

Einführung

Microservices

Msc. Informatik

06. April 2021

Organisatorisches

Generelles

Veranstaltung:

- Vorlesung: Dienstags 5. Block
- Praktikum: Dienstags 6. Block

Kontakt:

- tom.kaesler@mni.thm.de
- per Discord

Über mich



Ein paar Worte über mich

- ehem. Masterstudent an der THM
- ehem. Betreuer der THM GitLab Plattform
- Einer der Hauptentwickler von ARSnova
- Beteiligt an diversen Projekten (THM cards, arsnova.click)
- Dozent für *Microservices*, *SWT-P*, *NVP* und neuerdings auch *Architekturen von Webservices* und *Verteilte Anwendungsentwicklung*
- Ein Jahrzehnt an Erfahrungen mit Websystemen

Praktikum

- 3er Gruppen
- Verwaltung über THM GitLab
- Leitfaden & Hilfestellung
- Präsentationen
- (Etwas) Zeit zum Coden

Prüfungsleistung

- Microservices-Projekt
- Alle Konfigurationsdateien (k8s, docker, ...)
- Stresstest (inkl. Testdaten)
- Monitoring-System
- Dokumentation
- Abschlusspräsentation

Einführung

Worum geht's?

Big Data, Cloud, Hadoop, Akka, Go, IaaS, SaaS, FaaS, Openshift, Kubernetes, Prometheus, Istio, Docker, Docker Swarm, Docker Compose, GitLab, Jenkins X, Kafka, gRPC, REST, GraphQL, ...

- Webservices mit REST-like Schnittstellen
- Datenhaltung mit NoSQL / SQL
- Docker
- Kubernetes (Minikube)
- Event-System in einem Microservice-Ecosystem
- Monitoring und horizontale Skalierbarkeit
- CI/CD/GitLab

Definition

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.¹

– Martin Fowler

- Lose gekoppelte Services, die Business-Logic implementieren
- Skalierbar
- Ausfallsicher
- Heterogenes System

¹<https://martinfowler.com/articles/microservices.html>

Woher kommen Microservices?

Wichtige Zutaten

- Domain-Driven Design
- Continuous Delivery
- Hexagonale Architektur
- Virtualisierungsplattformen
- Automatisierte Infrastrukturen

Aus diesen Zutaten sind letztlich Microservices hervorgegangen. Sie wurden nicht vorab entwickelt, sondern entstanden aus den bei der praktischen Anwendung zu beobachtenden Tendenzen und Mustern.

Eigenschaften und Vorteile

- Klein und darauf spezialisiert, eine bestimmte Aufgabe richtig gut zu erledigen
- Eigenständigkeit
- Einsatz verschiedenartiger Technologien
- Belastbarkeit
- Skalierbarkeit
- Komfortables Deployment
- Betriebliche Abstimmung
- Modularer Aufbau
- Austauschbarkeit

Wann sollten Microservices eingesetzt werden?

Gründe für Microservices:

- Notwendigkeit der Skalierbarkeit
- kleine, unabhängige Teams
- Verschiedenste Software mit verschiedensten Anforderungen (IoT)
- Leichte Erweiterbarkeit gefordert
- Bedarf an hoher Ausfallsicherheit

Beispiele aus der Produktion

- Netflix
- Monzo
- Spotify
- ebay
- Amazon
- Twitter
- SoundCloud
- Otto
- Zalando
- ...

Service-Oriented Architecture

Service-Oriented Architecture (SOA) ist ein Designansatz, bei dem mehrere Services kollaborieren, um einen Satz an Funktionalitäten bereitzustellen.

- SOA wurde entwickelt, um die Wiederverwendbarkeit von Software zu fördern
- Unklar, wie SOA *richtig* umgesetzt werden soll
- SOA kämpft mit vielen Problemen:
 - ▶ Kommunikationsprotokolle (SOAP)
 - ▶ Software zum Datenaustausch (Middleware) verschiedener Hersteller
 - ▶ fehlende Orientierungshilfe bei der Service-Aufteilung
 - ▶ falsche Empfehlungen zur Aufspaltung des Systems

Microservices ist ein aus der Praxis entstandener SOA-Ansatz, der nicht von Herstellern getrieben wurde, die ihre Tools verkaufen wollen.

Microservices und SOA verhält sich wie Extreme Programming oder Scrum zur agilen Softwareentwicklung.

Verfahren zur Aufspaltung

Die Vorteile der Microservices gehen auch auf einen wesentlichen Aspekt zurück: Aufteilung der Services. Können die Vorteile auch durch andere Verfahren der Aufspaltung erzielt werden?

- Programmbibliotheken
 - ▶ Wiederverwendbarkeit und Teamorganisation gegeben
 - ▶ Bibliothek ist in der Regel an die gleiche Sprache wie das Hauptsystem gebunden
 - ▶ Ohne dynamische Einbindung ist kein separates Deployment möglich
 - ▶ Belastbarkeit/Skalierbarkeit durch Bibliotheken nicht naheliegend
- Module
 - ▶ OSGI: Hohe Komplexität, die Vorteile nicht aufwiegt
 - ▶ Erlang: Gute Möglichkeiten zur Modularisierung, dennoch Beschränkung auf eine Technologie
 - ▶ Module allgemein: So viel wir uns anstrengen, ohne harte Isolation (Prozesse!) neigen wir dazu, unsere Module untereinander zu stark zu koppeln.

Der Umfang eines Services

Ein Microservice sollte immer eine kleine im Sinne der Business-Logic zusammengehörige Funktionalität implementieren.

Typischer Scope:

- Eine Ressource im REST-Kontext
- Userverwaltung inkl. Autorisierung
- Suchmaschine
- Newsfeed / Akkumulation von Daten
- Aufwendige Berechnung (Statistiken)

Bounded Context³

[...] total unification of the domain model for a large system will not be feasible or cost-effective.²

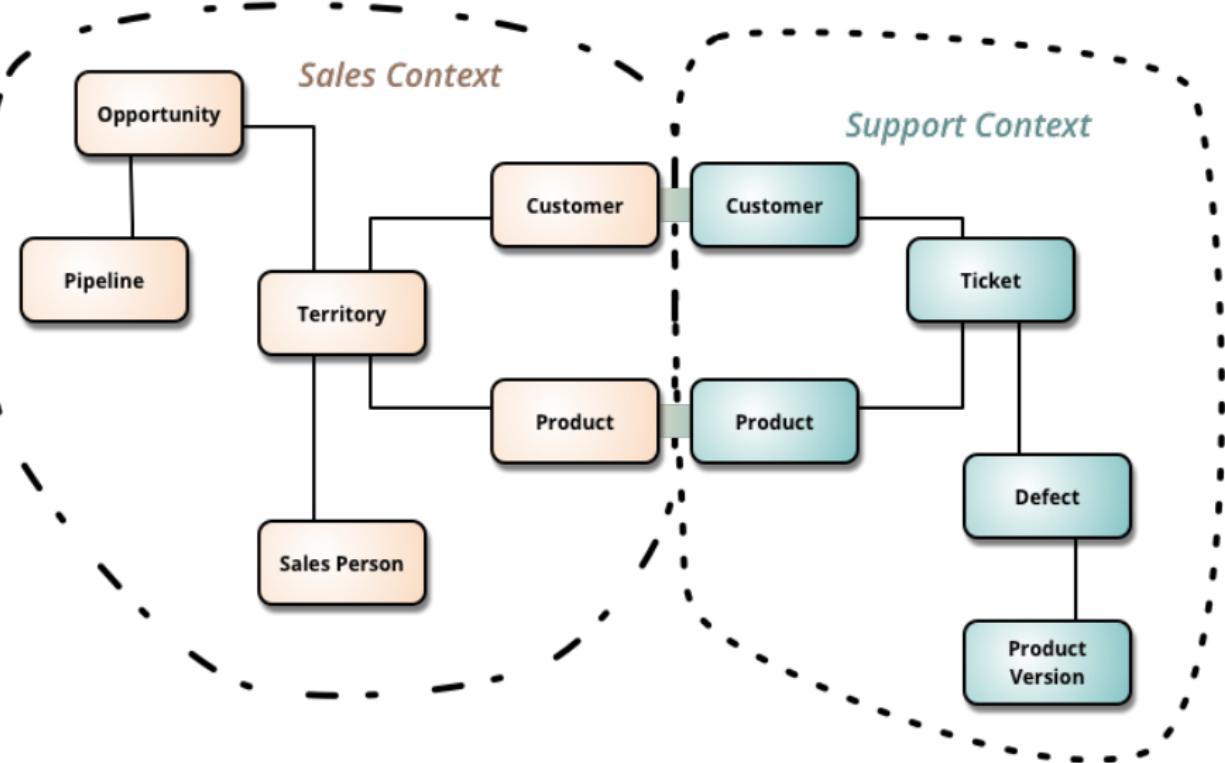
– Eric Evans

Ein Model für die Software wird schnell zu komplex. Daher wird der Kontext eingegrenzt:

- Ein Bounded Context hat alleine schon Daseinsberechtigung
- Bounded Contexts teilen sich nur zu Teilen Konzepte wie z.B. User
- Domain- Driven Design Patterns dienen dem Austausch zwischen Bounded Contexts

²Domain-Driven Design: Tackling Complexity in the Heart of Software

³<https://martinfowler.com/bliki/BoundedContext.html>



Von Root-Servern zum Cluster

"Monolithische" Software

- Software auf dedizierten Servern
- Skalierbarkeit begrenzt durch Hardware
- Verteilte Systeme werden schwer zu managen

Microservices

- Aufteilung der Business-Logic in (micro) Services
- Hardware on-demand aus "der Cloud"
- Skalierung in die Horizontale

Unterschiede: Monolith - Microservices

Monolithische Software:

- Alle Business-Logic in einem Prozess
- Eine Architektur für alle Anforderungen
- Cache sitzt i.d.R. direkt im System
- Eine API in einer Version

Microservices

- Business-Logic aufgeteilt
- Kleine Services mit speziellem Stack
- Dedizierte Cache Server
- Verteilte API, eventuell mit API Gateway

Was ist mit Legacy Software?

Fast alle großen Microservice-Systeme sind aus Monolithen entstanden:

- Skalierungsbedarf kommt selten über Nacht
- Monolithen sind deutlich simpler
- Teamgröße oft anfangs überschaubar

Ein monolithisches System ist keine Sackgasse:

- Event-System mit Anbindung an den Monolith
- Neue Features als eigene Services
- Alte Features können als Bounded-Context refaktorisiert werden

Die Schattenseite

- Komplexe Architektur
- Overhead durch Kommunikation im verteilten System
- Netzwerklatenz
- Setup (Ecosystem/Deployment) wird deutlich umfangreicher

Der 'Ballast' in einem Microservice-Ecosystem

- Datenbankmanagementsystem(e)
- Message-Broker: Events
- Cluster-Management-Komponente
- API-Gateway
- Zugriffsregeln
- Monitoring
- Proxys

Leitfaden der Veranstaltung

- ① Intro
- ② Vorstellung eines Microservices
- ③ Distributed-Data
- ④ Event-System/Event-Sourcing/CQRS
- ⑤ Message-oriented Middleware
- ⑥ Service-Mesh
- ⑦ Kubernetes
- ⑧ Production-Readiness
 - ▶ Monitoring
 - ▶ Testing
 - ▶ Updates/Changes
 - ▶ Dokumentation

Ein kleiner Hintergrund

Warum ich mich mit Microservices beschäftige

Microservice-basierte Architekturen sollten nicht leichtfertig verwendet werden.

Die Komplexität ist hoch, der initiale Aufwand noch höher, und viele der Vorteile kommen erst bei großen Anwendungen, entwickelt von vielen Teams und genutzt durch tausenden gleichzeitige Clients zum tragen.

Wie verträgt sich diese Aussage mit dem Projekt dieser Veranstaltung und auch ARSnova also (micro)service-basiertem System?

Ein großes Projekt vs. viele kleine Projekte

Die Aufteilung in Teams mit Eigenverantwortlichkeit ist eine Stärke von Microservices. Die naheliegende Option für das Projekt wäre daher eine Aufteilung der Gruppen auf die einzelnen Services und die Entwicklung eines großen Projektes.

Wer übernimmt dann allerdings die architekturellen Aufgaben der Inter-Service Kommunikation? Wer schafft die Regeln? Kümmert sich im die Orchestrierung?

Daher soll ein vom Funktionsumfang kleines Projekt von jeder Gruppe einzeln übernommen werden.

ARSnova als verteiltes System

ARSnova ist ein Websystem, welches durch meine Zeit als Masterstudent mitlerweile in eine service-orientierte Architektur überführt wurde. Es handelt sich im Grunde um Microservices, allerdings ist ein Service deutlich umfangreicher und entspricht dem **Core** der Anwendung.

Es hat sich vor allem durch Refactoring eines Featuresets die Gelegenheit ergeben, einen eigenes Backend dafür zu schreiben. Dadurch mussten jedoch Mechanismen zur Synchronisation, Authorisierung etc. eingeführt werden.

Mittlerweile existieren diverse Microservices in dem Ecosystem, inklusive einer für Microservice-basierte Systeme gängiger Patterns wie z. B. Gateways.

Zu guter Letzt...

... ist es das Interesse an dem Architekturstil.

Bei architekturellen Fragen gibt es keineswegs einfache und vorgefertigte Lösungen.

Vor- und Nachteile müssen abgewogen, Möglichkeiten der Umsetzung evaluiert werden.

Diese Veranstaltung dient daher insbesondere einem Zweck: Erfahrung mit diesem Architekturstil sammeln, um für spätere Projekte auf die Erfahrung zurückgreifen und qualifizierte Entscheidungen treffen zu können.

