



# **Vorlesung „Service and Cloud Computing“**

## **3. Microservices**

**Dr.-Ing. Iris Braun**

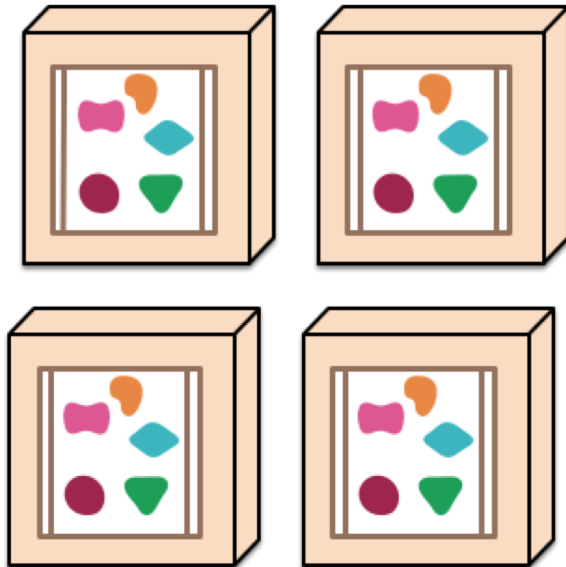
- Microservices
  - Architektur
  - Entwicklung
- DevOps, Continuous Delivery
- Cloud Native

# Monolith vs. Microservices

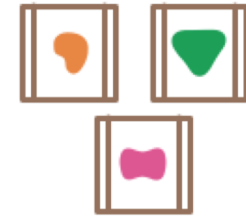
*A monolithic application puts all its functionality into a single process...*



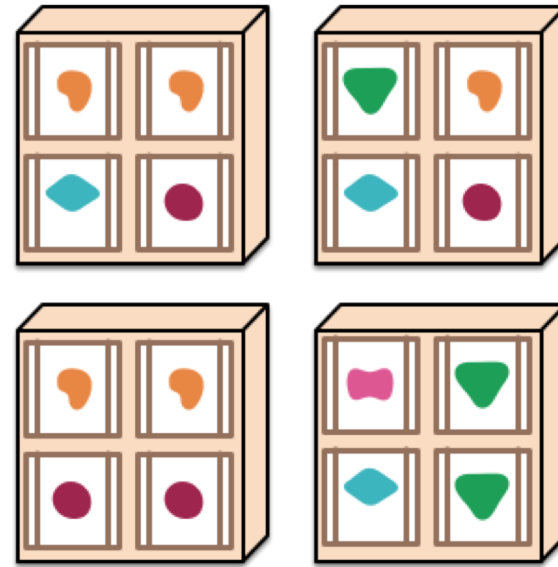
*... and scales by replicating the monolith on multiple servers*



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



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

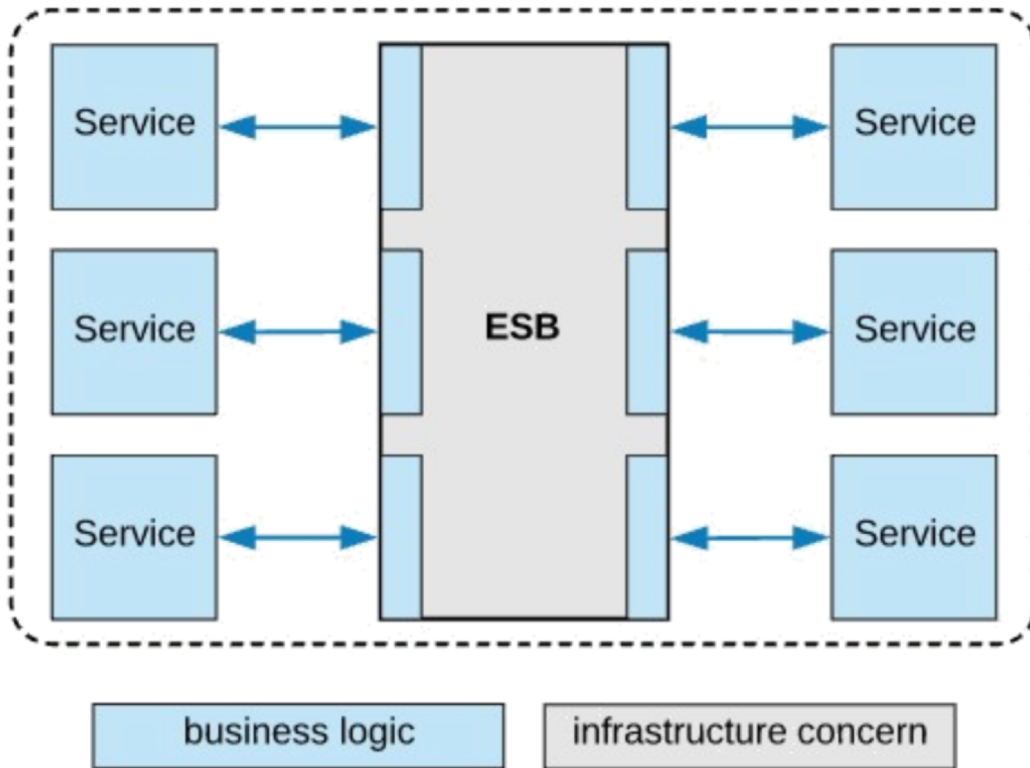


[1]

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

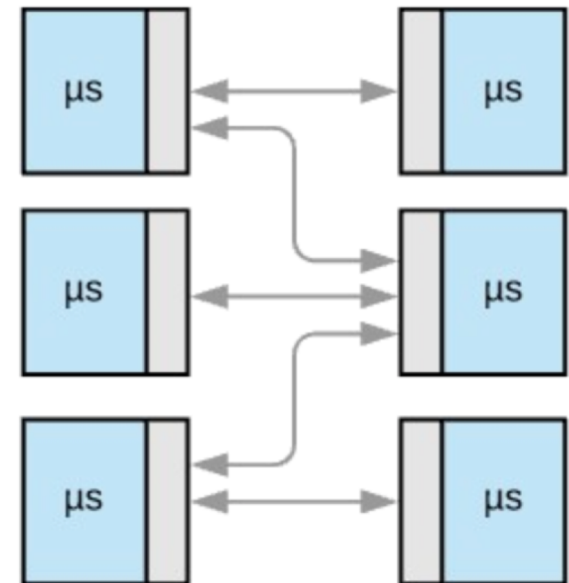
## Service Oriented Architecture

(Smart pipes, dumb endpoints)

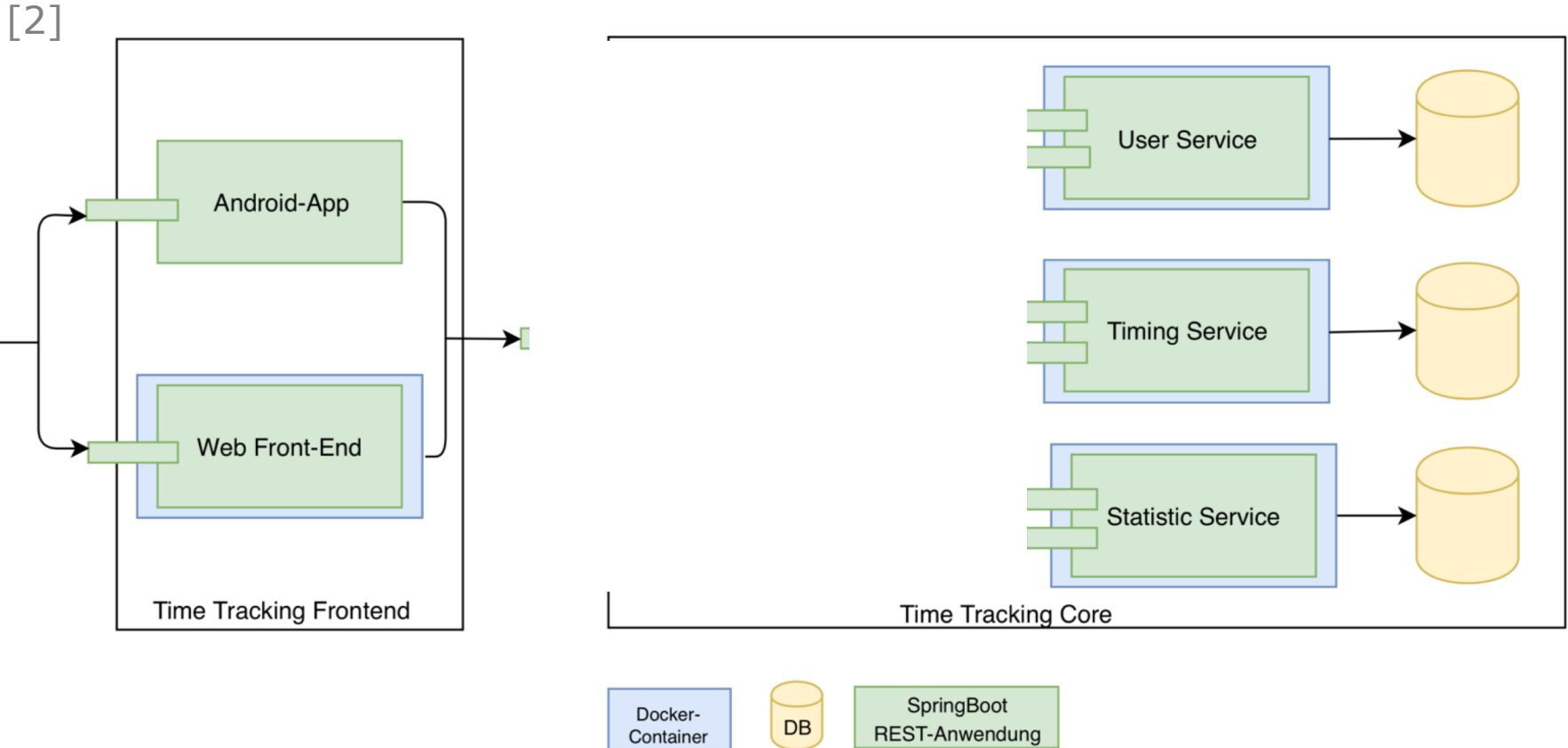


## Microservices Architecture

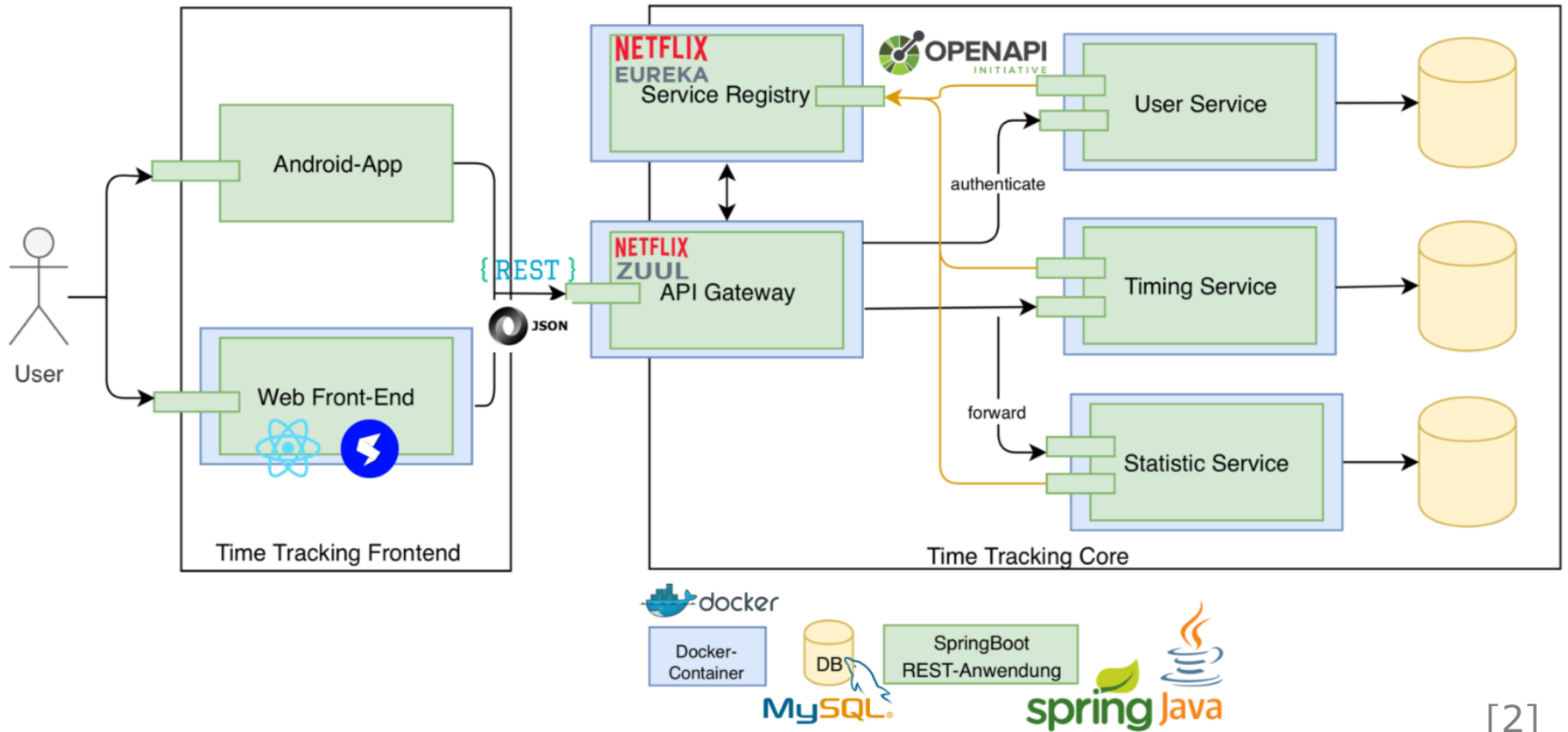
(Smart endpoints, dumb pipes)



[6]



# Mapping of technologies onto concepts



[2]

## Netflix Spring Cloud

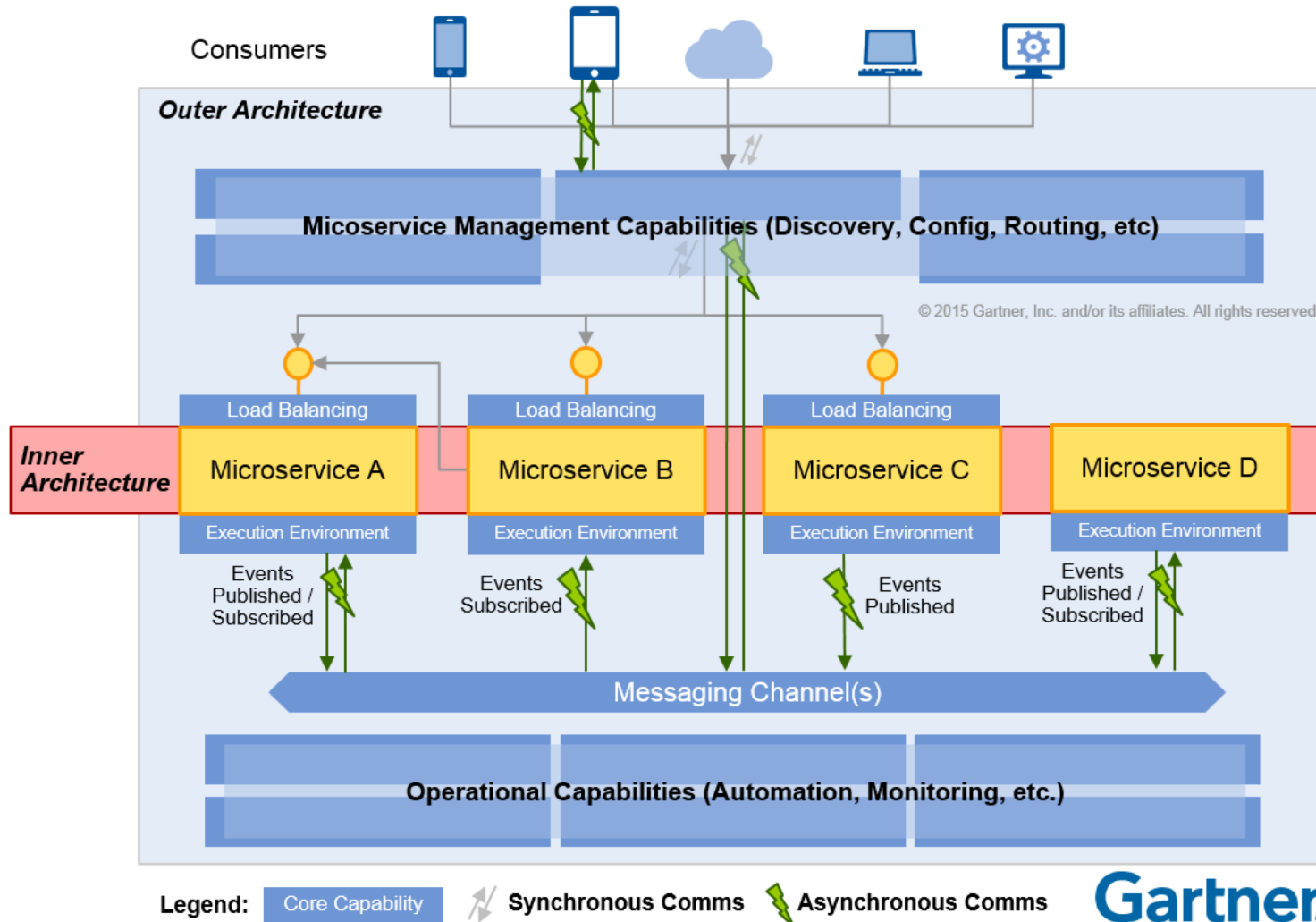
- Technologie Stack für Microservices von Netflix entwickelt
- kann mit **Spring** verwendet werden, um Microservices in Java zu entwickeln
- Konfiguration via YAML Files

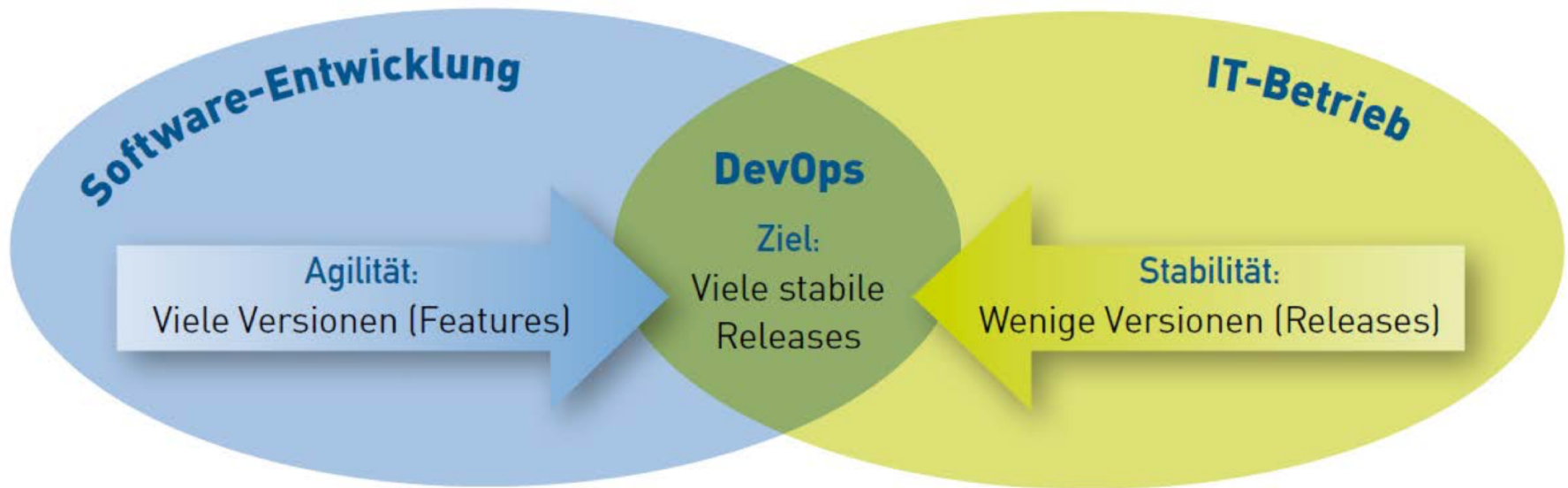
## Eureka

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

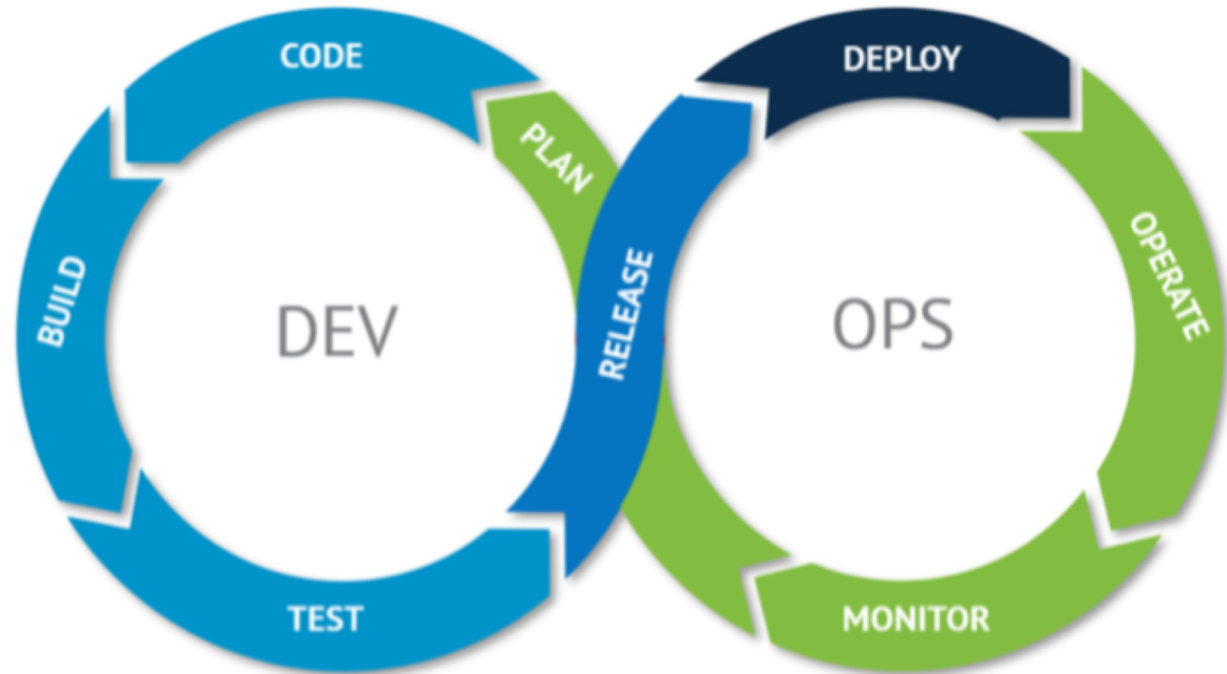
## Zuul

- 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 Diensten an
- **angepasste Routen** können in application.yml definiert werden





- 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 [3]

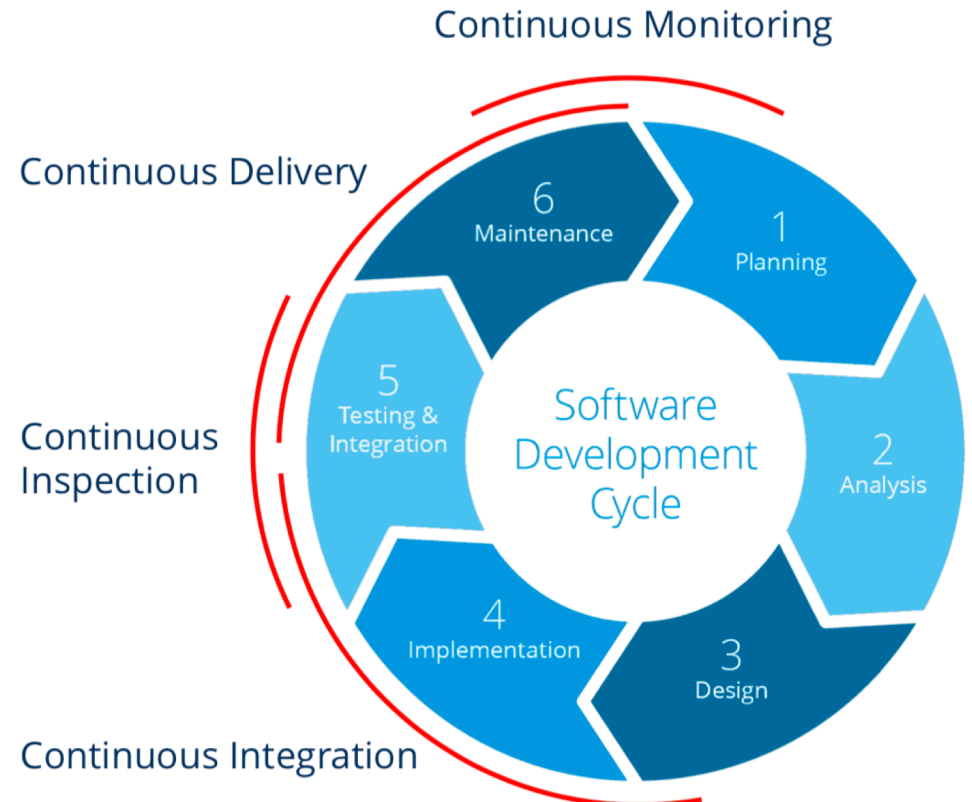


[4]

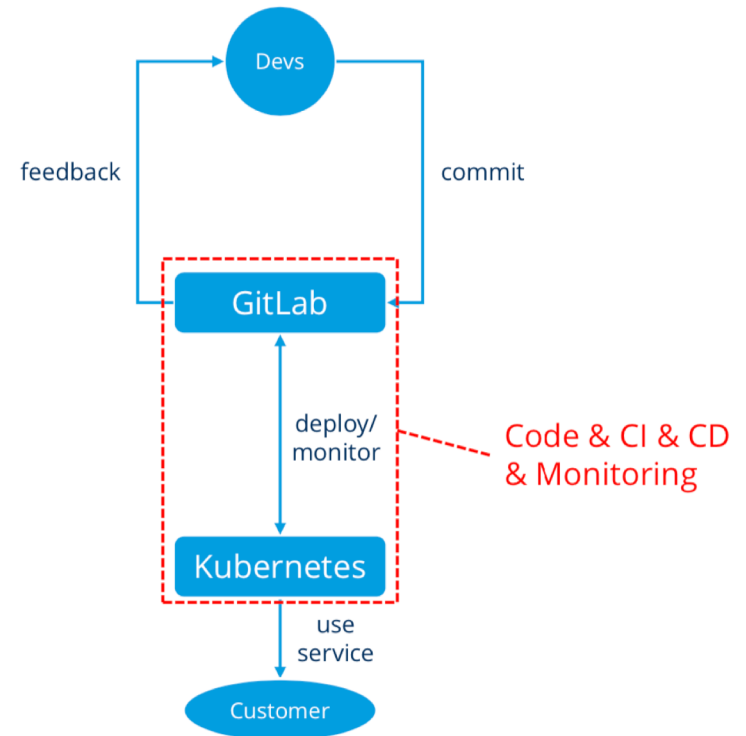
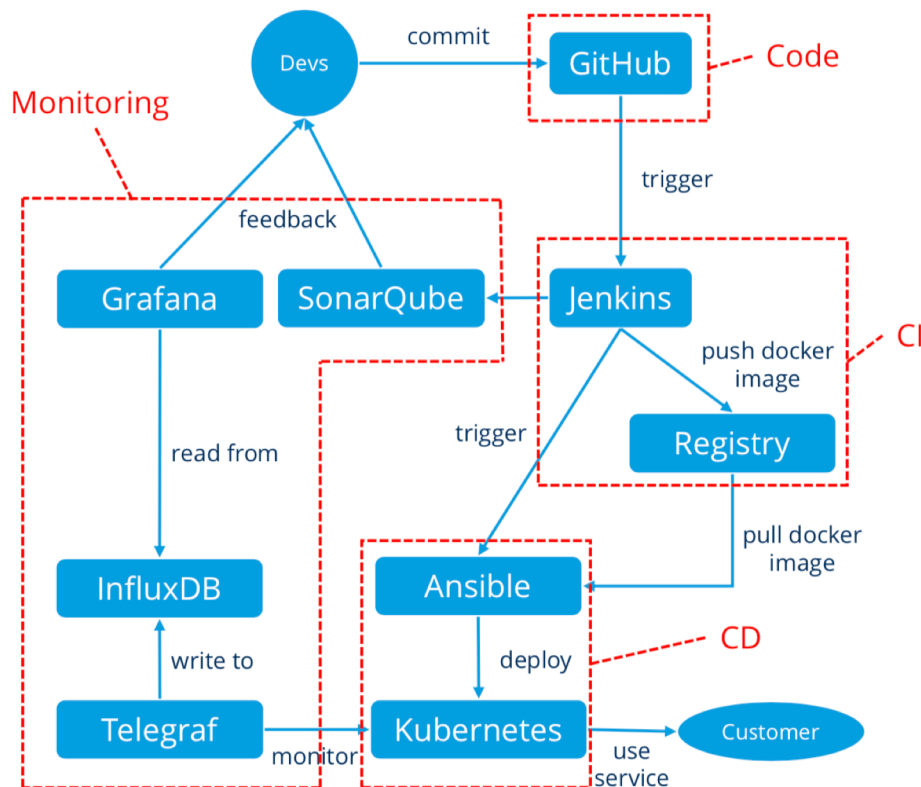
## Automatisierung ist der Schlüssel zum DevOps-Erfolg

- 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
- Lasttests für Verfügbarkeit und Performance (z.B. mit jMeter)
- kontinuierliches Monitoring, um Fehler, Ausfälle und Performance-Abfall schnell zu bemerken (z.B. Prometheus, Grafana)

- Continuous Integration – CI:
  - Automatisches Bauen von Artefakten
  - Automatisches ausführen von Unit- und Integrationstests
- Continuous Delivery – CD:
  - Automatisches deployen getesteter Artefakte
- Continuous Inspection:
  - Automatisches analysieren des Codes auf Vulnerabilities und Codequalität
- Continuous Monitoring:
  - Automatisches sammeln von Metriken



<p><b>Application Lifecycle Mgmt.</b></p>	<p><b>SCM/VCS</b></p>	<p><b>Testing</b></p>	<p><b>Deployment</b></p>	<p><b>Cloud / IaaS / PaaS</b></p>
<p><b>Communication &amp; ChatOps</b></p>	<p><b>CI</b></p>	<p><b>Testing (continued)</b></p>	<p><b>Config Mgmt./Provisioning</b></p>	<p><b>Orchestration &amp; Scheduling</b></p>
<p><b>Knowledge Sharing</b></p>	<p><b>Build</b></p>	<p><b>Testing (continued)</b></p>	<p><b>Artefact Management</b></p>	<p><b>BI / Monitoring / Logging</b></p>
<p><b>Database Management</b></p>	<p><b>Database Management (continued)</b></p>	<p><b>Testing (continued)</b></p>	<p><b>Artefact Management (continued)</b></p>	<p><b>BI / Monitoring / Logging (continued)</b></p>



## Multitoolpipeline

Benötigte Dateien – IaC:

- Dockerfile
- Jenkinsfile
- Ansible Inventory
- Ansible Playbook
- Allerlei Konfigurationen

Vorteile:

- Quasi endlos erweiterbar
- Teilweise großer Community support

Nachteile:

- Hoher Wartungsaufwand
- Viel Einarbeitung

## GitLab Pipeline

Benötigte Dateien - IaC:

Mit AutoDevOps:

- nichts

Ohne AutoDevOps

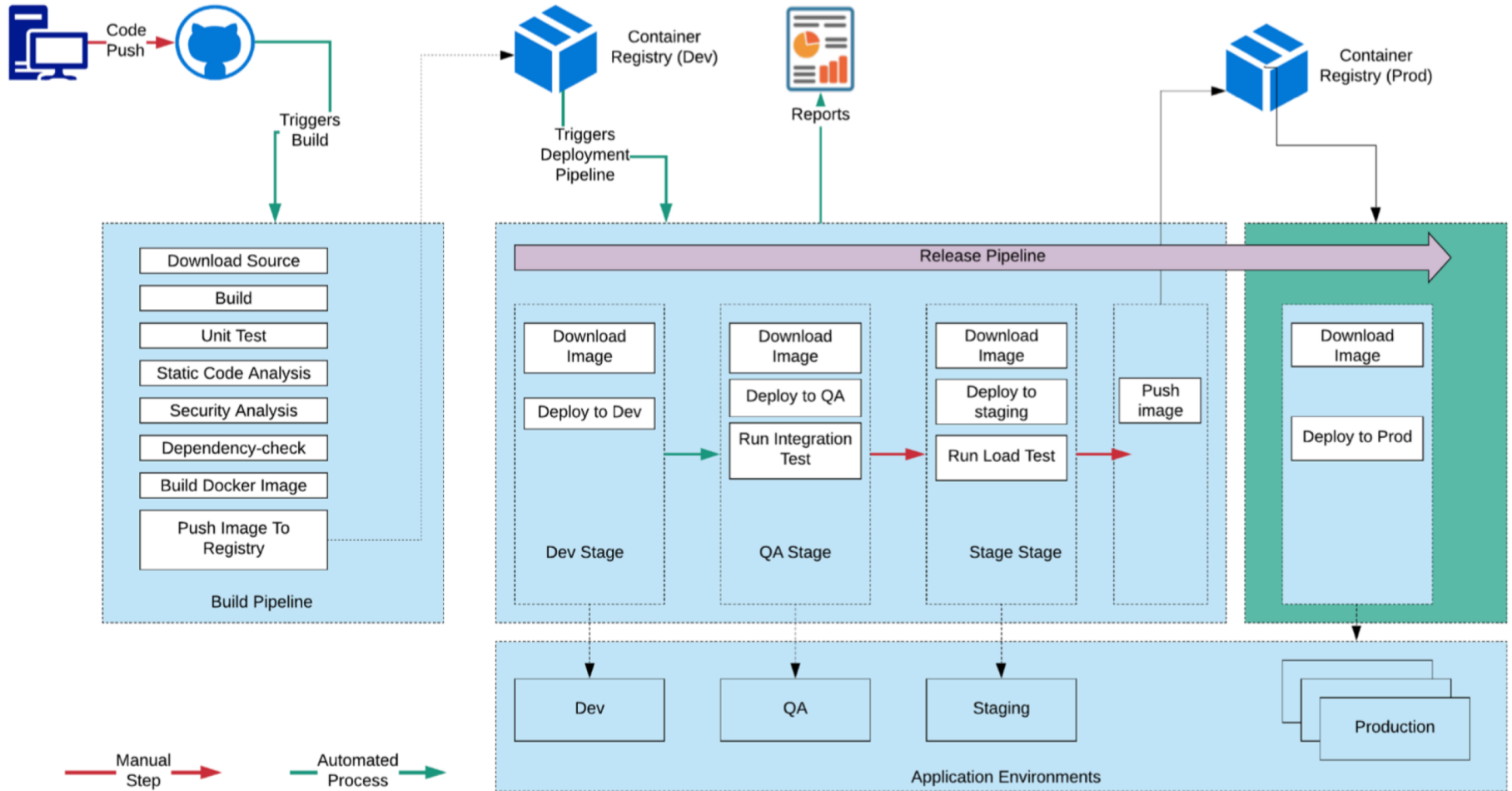
- gitlab-ci.yml

Vorteile:

- Wenig Einarbeitung
- Kaum Wartungsaufwand bei PaaS

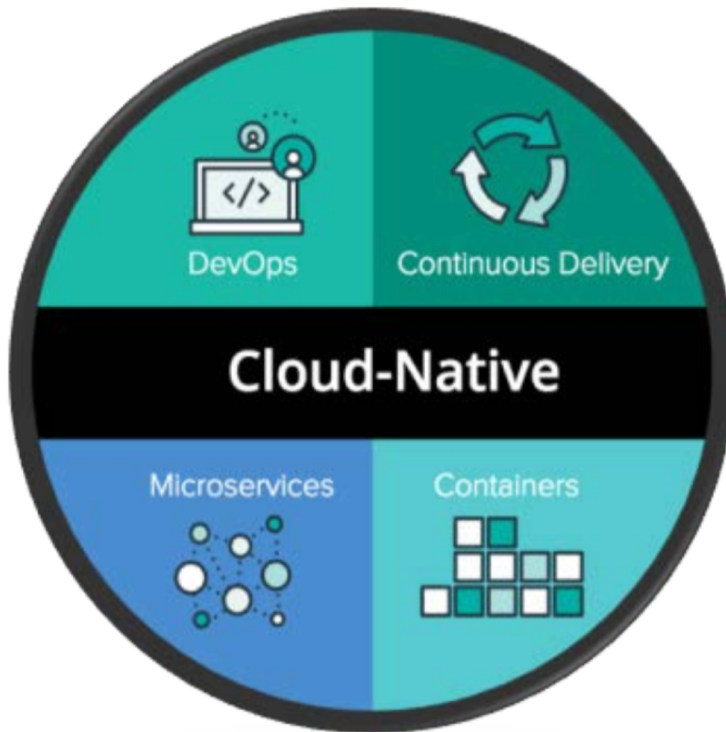
Nachteile:

- Vendor-Lock-In
- Beschränkte Features



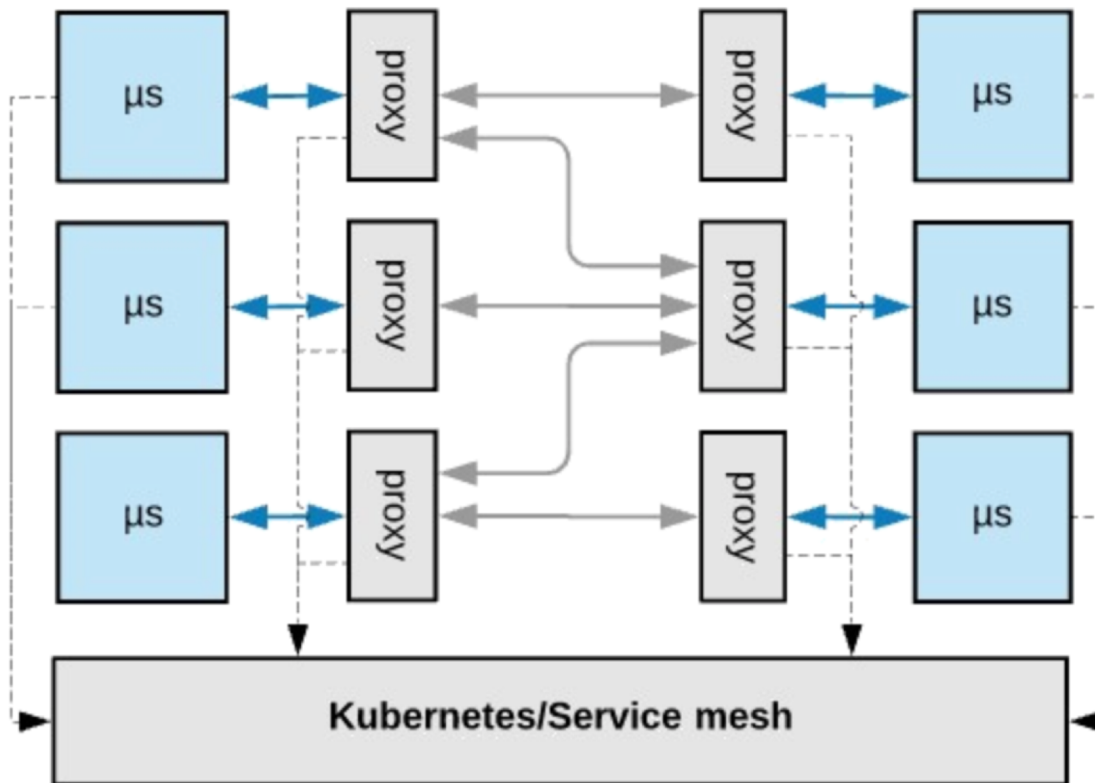
Anwendungen, die speziell für die Cloud-Computing-Architektur entwickelt werden

- verteiltes System von Microservices, das in Containern paketiert ist.
- Die einzelnen Container sind dynamisch auf den verschiedenen Knoten und Servern der Cloud-Umgebung ausführbar.
- Testautomatisierung und CI-Pipeline
- Die Services können bei Bedarf repliziert und verteilt werden. (verteilte Redundanzmechanismen, Service-Orchestrierung)
- Die Microservices sind lose gekoppelt. Oft arbeiten sie völlig unabhängig voneinander und stellen nur eine einzige Funktion bereit.



[5]

**Cloud Native Architecture**  
(Infrastructure focused smart platform,  
business logic focused smart services)



[6]

- [1] <https://martinfowler.com/articles/microservices/>
- [2] [https://github.com/edison-trent1337/SCC\\_TimeTracker/blob/master/orga/Documentation/documentation.pdf](https://github.com/edison-trent1337/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/>

- Entwerfen Sie für den Anwendungsfall, den Sie im Praktikum entwickeln werden, eine Microservice-Architektur.
- 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?