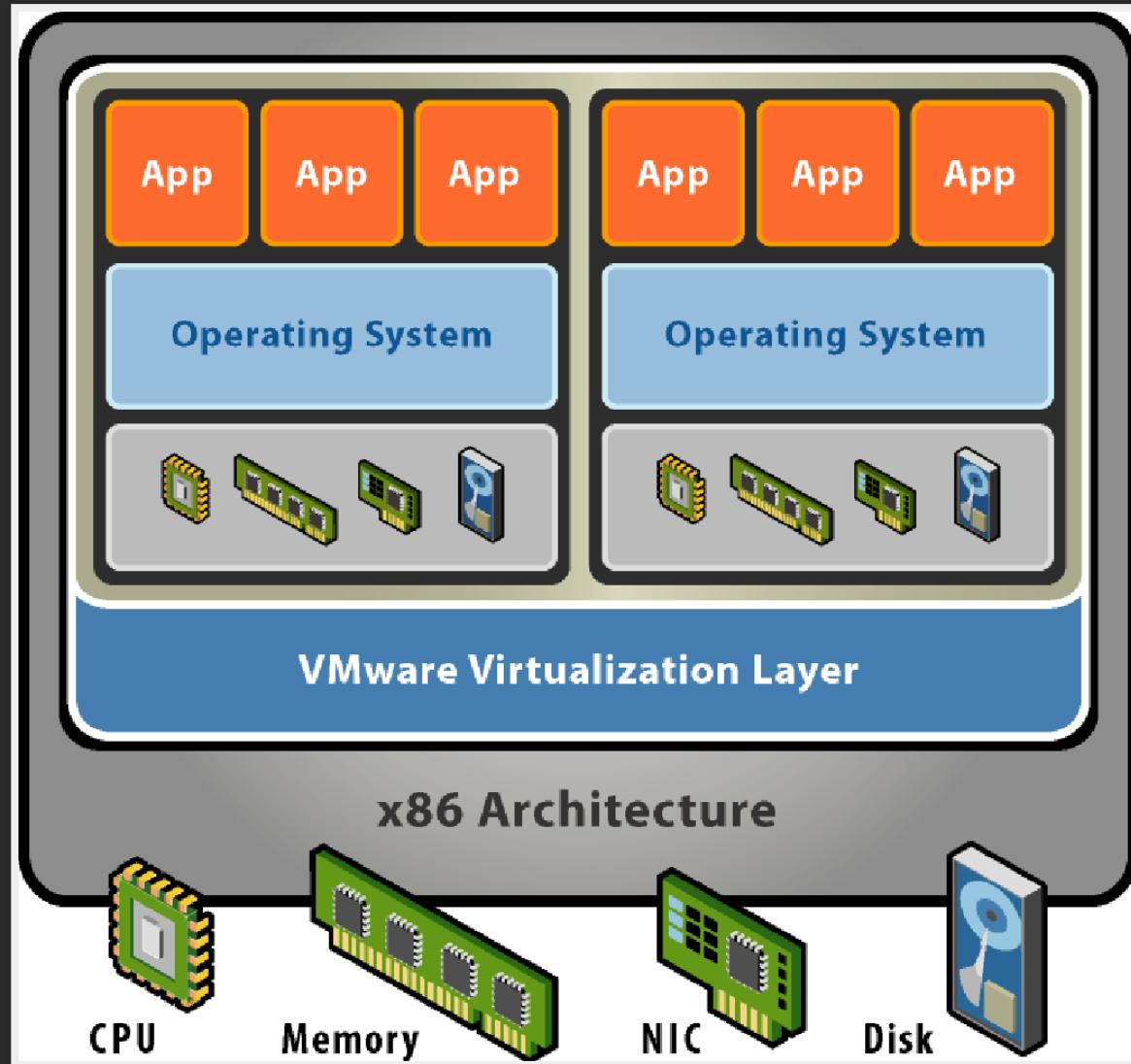
- Systemvirtualisierung vs Betriebssystem-Virtualisierung

# SYSTEMVIRTUALISIERUNG



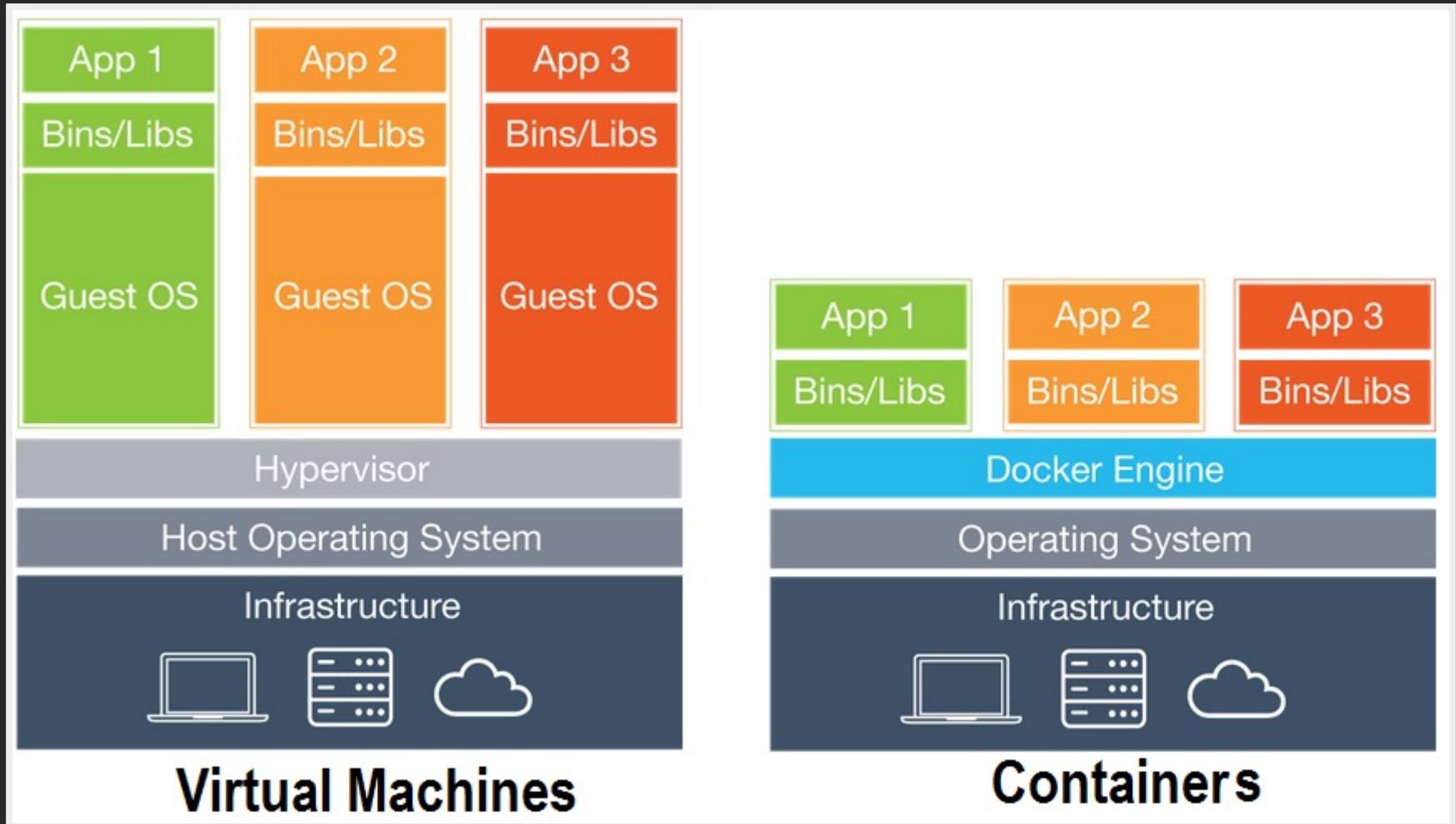
<https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise>

# SYSTEMVIRTUALISIERUNG



# BETRIEBSSYSTEM-VIRTUALISIERUNG

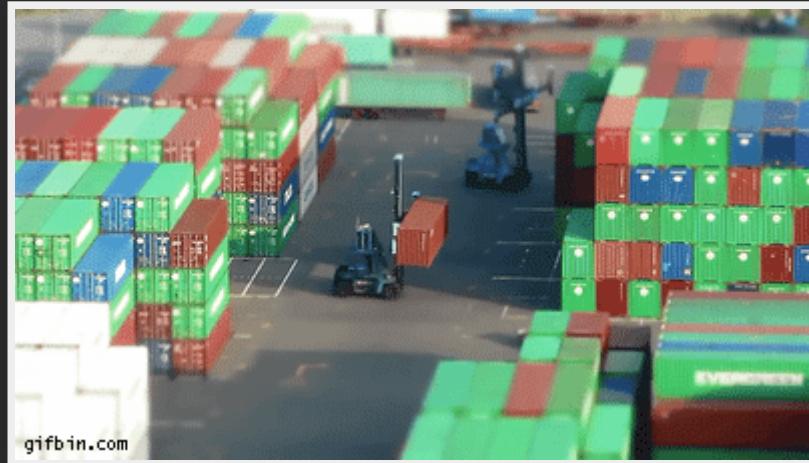
# BETRIEBSSYSTEM-VIRTUALISIERUNG VS SYSTEMVIRTUALISIERUNG



# FRAGE: WIE NENNT MAN FOLGENDER ART VON SOFTWARE?



# CONTAINER ORCHESTRATION



# 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)

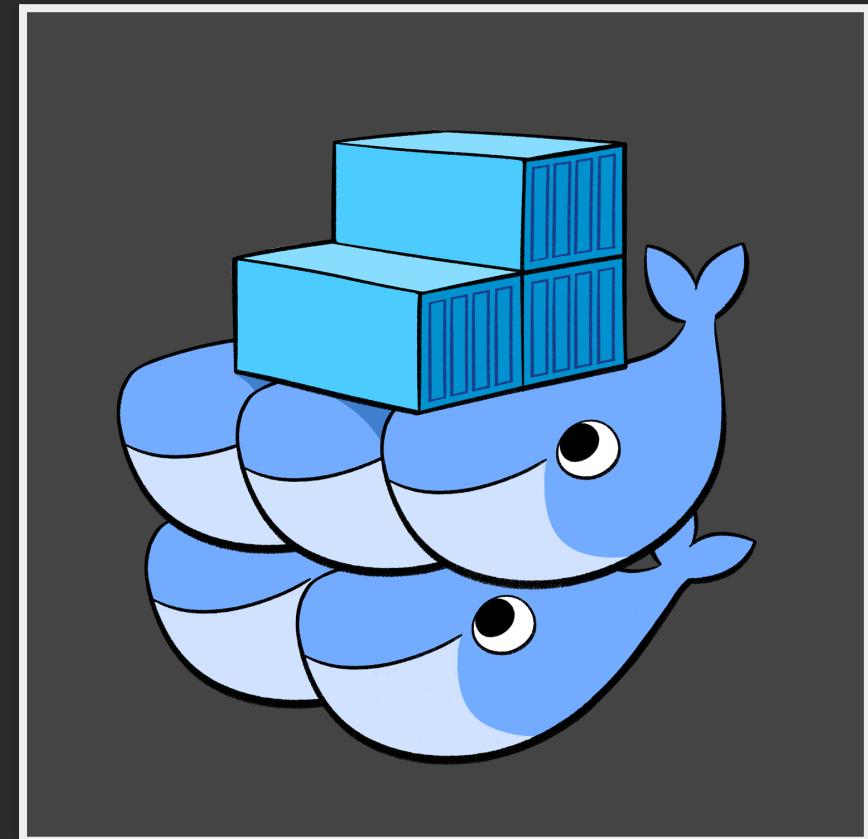
## Bedeutung für Cloud Computing

- 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:

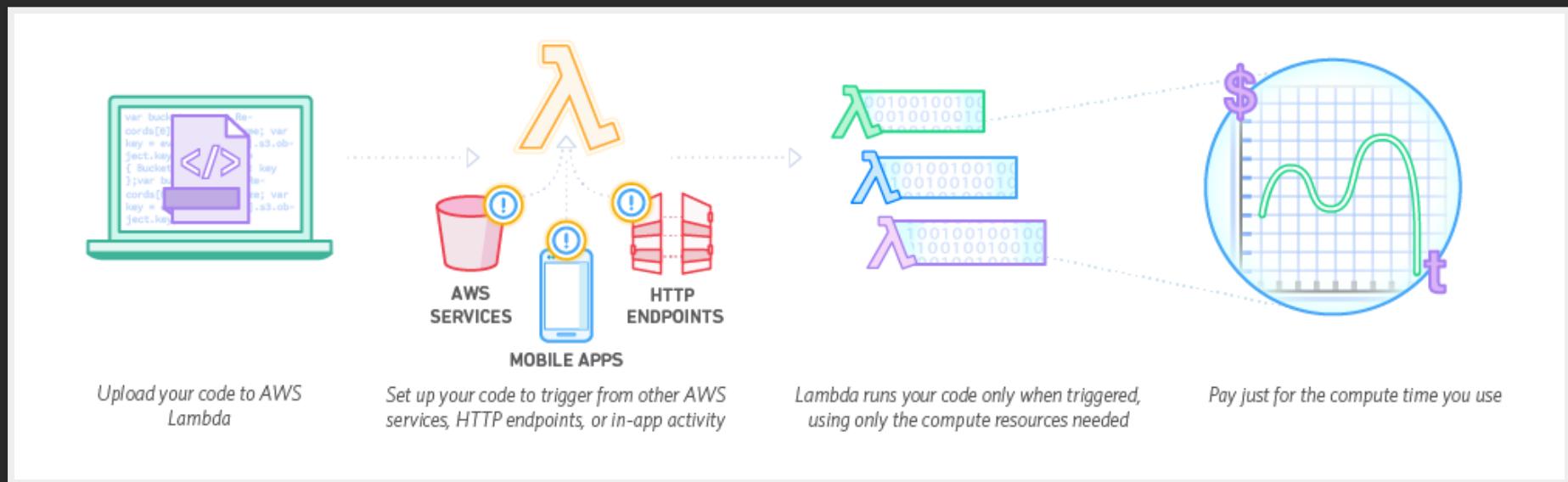


Kubernetes



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:

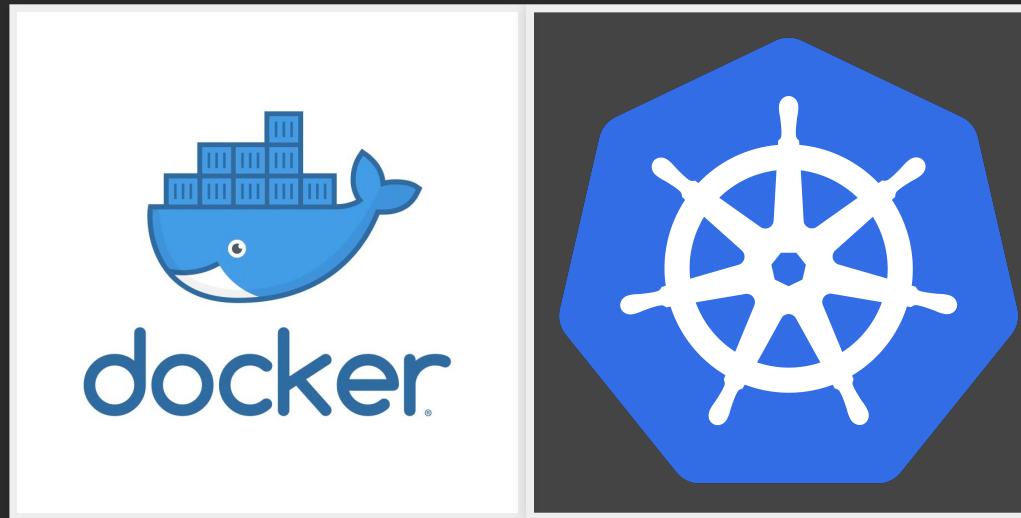
AWS Lambda, Apache OpenWhisk, Google Cloud Functions



# 2. CLOUD COMPUTING

## C. WORKSHOP SESSION

# WORKSHOP SESSION: DOCKER & KUBERNETES



<https://www.docker.com>

<https://kubernetes.io/>

# ZIEL

- Erstellung eigener Docker Container
- Betrieb von Docker Container in der Cloud

# WERKZEUGE

- Docker (docker)
- Azure CLI (az)
- Kubernetes CLI (kubectl)

# AUFGABE 1

Erstellt ein Docker Image für die Hello-World App:

- `docker run --name hello world -p 80:80 -it hello-world-app`

## AUFGABE 2

Betreibt die Hello-World App im MS Azure AKS Cluster:

- die App soll über das Internet zugreifbar sein
- RESTful API soll 'CarData' in MongoDB abspeichern

## AUFGABE 3

Betreibt die Hello-World App als MS Azure Function:

- die App soll über das Internet zugreifbar sein
- alle bisherigen RESTful API functions sollen implementiert werden

# 2. CLOUD COMPUTING

D. CLOUD ARCHITEKTUR-MUSTER

# 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

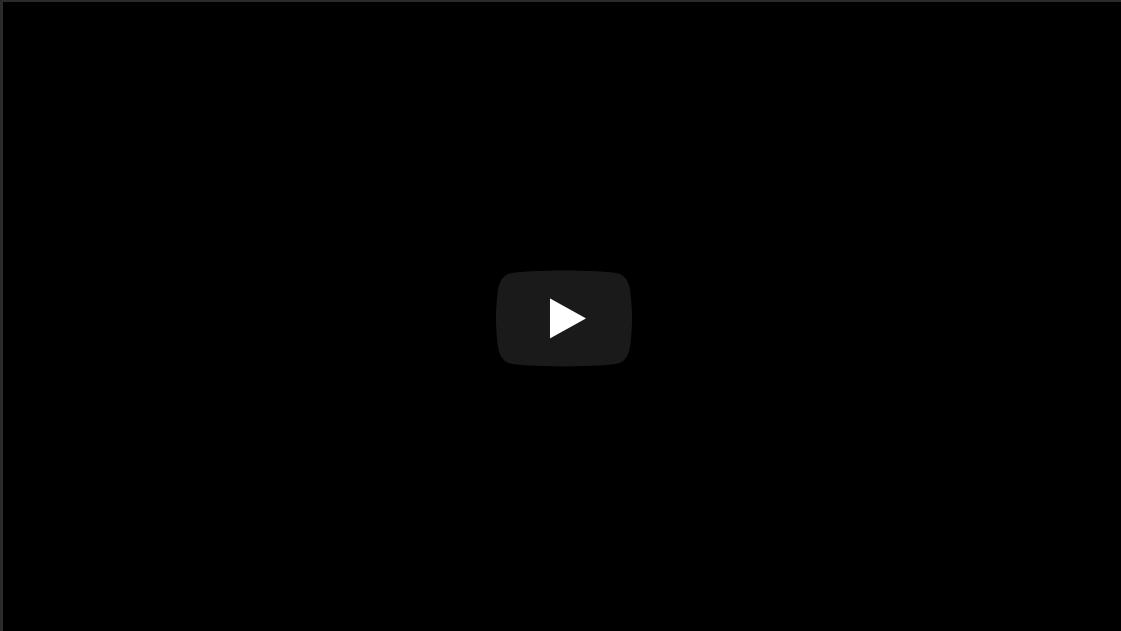
- Entkopplung von Funktionalitäten in kleine eigenständig lebende Funktionen
- Dekomposition von Applikationen in abgegrenzte, isolierte Module
- Überschaubarkeit, Einfachheit von Komponenten
- Offene Schnittstellen, keine proprietäre Protokolle, meistens Http + JSON/XML
- Programmiersprachenunabhängigkeit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

# 12-FACTOR APPS

The Twelve Factors

| #    | Factor              | Description   |
|------|---------------------|---|
| I    | Codebase            | There should be exactly one codebase for a deployed service with the codebase being used for many deployments.      |
| II   | Dependencies        | All dependencies should be declared, with no implicit reliance on system tools or libraries.                        |
| III  | Config              | Configuration that varies between deployments should be stored in the environment.                                  |
| IV   | Backing services    | All backing services are treated as attached resources and attached and detached by the execution environment.      |
| V    | Build, release, run | The delivery pipeline should strictly consist of build, release, run.   |
| VI   | Processes           | Applications should be deployed as one or more stateless processes with persisted data stored on a backing service. |
| VII  | Port binding        | Self-contained services should make themselves available to other services by specified ports.                      |
| VIII | Concurrency         | Concurrency is advocated by scaling individual processes.   |
| IX   | Disposability       | Fast startup and shutdown are advocated for a more robust and resilient system.                                     |
| X    | Dev/Prod parity     | All environments should be as similar as possible.  |
| XI   | Logs                | Applications should produce logs as event streams and leave the execution environment to aggregate.                 |
| XII  | Admin Processes     | Any needed admin tasks should be kept in source control and packaged with the application.                          |

- Wikipedia

## DISKUSSION

Was ist der Sinn und Zweck von 12 Factor Apps

- 15 min.



# 2. CLOUD COMPUTING

## E. SKALIERUNG

# AUTO-SCALING

- Essentieller Mechanismus im Cloud Computing um auf den Bedarf von Ressourcen zu reagieren
- dynamische Bereitstellung von VMs, Container, Speicher, Storage
- bidirektional: Scale-Up, Scale-Down
- Nutzen:
  - Elastizität: bessere Auslastung, Kostenoptimierung (Kosteneinsparung)
  - Verfügbarkeit und Zuverlässigkeit: durch Sicherstellung von min. Instanzen

## AUTOSCALING STRATEGIEN

- Reaktiv: Cloud Umgebung beobachtet Kernmetriken (CPU, Memory, Traffic) und reagiert auf Last
- Proaktiv:
  - Scheduling (scheduled scaling): Zeitlich geplante Skalierung auf Basis von Erfahrungswerten
  - Prädiktiv (predictive scaling): durch Vorhersage auf Basis von prädiktiver Analyse

## AUTO-SCALING BEI AWS (IAAS)

- CPU, Speicherauslastung, Netzwerk Traffic wird durch AWS EC2 überwacht
- alle Logs/Stdout werden an AWS Cloudwatch übermittelt (Logging/Monitoring-Service)

<http://docs.aws.amazon.com/autoscaling/latest/userguide/WhatIsAutoScaling.html>

- AMIs/VMs können zu Auto-Scaling Groups hinzugefügt werden, Konfiguration von Scaling-Plans:
  - Scaling Policies: reaktiv (standard), scheduled-scaling, dynamisch (auf Basis von Metriken aus Amazon SQS, Cloudwatch alarms)
  - min./max. und gewünschte Anzahl an Instanzen

<https://docs.aws.amazon.com/autoscaling/latest/userguide/as-using-sqs-queue.html>

## AUTO-SCALING BEI AWS (IAAS)

- Ausführung: auf Basis von Scaling Plans werden AMI Instanzen hinzugefügt / terminiert
- hinzufügen von Instanzen (auf Basis von Launch-Configuration):
  - Instanziierung aus AMI
  - boot-up Phase, Initialisierung, Konfigurierung per Skript (siehe AMI Design Strategien)
  - Anbindung an Cloud-Storage
  - Registrierung ins Virtual Private Cloud (Netzwerk)
  - Registrierung beim Elastic Load-Balancer

## AUTO-SCALING BEI AWS (IAAS)

- Status Überwachung: Health-Checking
  - über HTTP-Endpunkt: z.B. /heathcheck
  - healthy flag

## AUTO-SCALING BEI KUBERNETES (CAAS)

- Monitoring / Überwachung von Basis-Metriken: CPU, Memory-Auslastung
- Skalierung auf Pod-Ebene (Kubernetes Konzept: Zusammenfassung von Containern)
- Skalierung über die Definition einer Auto-Scaling Konfiguration
- Konfiguration wird über Selektoren auf Kubernetes Bausteine (Pod, ReplicaSets, Deployments) angewendet
- Auto-Scaling ist reaktiv

## AUTO-SCALING BEI KUBERNETES (CAAS)

- Ausführung: auf Basis der Auto-Scaler Konfiguration werden Container-Instanzen hinzugefügt / terminiert
- hinzufügen von Instanzen:
  - Pod Instanz wird erstellt, alle Container Definitionen werden aus dem konfiguriertem Image instanziert
  - Einbinden der Pod/Container Instanz ins private virtual network
  - Container Prozess wird gestartet (run-Befehl)
  - Registrierung beim Service und Load-Balancer

## AUTO-SCALING BEI KUBERNETES (CAAS)

- Status Überwachung: Health-Checking
  - über HTTP-Endpunkt: z.B. /heathcheck, (http-response-Code: 200 bedeutet "OK", ansonsten "NOK")
  - Überwachung der Pod/Container Prozesse: terminiert der Pod/Container Prozess, stoppt der Pod/Container, es wird ein neuer Pod/Container gestartet



# 2. CLOUD COMPUTING

## F. WORKSHOP SESSION

# WORKSHOP SESSION: MICROSERVICE MESH



# ZIEL

- Erstellung eines Microservice Backends in der Cloud
- Realisierung eines simplen Connected Mobility Use Cases

# WERKZEUGE

- Microsoft Azure
- Kubernetes
- NodeJS

## AUFGABE 1

Erstellt für einen simplen Connected Mobility Use Case  
das Backend System in der Cloud

Use-Case Beispiel:

- Auto Lokalisationsdienst
- Flottenmanagement (bspw. für Taxi Unternehmen)