TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Prof. Distributed and Networked Systems
Fakultät Informatik

Vorlesung „Service and Cloud Computing“

4. Microservices

Dr.-Ing. Iris Braun
WS 2024/2025

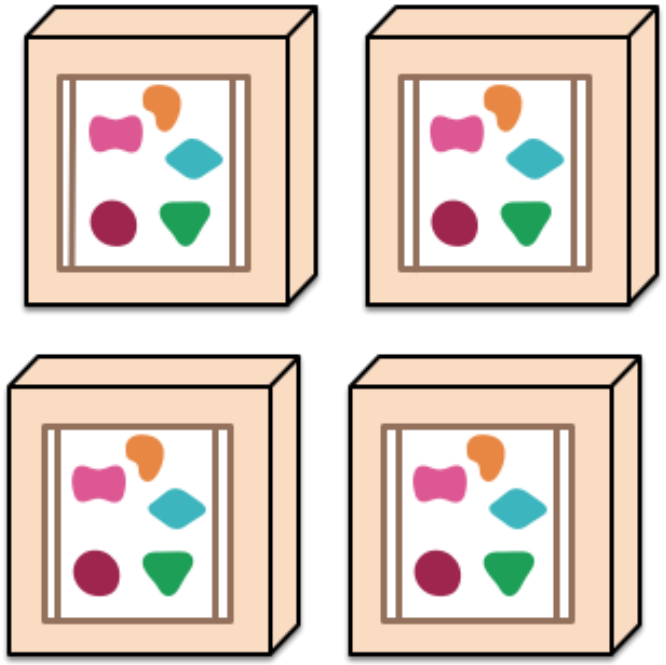
Gliederung

- **Microservices**
  - Architektur
  - Entwicklung
- **DevOps**
- **Cloud Native**
- **Container und Orchestrierung**

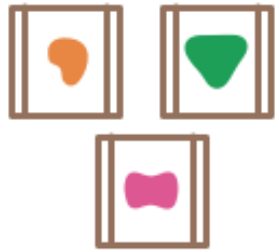
Monolith vs. Microservices

A monolithic application puts all its functionality into a single process...

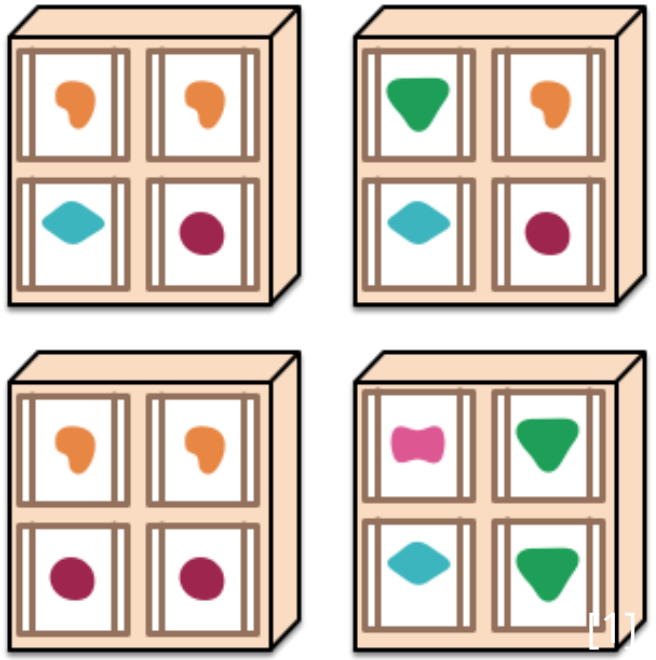
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



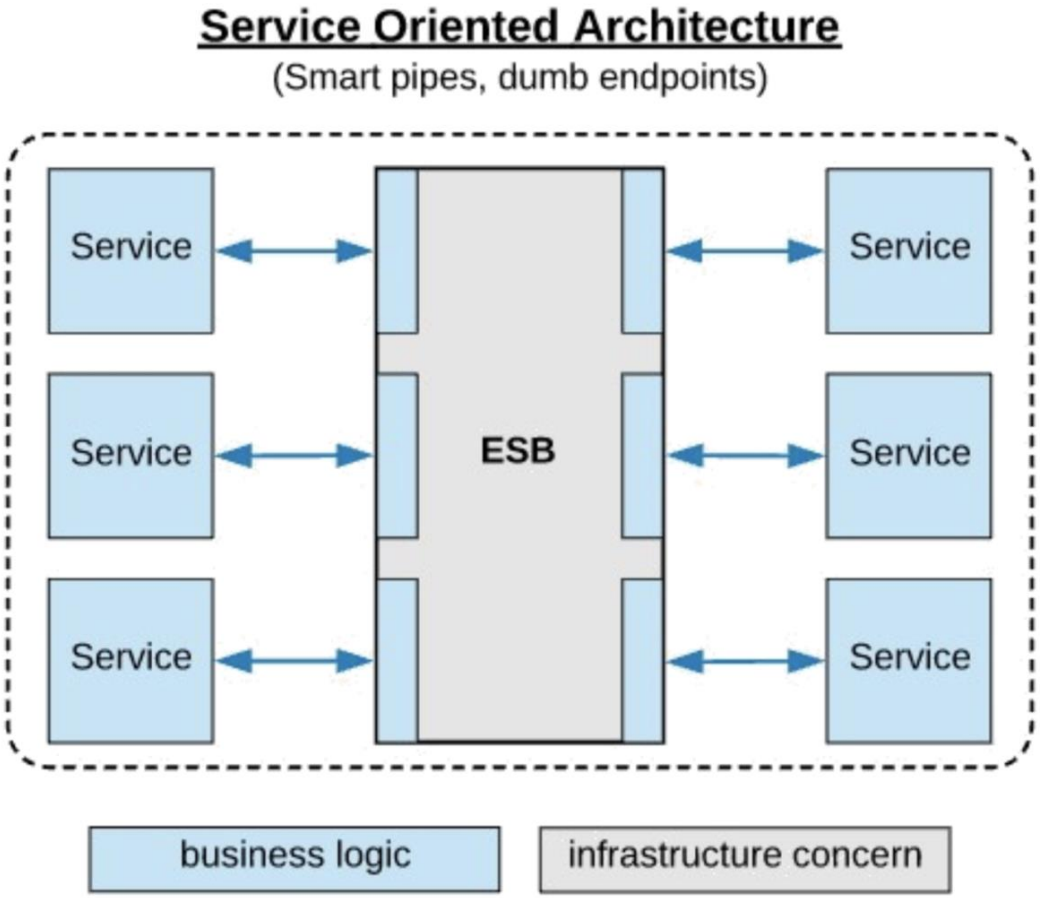
Microservices

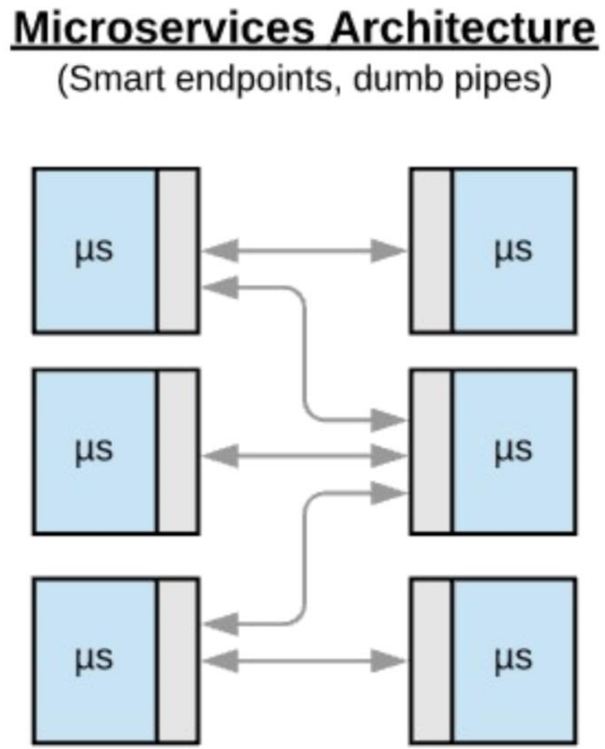
- neues Paradigma der Software-Entwicklung
- agile Softwareentwicklung in vielen Teams erfordert kleine Anwendungspakete mit möglichst wenig Abhängigkeiten
- Teams werden nicht nach Schichten geschnitten, sondern nach Features. Sie entwickeln voneinander unabhängig komplette Features, die von der UI bis zur Datenhaltung alles beinhalten.
- keine monolithische Software, sondern viele unabhängige Services, die dann in Cloud-Infrastruktur verteilt werden können

Microservices - „Shared-Nothing“-Prinzip

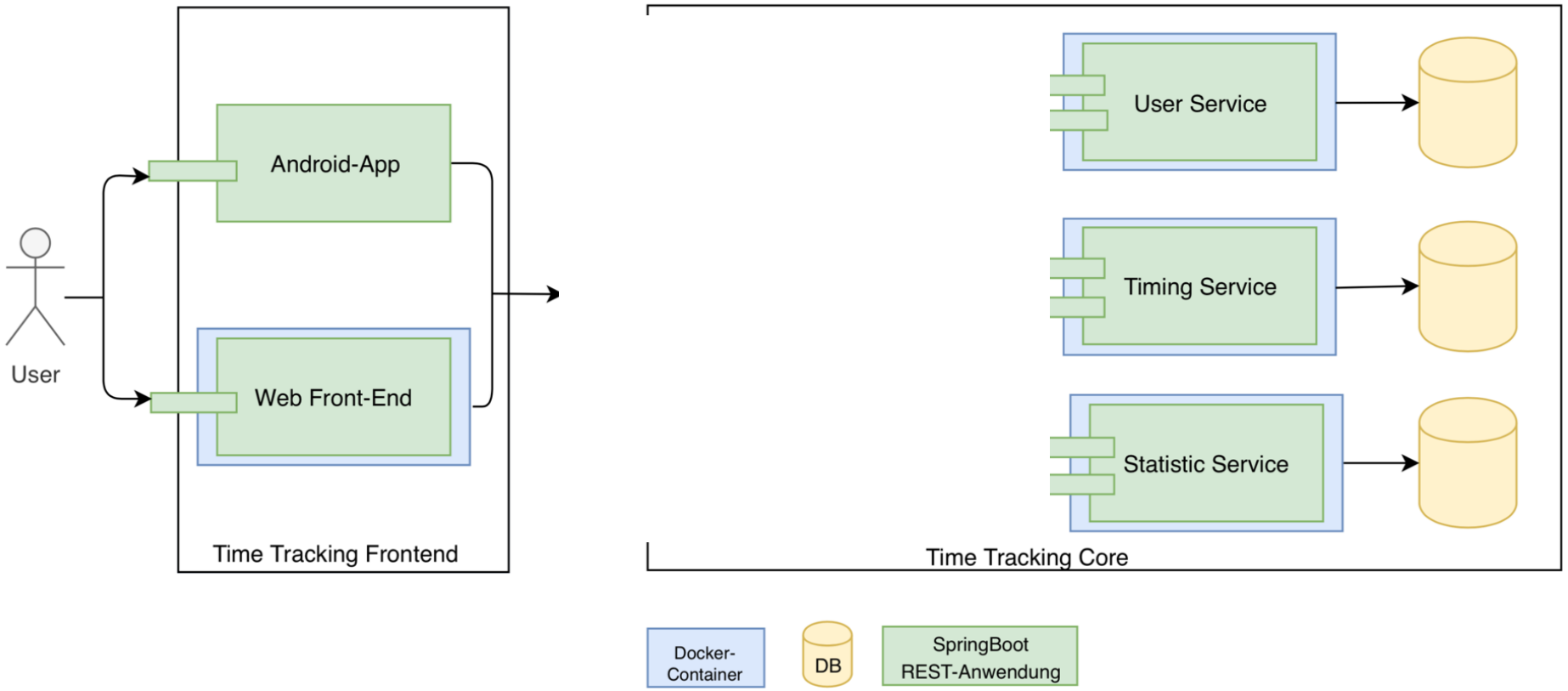
- Microservices teilen sich keinen Code und haben keine gemeinsame Datenhaltung („Shared-Nothing“-Leitbild)
- Kommunikation über ein gemeinsames Protokoll (z.B. über REST, HTTP)
- Definition einer simplen Grundarchitektur, in die die konkreten Dienste verschiedener Teams unabhängig voneinander eingebettet werden können (getrennt deploybare Einheiten, z.B. mit Docker)

klassische SOA vs. Microservices

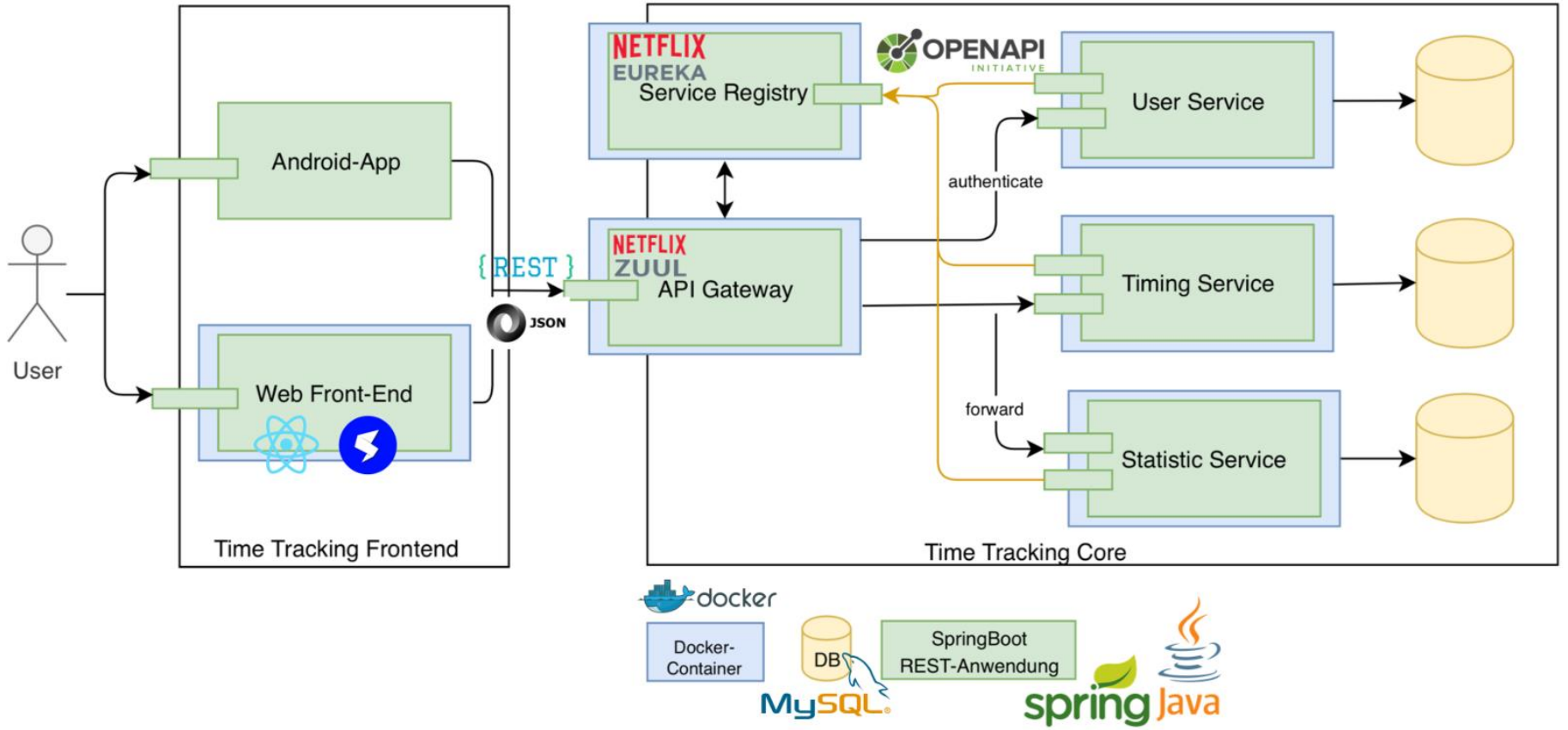




TimeTracker: Beispiel-Architektur einer Microservice Applikation



Technologien zur Umsetzung der Konzepte



Netflix Spring Cloud + Eureka/Zuul

Netflix Spring Cloud

- Technologie Stack für Microservices von Netflix entwickelt
- kann mit Spring verwendet werden, um Microservices in Java zu entwickeln
- Konfiguration via YAML Files

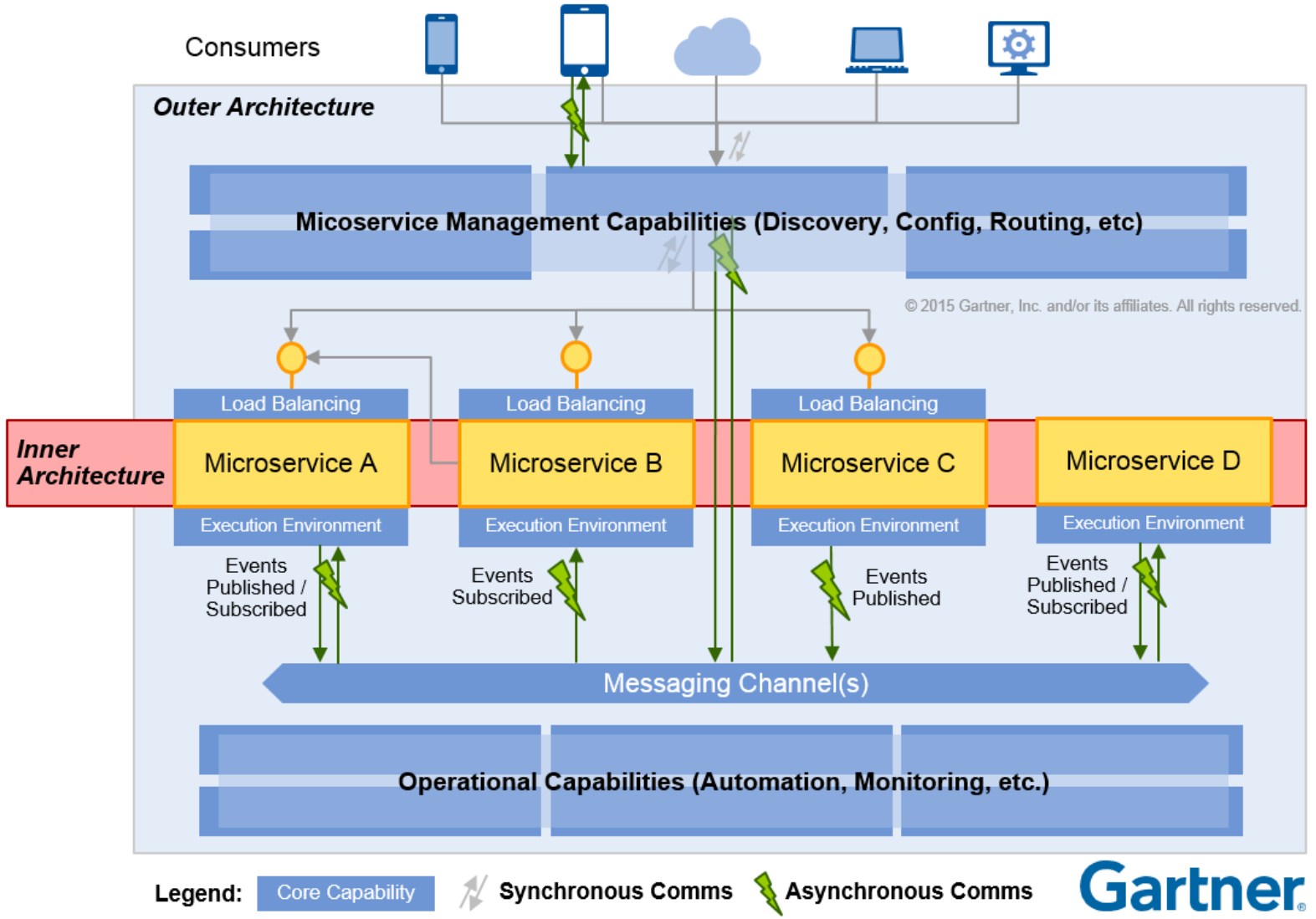
Eureka

- arbeitet als Service Registry
- Dienste benötigen die HTTP-Adresse des Registers, um sich zu registrieren

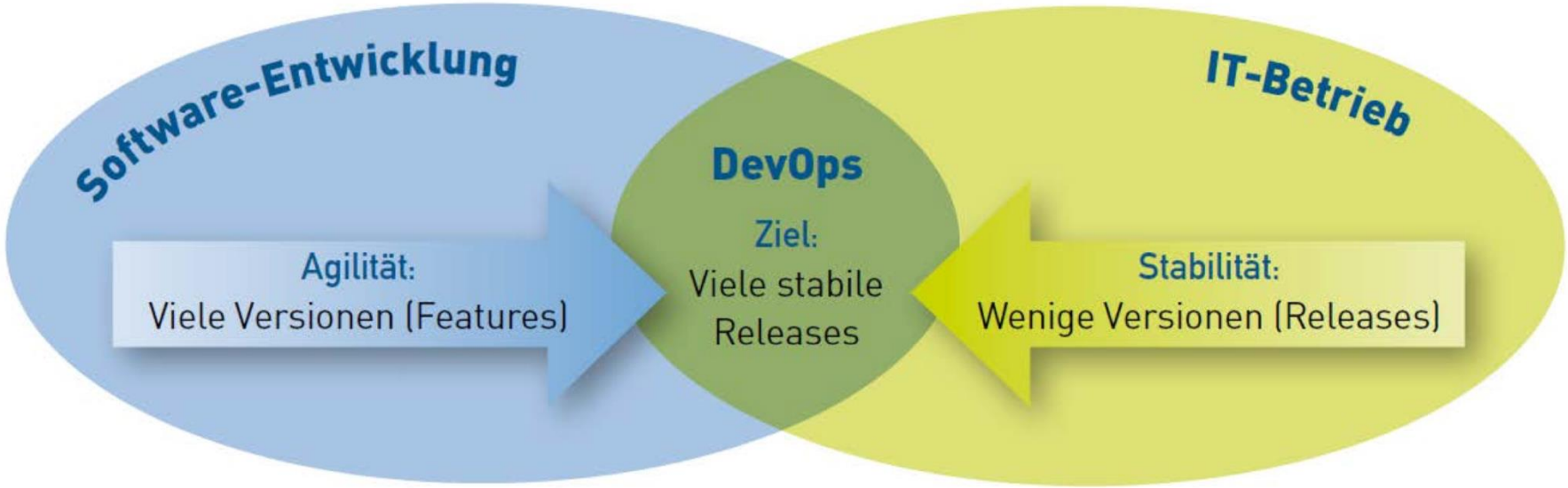
Zuul

- arbeitet als API-Gateway / Reverse Proxy
- übernimmt das Load Balancing zwischen Service-Instanzen
- automatisches Mapping der Pfade zu den Microservices
- fragt bei Registry nach den verfügbaren Dienstens an
- angepasste Routen können in application.yml definiert werden

Microservices Architektur am Bsp. von NetFlix

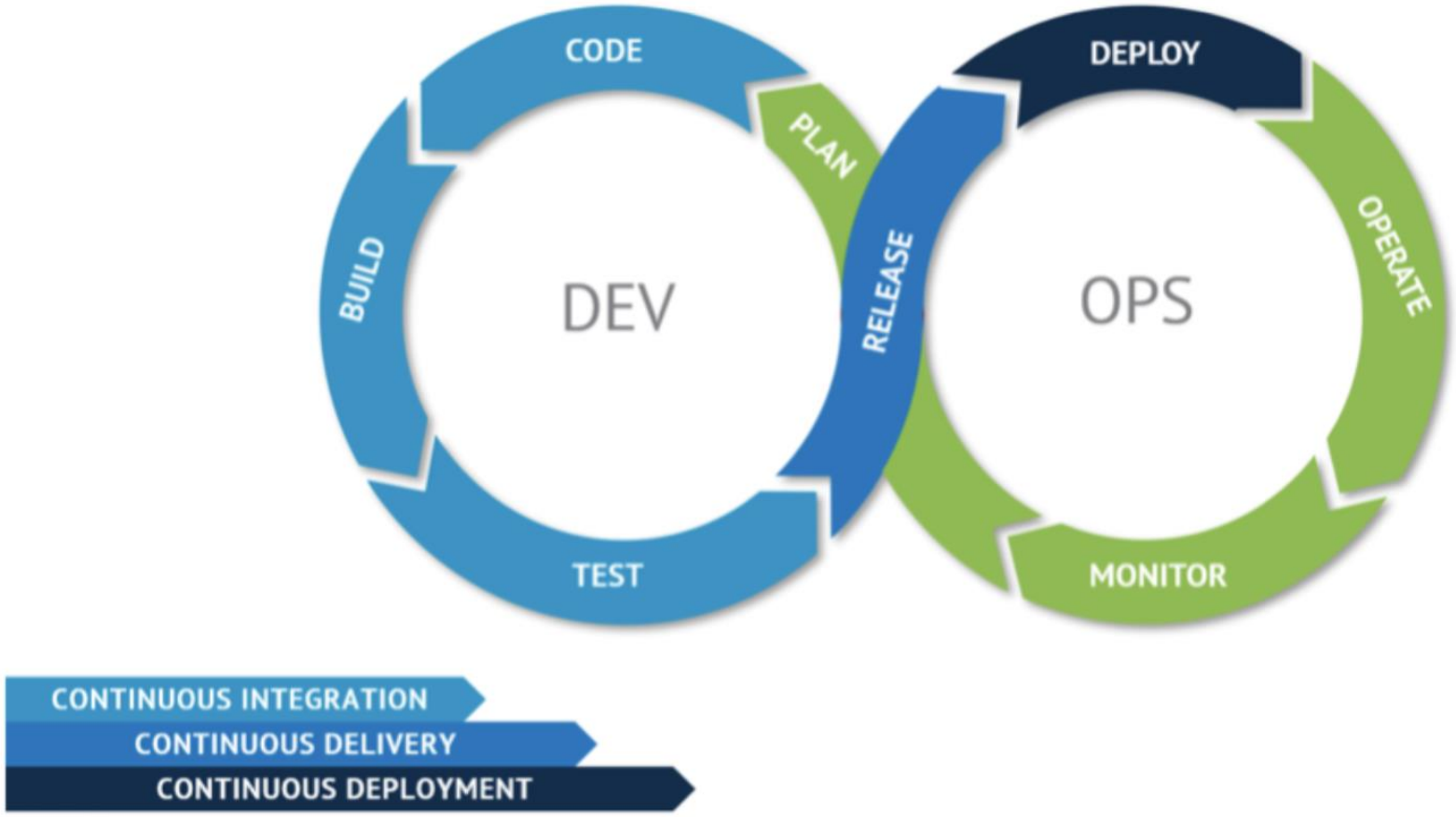


DevOps - Begriffsdefinition



- Mit DevOps sollen die Qualität der Software, die Geschwindigkeit der Entwicklung und der Auslieferung sowie das Miteinander der beteiligten Teams verbessert werden
- Entwicklung einer Kultur der (agilen) Zusammenarbeit zwischen Entwicklungsabteilung (Dev) und Systembetrieb (Ops)
- Qualitätssicherung und Effizienzsteigerung durch Automatisierung von Entwicklungs- und Betriebsaufgaben

DevOps und Continous Deployment

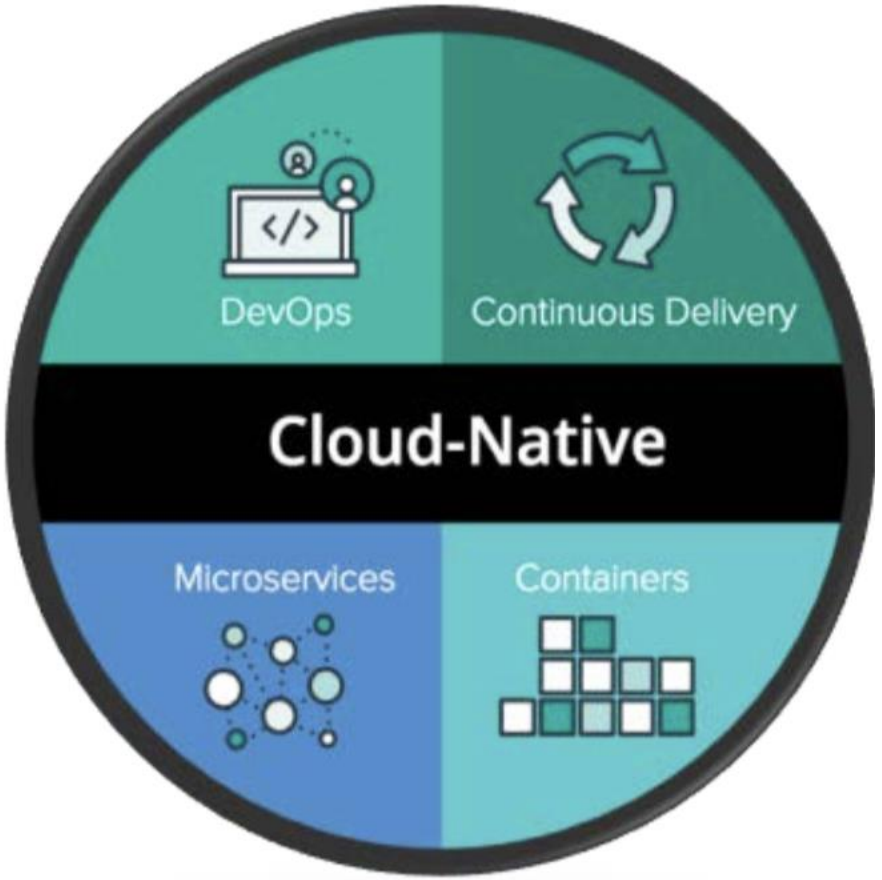


DevOps - Maßnahmen

Automatisierung ist der Schlüssel zum DevOps-Erfolg

- automatisierter Bau von Systemen aus dem Versionsmanagement (z.B. Git)
- automatisierte Code-Analysen und Ausführung von Unit-, Integrations- und Systemtests (test-getriebene Entwicklung)
- automatisierte Installation in Test- und Produktiv-Umgebungen (Deployment)
- Lasttests für Verfügbarkeit und Performance (z.B. mit jMeter)
- kontinuierliches Monitoring, um Fehler, Ausfälle und Performance-Abfall schnell zu bemerken (z.B. Prometheus, Grafana)

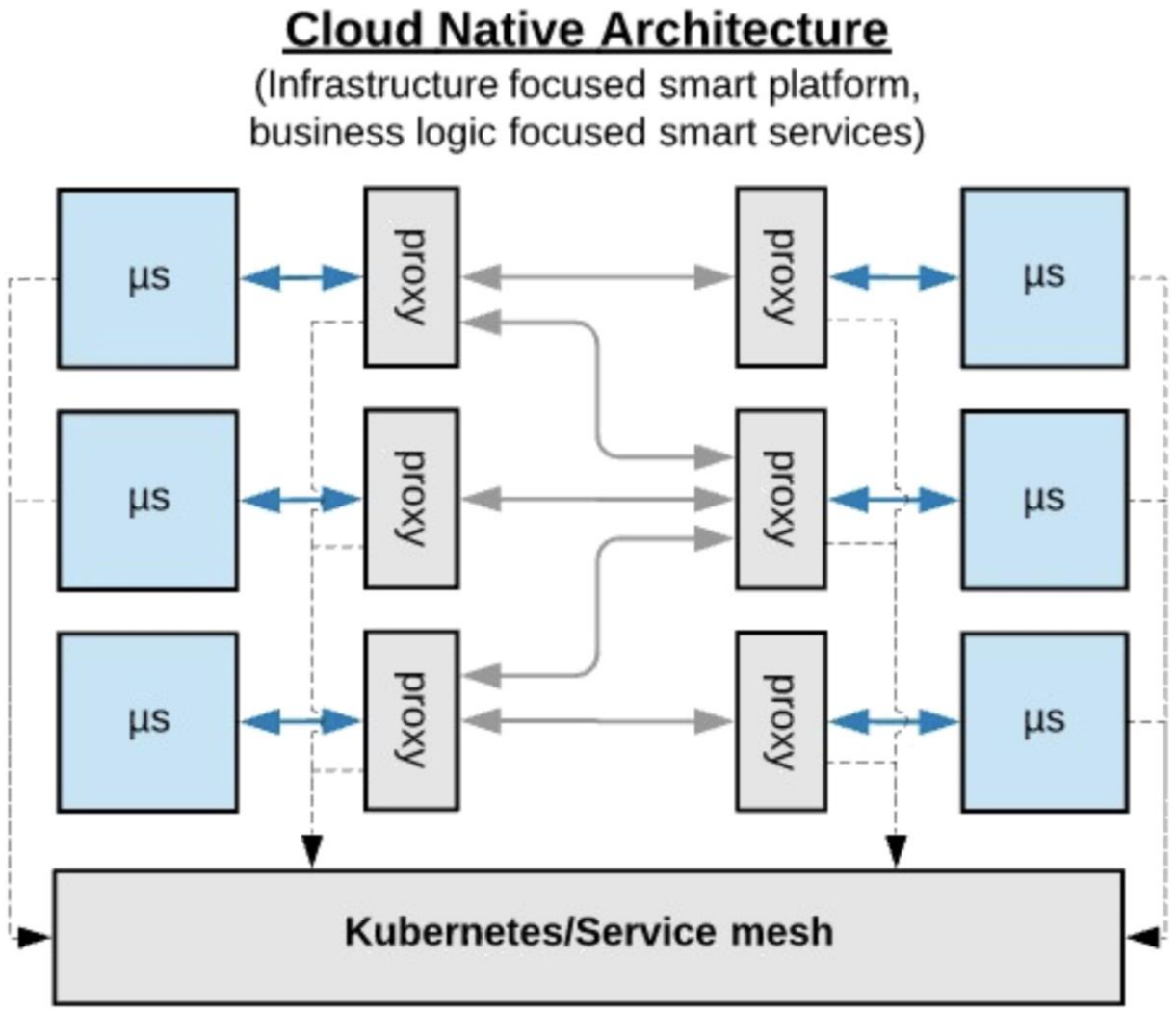
Cloud Native Applications (CNA)



Anwendungen, die speziell für die Cloud-Computing-Architektur entwickelt werden

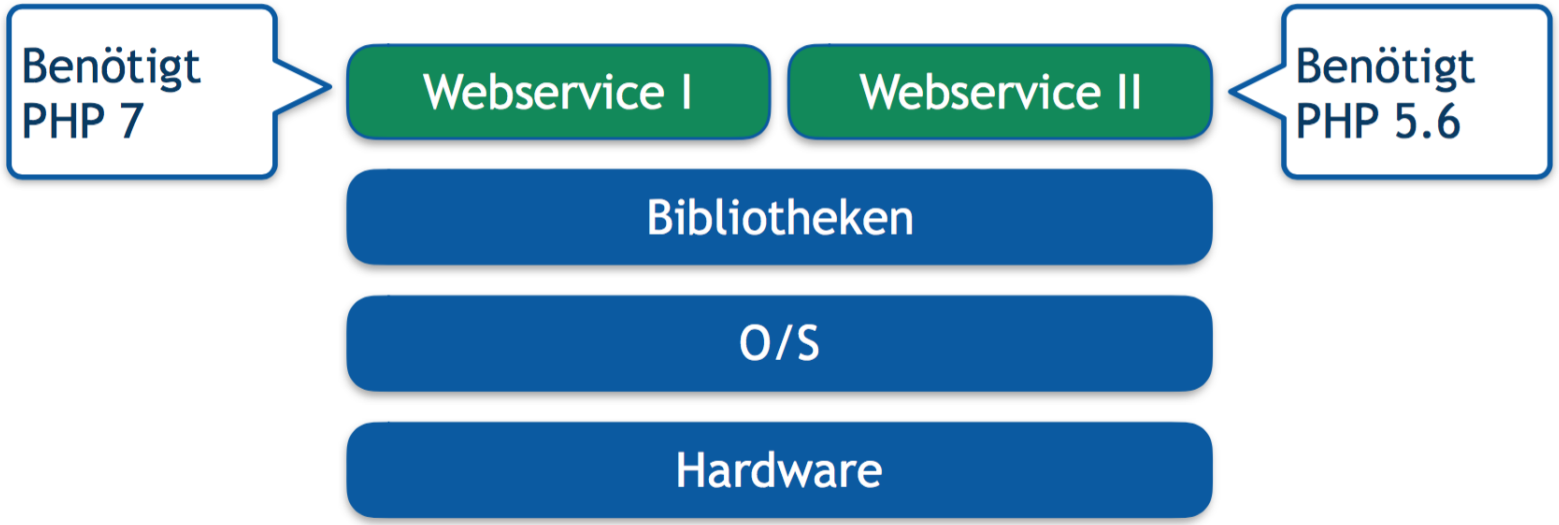
- verteiltes System von Microservices, das in Containern paketiert ist
- Container sind dynamisch auf den verschiedenen Knoten und Servern der Cloud-Umgebung ausführbar
- Testautomatisierung und CI/CD-Pipeline
- Services können bei Bedarf repliziert und verteilt werden. (verteilte Redundanzmechanismen, Service-Orchestrierung)
- Microservices sind lose gekoppelt, arbeiten völlig unabhängig voneinander und stellen nur eine einzige Funktion bereit. (Share-Nothing-Prinzip)

Cloud Native Architecture



Container

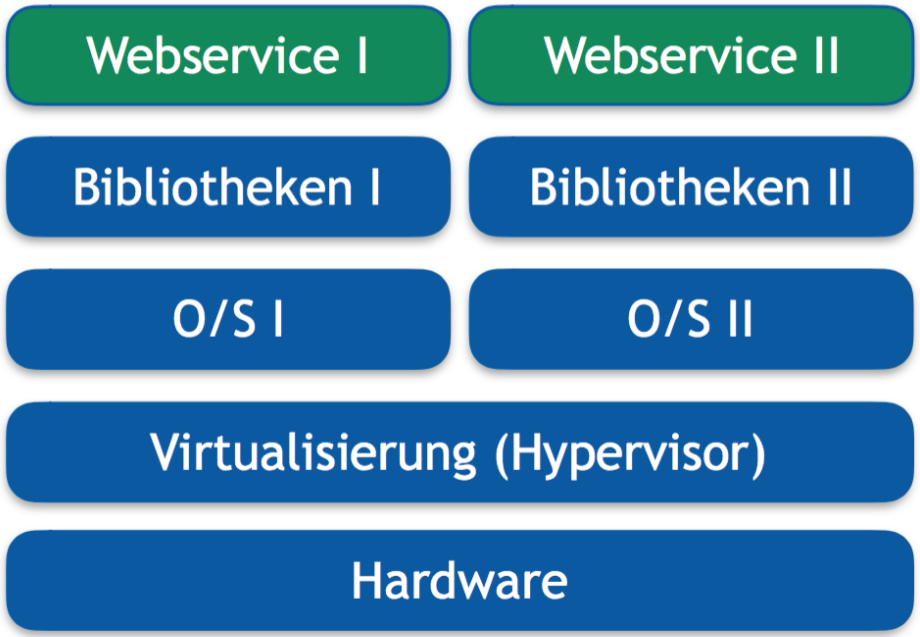
Klassisches Deployment von Webservices



- Kollision, da unterschiedliche Versionen nötig

Container

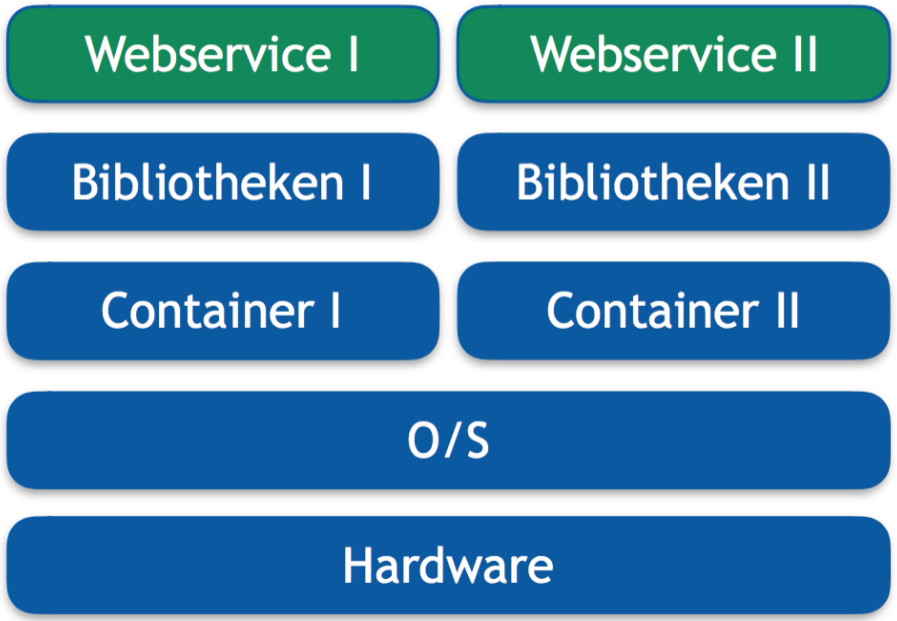
Lösung durch Virtualisierung



- Admin muss mehrere Betriebssysteme aktualisieren, absichern, etc.
- Overhead

Container

Lösung durch Container (z.B. Docker)



- Keine Virtualisierung des Betriebssystems notwendig
- Anwendung und alle benötigten Bibliotheken als Image in Container ausführbar

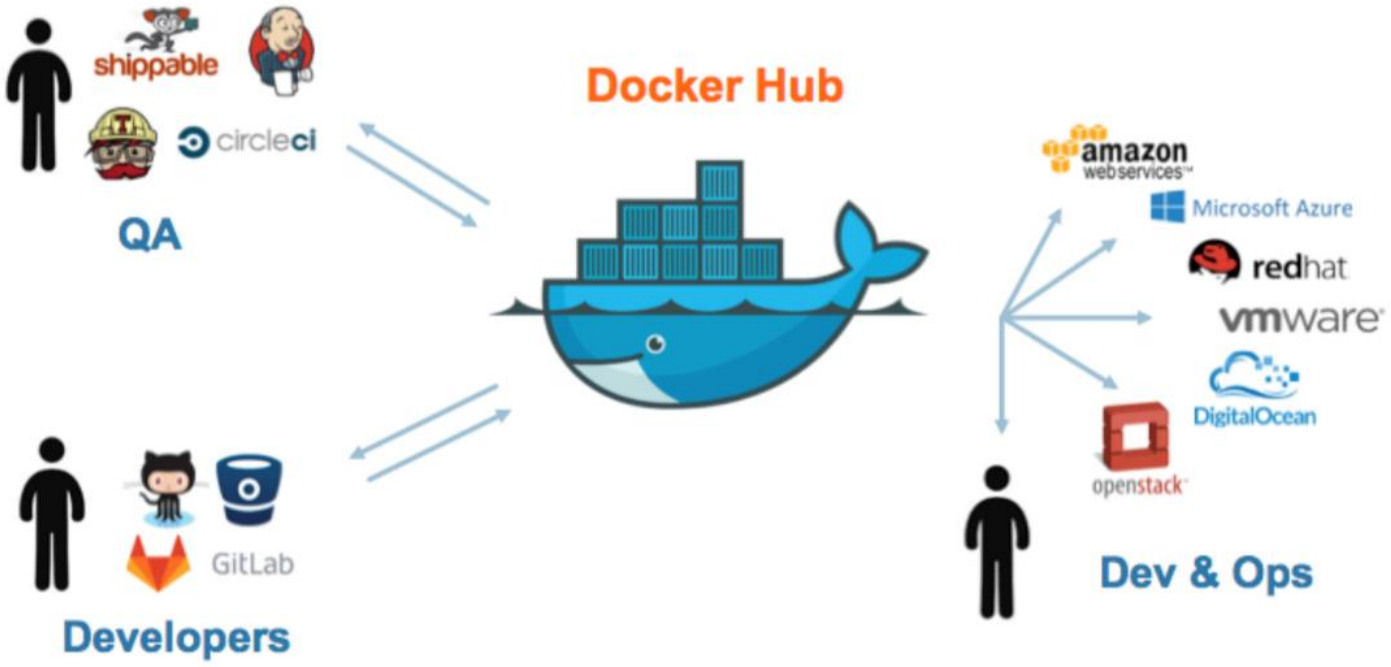
Docker

3-Phasen-Konzept

***Build*:** Erstellung von Container-Images
***Ship*:** Verteilung der Images mit Docker Hub
***Run*:** Ausführung der Images mit der Docker Engine

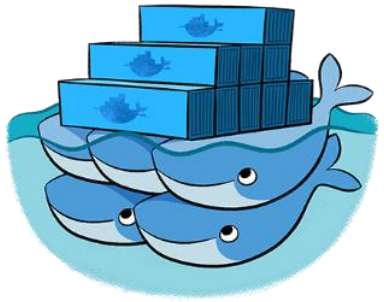
Container (Instanz von Image)

- ohne Virtualisierung ausgeführt
- Nutzung von Linux-Technologien: Namespaces, Control groups, Union file system
- Auf Windows Virtualisierung notwendig



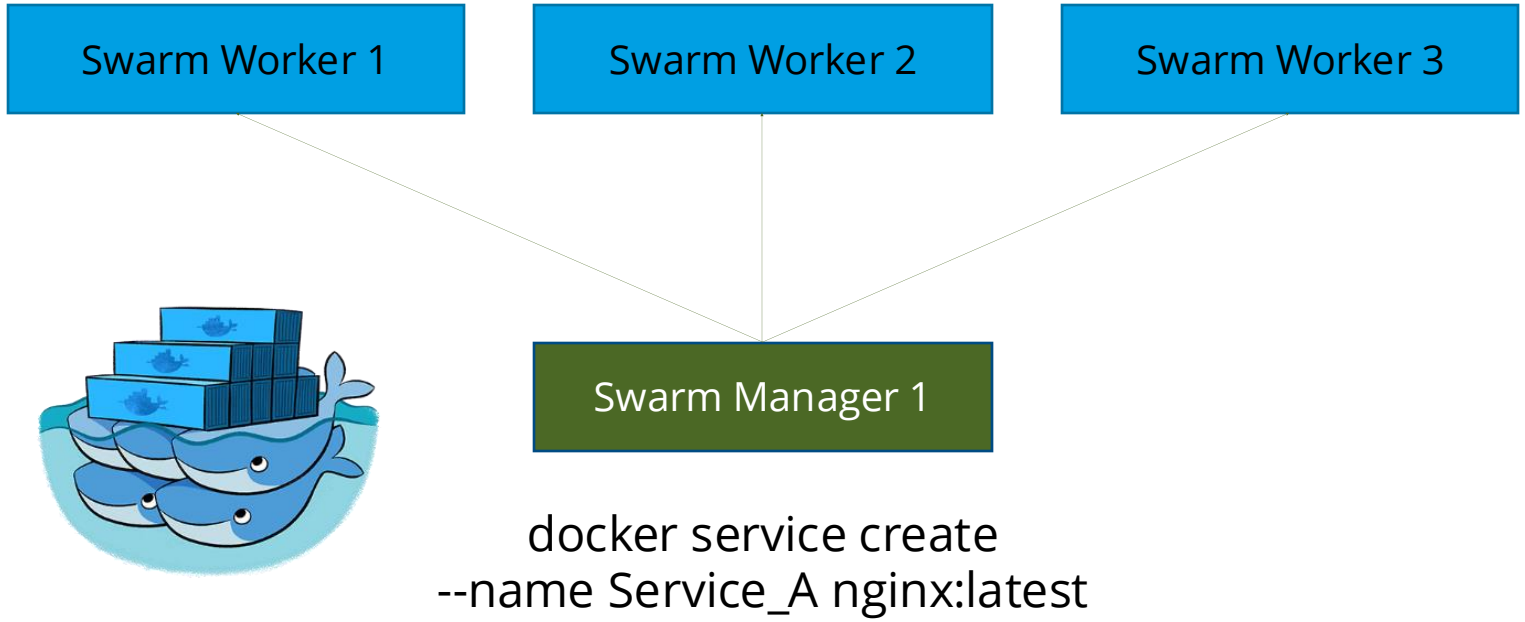
Containerorchestrierung

- Vorteil von Containern: Abgeschlossenheit erleichtert die Skalierung
- Containerorchestrierung ermöglicht von einer zentralen Stelle mehrere Knoten und Container zu verwalten
- Container werden automatisch verteilt
- Zugriff auf Container erfolgt transparent
- Populäre Orchestrationstools:
  - Docker Swarm
  - Kubernetes

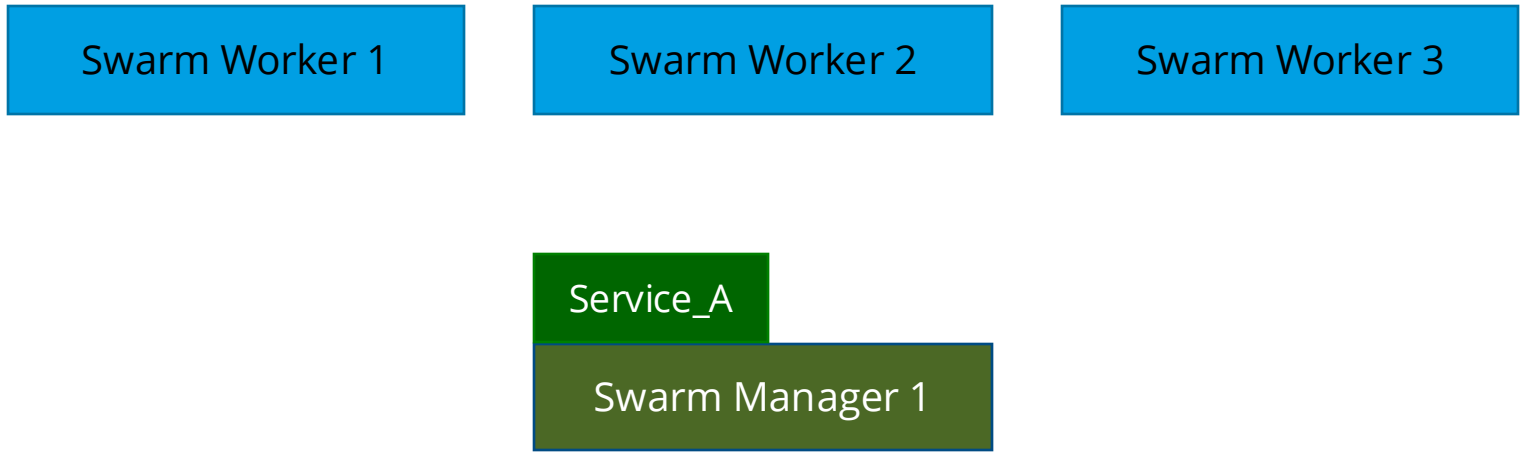


Docker Swarm - Beispiel

- direkt in Docker Engine integriert
- „Schwarm“ aus mehreren Docker-Hosts (Worker + Manager)
- Swarm-Manager stellt den gewünschten Zustand (desired state) her

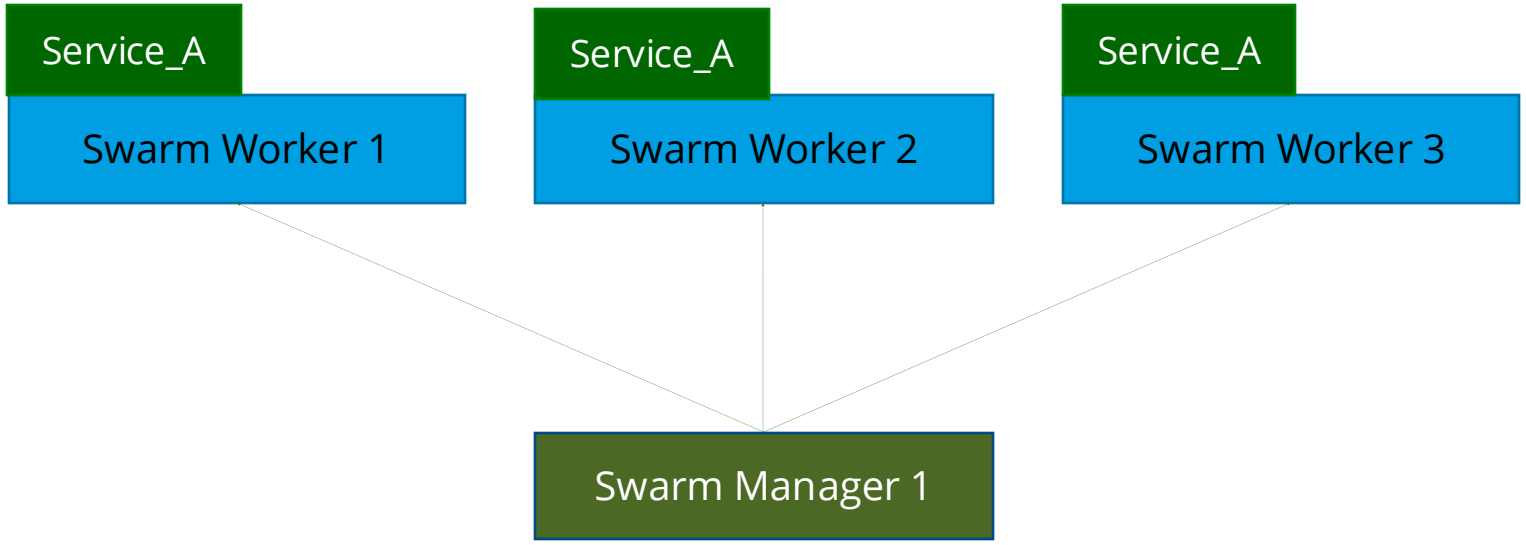


Docker Swarm - Beispiel



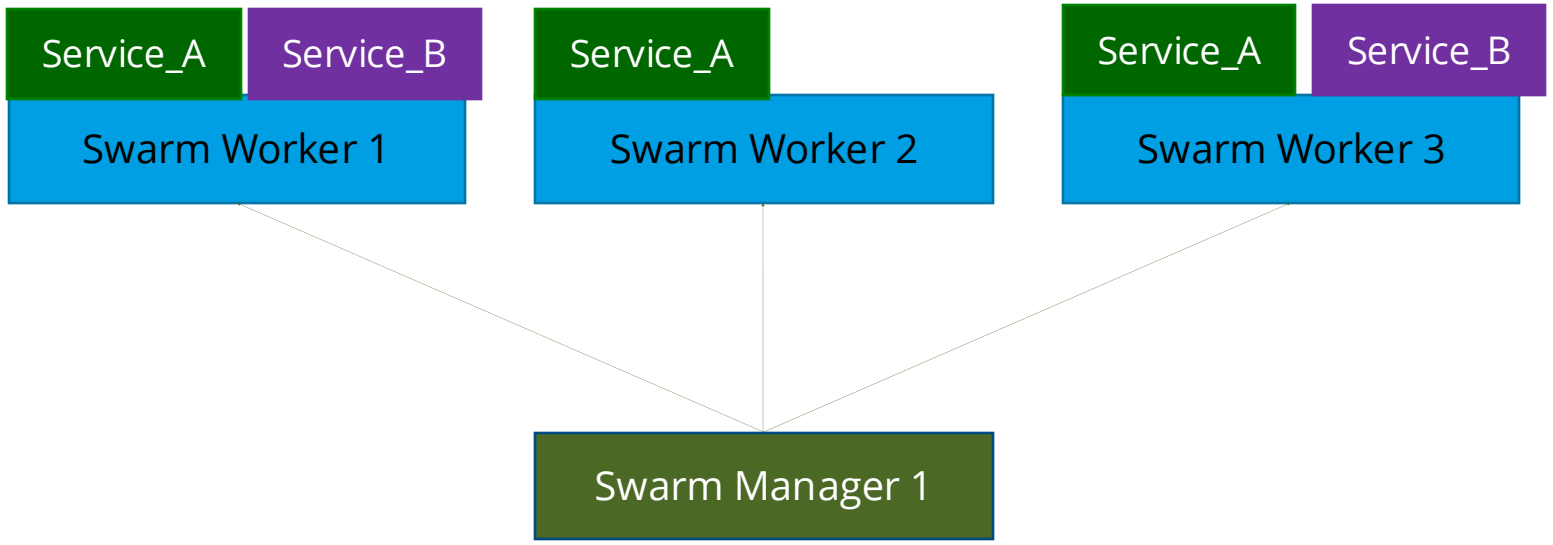
docker service scale Service_A=3

Docker Swarm - Beispiel



docker service create --replicas=2
--constraint node.role==worker --name Service_B redis:latest

Docker Swarm - Beispiel

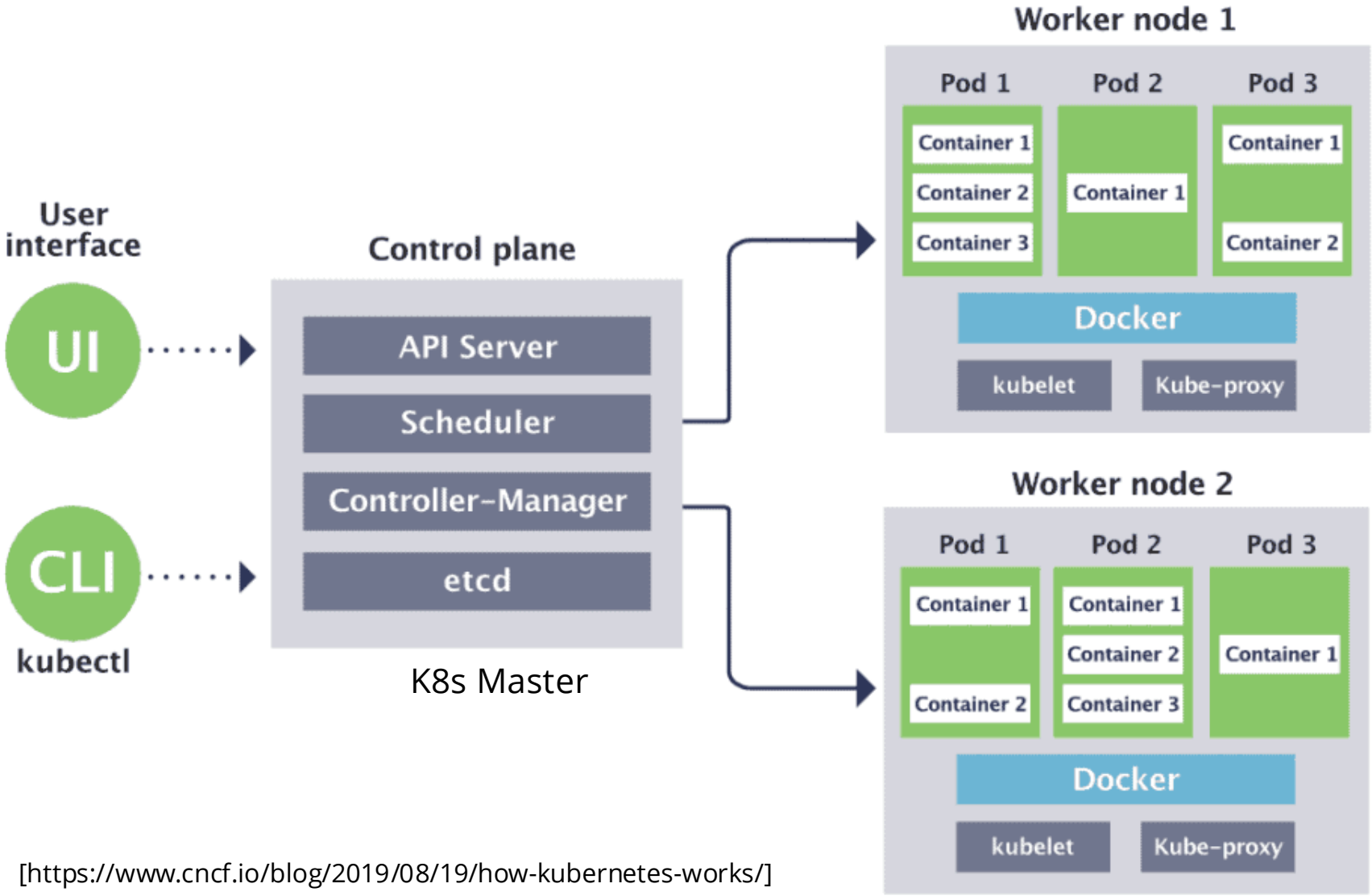


Kubernetes (K8s)

Containerorchestrierungsplattform

- bietet Plattform zur Verwaltung von komplexen Containerinfrastrukturen
- „Kubernetes“ ist griechisch und bedeutet Steuermann
- baut auf 15 Jahre Forschung, Entwicklung von Google auf (Borg)
- seit 2015 Quelloffen und eines der Topprojekte auf Github
- 2015 Cloud Native Computing Foundation (CNCF) gegründet, Projekt der Linux Foundation
- Hoch erweiterbar durch Entwicklung eigener Komponenten (z.B. Scheduler)
- „pets“ (Pflege einzelner Instanzen) vs. „cattle“ (Pflege der Plattform)

Kubernetes (K8s) - Architektur



[https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/]

Referenzen

[1] https://martinfowler.com/articles/microservices/

[2] https://github.com/edisontrent1337/SCC_TimeTracker/blob/master/orga/Documentation/documentation.pdf

[3] http://eprints.uni-kiel.de/29215/1/2015-07-10Architekturen.pdf

[4] https://www.talend.com/blog/2019/04/17/building-a-ci-cd-pipeline-with-talend-and-azure-devops/

[5] https://medium.com/velotio-perspectives/cloud-native-applications-the-why-the-what-the-how-9b2d31897496

[6] https://www.infoq.com/articles/microservices-post-kubernetes/

weitere Empfehlungen:

- 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr https://www.youtube.com/watch?v=LdOe18KhtT4
- Robert C. Martin http://clean-code-developer.de/ https://www.youtube.com/watch?v=ecIWPzGEbFc
- Beyond CI/CD: GitLab's DevOps vision https://about.gitlab.com/2017/10/04/devops-strategy/

Gruppenaufgabe

- Entwerfen Sie für den Anwendungsfall, den Sie im Praktikum entwickeln werden, eine Microservice-Architekur.
- Überlegen Sie, welche Funktionen Sie in den Microservices kapseln möchten.
- Beschreiben Sie die REST-API der einzelnen Services.
- In welcher Programmiersprache möchten Sie die Services implementieren?
- Welche Abhängigkeiten bestehen zwischen den Services?
- Welche Infrastruktur-Komponenten benötigen Sie für den Betrieb?