Technische Hochschule Ingolstadt
Fakultät für Elektrotechnik und Informatik

Zukunft in Bewegung

Microservices & Cloud

Architektur und Entwurfsmuster der Software Technik

Prof. Dr. Bernd Hafenrichter 29.05.2018

Architekturform: Microservices

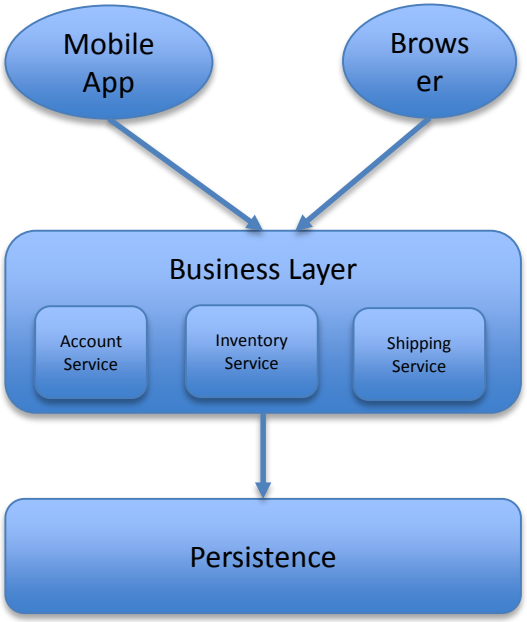
Typische Probleme monolithischer Systeme

- Lange Releasezyklen
- Diskussionen und Abstimmungen über gemeinsam genutzten Code. Wer darf wann was ändern?
- Upstream-/Downstream-Abhängigkeiten von Code verschiedener Teams.
- Komplexe Schnittstellen
- Gemeinsame Nutzung von Datenbanken erschwert die Veränderung der bestehenden Schemata.

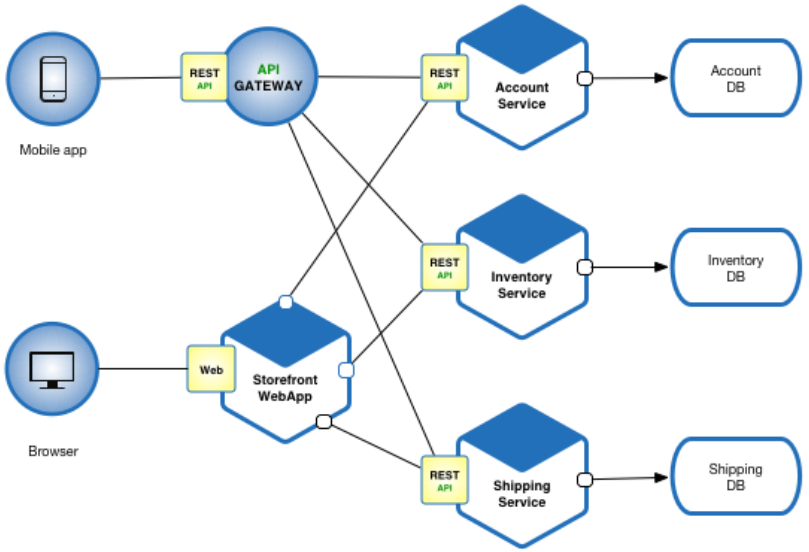
Lösung: Microservices-Architektur

- Definition einer simplen Grundarchitektur, in die die konkreten Features verschiedener Teams unabhängig voneinander eingebettet werden können.
- Teams werden nicht nach Schichten geschnitten, sondern nach Features.
- Sie entwickeln voneinander unabhängig komplette Features, die von der UI bis zur Datenhaltung alles beinhalten.
- Die Features der Teams teilen sich keinen Code und haben keine gemeinsame Datenhaltung (Shared-Nothing-Leitbild).
- Die Features werden als getrennt deploybare Einheiten in die Grundarchitektur eingebettet. Die Kommunikation geschieht über ein gemeinsames Protokoll (z.B. über HTTP in einer REST-Architektur).

Architekturform: Microservices



Monolitihic-Application



Microservices

Architekturform: Microservices

Benefits

- Each microservice is relatively small
  - Easier for a developer to understand
  - The IDE is faster making developers more productive
  - The application starts faster, which makes developers more productive, and speeds up deployments
- Each service can be deployed independently of other services - easier to deploy new versions of services frequently
- Easier to scale development. It enables you to organize the development effort around multiple teams. Each team can develop, deploy and scale their services independently of all of the other teams.
- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- Each service can be developed and deployed independently
- Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack.

http://microservices.io/patterns/microservices.html

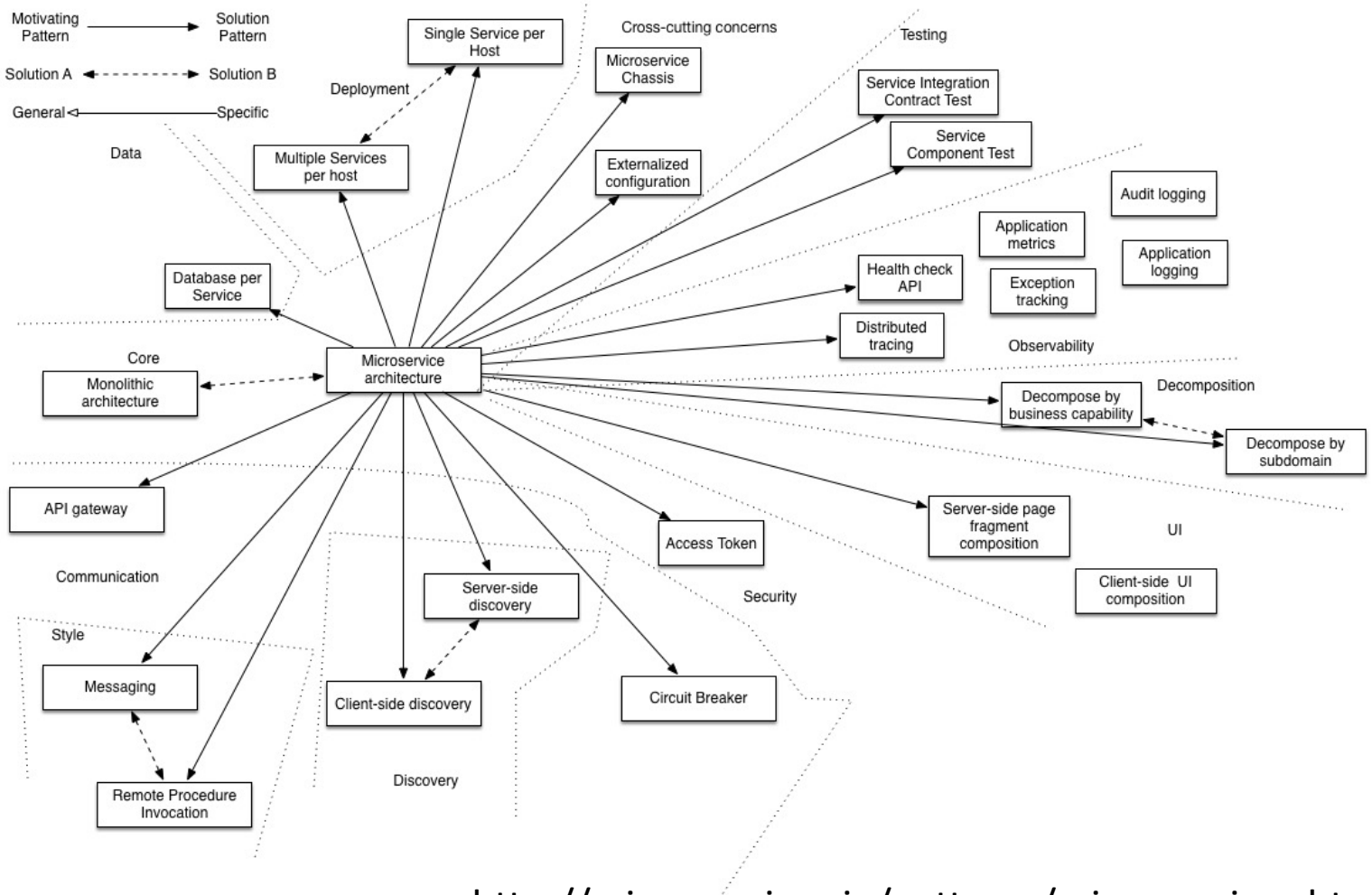
Architekturform: Microservices

Drawbacks

- Developers must deal with the additional complexity of creating a distributed system.
  - Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.
  - Testing is more difficult
  - Developers must implement the inter-service communication mechanism.
  - Implementing use cases that span multiple services without using distributed transactions is difficult
  - Implementing use cases that span multiple services requires careful coordination between the teams
- Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different service types.
- Increased memory consumption. Services run in an own JVM or one can hava one virtual machine per Service

http://microservices.io/patterns/microservices.html

Architekturform: Microservices



http://microservices.io/patterns/microservices.html

Architekturform: Microservices

Patterns für Microservices

Data

- Database per service

Decomposition

- Decompose by business capability
- Decompose by subdomain

Security/Sicherheit

- Access token

Service Discovery

- Client-side service discovery pattern
- Server-side service discovery pattern

http://microservices.io/patterns/microservices.html

Architekturform: Microservices

Physical distribution

- Multiple service instances per host pattern
- Single Service Instance per Host pattern

Observability/Überwachung

- Application metrics
- Distributed tracing
- Health Check
- Log aggregation

Cross-Cutting-Concerns/Querschnittsthemen

- Externalized configuration
- Microservice chassis

http://microservices.io/patterns/microservices.html

Die 12-Factor-App

Heute wird Software oft als Dienst geliefert - auch Web App oder Software-As-A-Service genannt. Die Zwölf-Faktoren-App ist eine Methode um Software-As-A-Service Apps zu bauen die:

- deklarative Formate benutzen für die Automatisierung der Konfiguration, um Zeit und Kosten für neue Entwickler im Projekt zu minimieren;
- einen sauberen Vertrag mit dem zugrundeliegenden Betriebssystem haben, maximale Portierbarkeit zwischen Ausführungsumgebungen bieten;
- sich für das Deployment auf modernen Cloud-Plattformen eignen, die Notwendigkeit von Servern und Serveradministration vermeiden;
- die Abweichung minimieren zwischen Entwicklung und Produktion, um Continuous Deployment für maximale Agilität ermöglichen;
- und skalieren können ohne wesentliche Änderungen im Tooling, in der Architektur oder in den Entwicklungsverfahren.

https://12factor.net/de/

Literaturverzeichnis

https://jaxenter.de/prinzipien-skalierbarer-architektur-848

https://www.informatik-aktuell.de/entwicklung/methoden/skalierbare-anwendungsarchitektur.html

https://www.it-agile.de/wissen/agiles-engineering/skalierbare-software-architekturen/

https://blog.codecentric.de/2011/08/grundlagen-cloud-computing-cap-theorem/

http://www.wikipedia.de