

Microservices

Verteilte Softwaresysteme

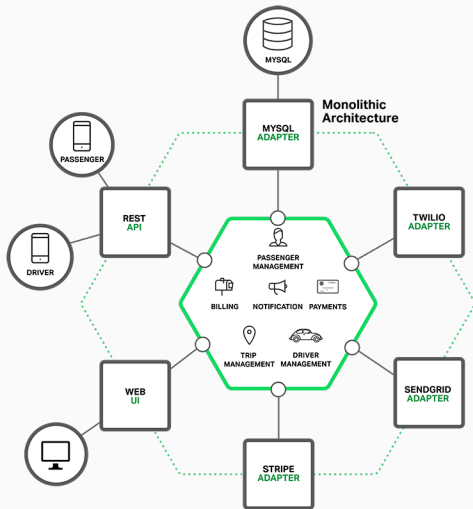
Prof. Dr. Oliver Braun

Letzte Änderung: 14.01.2019 22:07

Was sind Microservices?

- Softwarearchitektur-Pattern das
 - monolithische Anwendungen in kleine Dienste aufteilt
 - die über sprachunabhängige Protokolle kommunizieren
- *Loosely coupled service oriented architecture with a bounded context (Adrian Cockcroft, Netflix)*
- *An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (Martin Fowler)*
- Idee ist nicht neu
 - Serviceoriented Architecture
 - Unix Prozesse und Pipes

Beispiel: Taxi-Anwendung als Monolith



Quelle: <https://www.nginx.com/blog/introduction-to-microservices/>

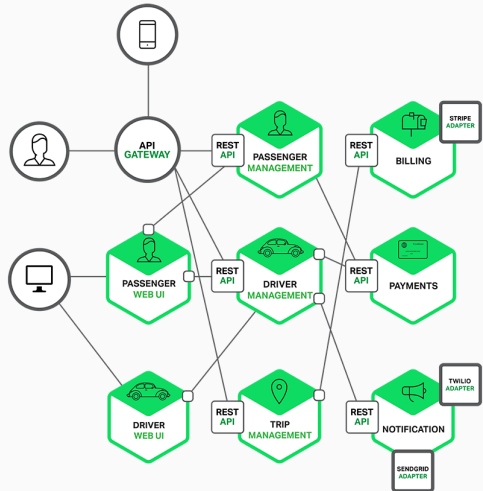
- intern sicherlich modular aufgebaut
- als Monolith deployed
- übliche Architektur
- einfach zu entwickeln
- einfach zu testen
- einfach zu deployen

Probleme von Monolithen

- erfolgreiche Anwendungen wachsen schnell und werden unübersichtlich groß
- damit wird auch die Organisation der Entwicklung extrem schwierig
- ein Entwickler kann gar nicht mehr die gesamte Anwendung überblicken und verstehen
- der simple Start einer Applikation (und damit die tote Zeit bei der Entwicklung) dauert immer länger
- Continuous Deployment ist quasi unmöglich
 - heute ist es teilweise üblich mehrfach pro Tag Änderungen ins Produktivsystem zu pushen
- Monolithen können nur als Ganzes skaliert werden (wenn überhaupt)
- weil alle Module in einem Prozess laufen, kann ein Bug die gesamte Applikation zum Absturz bringen
- Frameworks oder Programmiersprachen können kaum gewechselt werden

Beispiel: Taxi-Anwendung mit Microservices

- Lösung für viele der genannten Probleme sind Microservices
- den Schritt dorthin haben schon viele große Firmen vollzogen:
 - Amazon
 - ebay
 - Netflix
 - ...



Quelle: <https://www.nginx.com/blog/introduction-to-microservices/>

- **loosly coupled**
 - ein Microservice kann völlig unabhängig von allen anderen geupdated werden
 - kein anderer Service muss geändert werden
 - ist das nicht der Fall, so sind es keine Microservices
- **bounded context**
 - ein Microservice muss *self-contained* sein
 - d.h. er kann verstanden werden ohne etwas über die Services zu wissen, die er nutzt, außer deren API

- eigene Datenspeicherung für jeden Microservice
 - wird eine gemeinsame Datenbank genutzt, ist das zu verlockend für die Teams die APIs zu umgehen und direkt darauf zuzugreifen
 - Konsistenz der Daten muss auf anderem Wege sichergestellt werden, z.B.
 - Master Data Management
 - Daten können von Services gecached werden und über Events verändert werden
 - aber nur ein Microservice **besitzt** und **ändert**

- der gesamte Code in einem Microservice sollte auf gleichen Reife- und Stabilitätslevel sein (*level of maturity and stability*)
 - Netflix Philosophie: wenn Teile des Codes in einem Microservice geändert werden müssten oder Code dazu kommen müsste, wird der Microservice unverändert gelassen und ein neuer mit den neuen Features geschrieben
 - wird auch als **immutable infrastructure** Prinzip bezeichnet

- getrennte Builds für jeden Microservice
- Deployment in Containern
- betrachte Server als zustandslos
 - Server, insbesondere jene die mit Kunden agieren, sollten austauschbar sein
 - alle sollen die selben Funktionen bieten, so dass keiner einzelner wichtig ist
 - wichtig ist nur, dass es genug davon gibt
 - per *autoscaling* kann die Anzahl angepasst werden

- die Dienste sind möglichst klein
- *architectural principle of single responsibility*
 - entsprechend der Unix-Philosophie: *do one thing and do it well*
- die Organisation sollte automatisiertes Testen und Deployment nutzen
- die Kultur und Designprinzipien sollten gut mit Fehlern umgehen

- Entwicklung kann einfacher skaliert werden
 - ein Service gehört immer einem Team
- leichter zu verstehen
 - wenig Code in einem Service, üblich < 1000 LOC
- einfacher zu deployen
 - Services werden unabhängig voneinander deployed, skaliert und gemanagt
- verbesserte Fehlertoleranz und Isolation
- höhere Entwicklungsgeschwindigkeit
 - Businessanforderungen können schneller umgesetzt werden (was zu beweisen wäre)
- wiederverwendbare Dienste und Rapid Prototyping
 - entsprechend der Unix-Philosophie, Dienste nutzen und neue Funktionalität schnell zusammen bauen (Shell-Scripts)

- der Name führt evtl. in die Irre, Microservices sind nicht notwendigerweise *winzig*
- statt einem Monolith wird ein verteiltes Softwaresystem mit allen seinen Anforderungen implementiert
- partitionierte Datenbankarchitektur
 - Transaktionen über mehrere Businessentitäten nicht mehr einfach durchführbar
 - verteilte Transaktionen sind keine Lösung
- Testen einer Microservices-Anwendung ist sehr viel komplexer
- wenn neue Anforderungen mehrere Services betreffen, müssen diese in der passender Reihenfolge verändert werden
- Deployment der gesamten Anwendung komplexer
 - Hailo hat ca. 160 Microservices, Netflix ca. 600
 - jeder Service kann einzeln skaliert werden

Micro

- Microservice Toolkit <https://micro.mu/>
- **Go Micro**
 - RPC-Framework für Microservices in Go
 - über Plugins: Service Discovery, Load Balancing, Encoding, synchrone und asynchrone Kommunikation
- **Micro API**
 - API-Gateway als Single-Entry-Point
- **Micro Web**
 - Web Dashboard und Reverse Proxy für Micro Web Apps
- **Micro CLI**
 - CLI zur Interaktion mit den Microservices
- **Micro Bot**
 - ein Bot über den mit den Services per Slack, XMPP, E-Mail interagiert werden kann
- **Service Templates**

Ein einfaches Beispiel (1/2)

- [Consul](#) als Service Registry

```
$ docker run -p8500:8500 consul
```

- Micro installieren

```
$ go get github.com/micro/micro
```

- starte den **Greeter**-Service:

```
$ go get github.com/micro/examples/greeter/srv && srv
```

- Services auflisten

```
$ micro list services
```

- Infos über den **Greeter**-Service

```
$ micro get service go.micro.srv.greeter
```

- Service aufrufen

```
$ micro call go.micro.srv.greeter Say.Hello \  
    '{"name": "Oliver Braun"}'
```

- oder über das Web

```
$ micro web
```


- Start mit dem Template

```
$ micro new -type "srv" \  
    github.com/ob-vss-ws18/ob-vss-ws18/micro/greeter
```