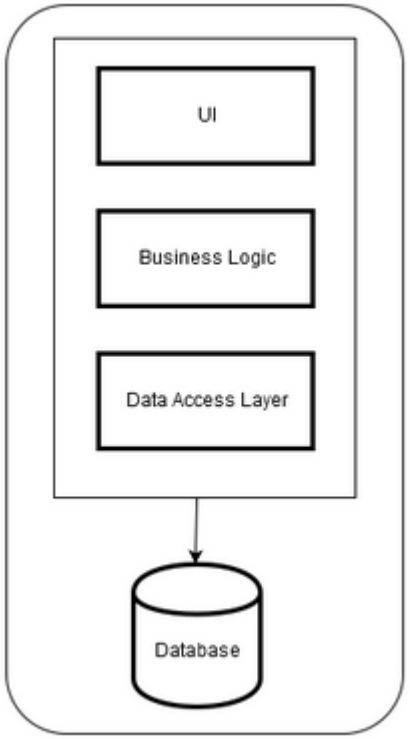
Microservices

Autor: Sebastian Janzen

Monolith

- Monolithische Anwendung wird als ein Ganzes entwickelt
- Drei-Schichten-Architektur
  - Clientseitige Applikation
  - Serverseitige Applikation
  - Datenbank



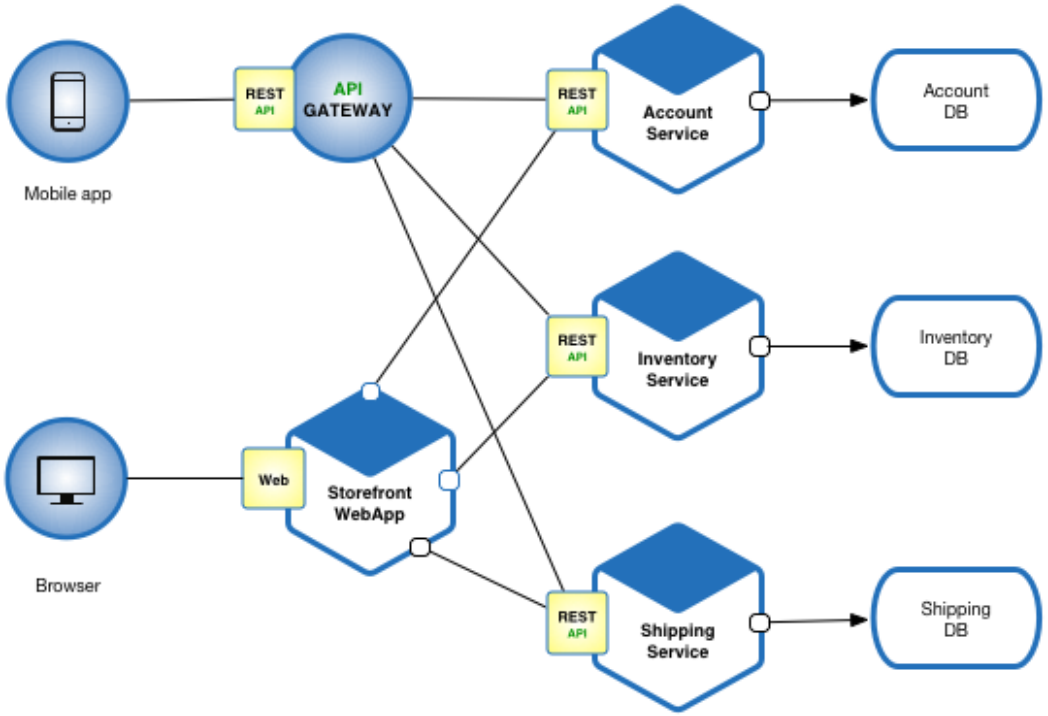
Monolithic Architecture

Monolith

- Zum Beispiel eine serverseitige Applikation ist zuständig für Zugriffe auf die Datenbank und Interaktionen mit dem Browser
- Eine Änderung im System führt zu einer neuen Softwareversion
- Die ganze Logik konzentriert sich in einer ausführbaren Datei
- Für neue Änderungen und Korrekturen muss das ganze System neu erstellt werden, das macht auf Dauer eine solche Anwendung schwierig zu entwickeln
- Es ist aufwändig die Modularität des Systems aufrecht zu erhalten, ohne das ungewollte Abhängigkeiten im System entstehen

Microservices

- Ein Microservice ist ein leichtgewichtiger autonomer Dienst der eine einzige Aufgabe erfüllt und mit anderen ähnlichen Diensten über eine gut definierte Schnittstelle kollaboriert.



Charakteristiken von Microservices

Komponentisierung via Services

- Als Komponenten werden einzelne Services und Bibliotheken verstanden
- Der Vorteil eines Komponenten ist in der Unabhängigkeit
- Bei Änderung einer einzelnen Bibliothek in einem Prozess, müsste die gesamte Anwendung neu aufgesetzt werden
- Wäre ein Services verändert wären andere Services unbetroffen
- Einzelne Softwarekomponenten sind besser voneinander getrennt

Charakteristiken von Microservices

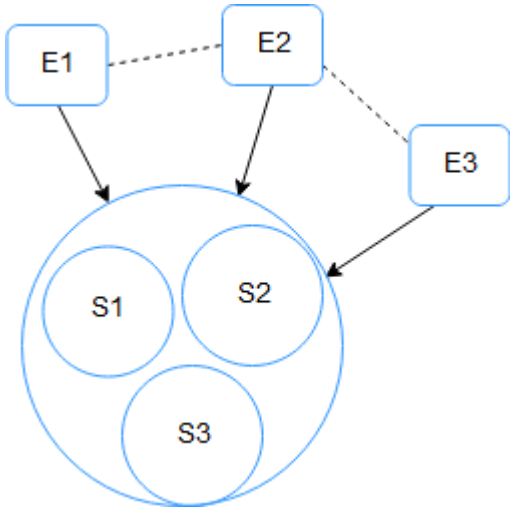
Aufbau um Business Capabilities

- Business Capabilities definieren wichtige Businessfunktionen
- Sie beschreiben was das Kernfähigkeiten eines Unternehmen sind
- Microservices sollen um Business Capabilities aufgebaut werden
- Gesetz von Conway
  - „Jede Organisation die ein System entwirft bekommt am Ende ein Entwurf welches die Kommunikationsstruktur der umzusetzenden Organisation nachbildet “

Charakteristiken von Microservices

Aufbau um Business Capabilities

- Gesetz von Conway Beispiel
  - Entwicklerteams E1-E3 arbeiten zusammen am Projekt
  - Am Ende wird die Architektur aus Subsystemen S1-S3 bestehen
  - Qualität und Art der Schnittstellen wird der Kommunikation zwischen den Teams entsprechen



Charakteristiken von Microservices

Service als Produkt

- Software sollte als ein Projekt, sondern als ein Produkt angesehen werden
- Entwicklerteams sollten während des gesamten Lebenszyklus sich um ihr Produkt kümmern
- Vorteile sind ständiger Feedback von Nutzern und engere Bindung an das eigene Produkt
- Software wird mehr als ein Bündel von Funktionalitäten
- Es wird eine andauernde Beziehung aufgebaut, um die Business Capability zu verbessern

Charakteristiken von Microservices

Smart endpoints and dumb pipes

- Applikationen der Microservices sollten abgekoppelt und zusammenhängend wie möglich sein
- Zum Beispiel: Empfangen einer Anfrage, Bearbeitung und senden einer Antwort
- Protokolle sollten hingegen unkompliziert sein
- Eine Möglichkeit wäre eine Infrastruktur zu nutzen, die nur als Router von Nachrichten fungiert (dumb pipes)

Charakteristiken von Microservices

Dezentralisierte Führung

- Das ist ein Konzept welches erlaubt den einzelnen Entwicklerteams ihre Stärken in verschiedenen Technologien anzuwenden und zusammen ein leistungsfähiges Produkt zu entwickeln
- Passende Technologie pro Service: **Polyglot Programming**
- Entscheidungen über Entwicklung liegen auf Ebene der Entwickler
- *You build it, you run it!* – Konzept von Amazon
- Entwicklung, Installation, Überwachung und Steuerung wird von Entwicklern übernommen

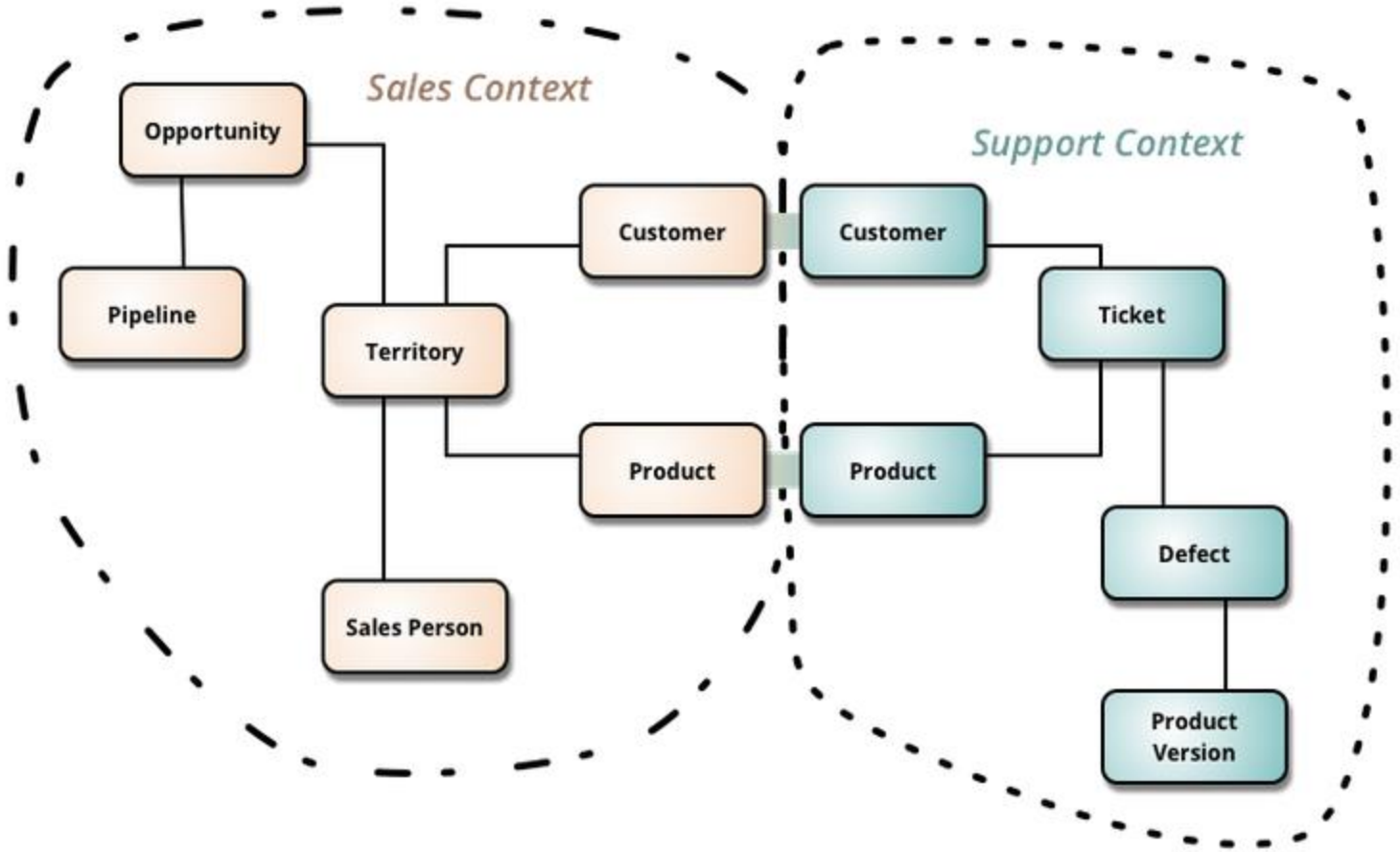
Charakteristiken von Microservices

Dezentralisiertes Datenmanagement

- **Bounded Context** beschreibt eine Abgrenzung in der ein bestimmtes Modell definiert und verwendet wird
- Es teilt komplexe Domänen in mehrere Kontextgrenzen und beschreibt die Beziehungen zwischen ihnen
- Microservices besitzen eine natürliche Korrelation zu Bounded Context, denn dieser verdeutlicht und verstärkt eine Trennung in verschiedene Kontextbereiche

Charakteristiken von Microservices

Dezentralisiertes Datenmanagement



Charakteristiken von Microservices

Dezentralisiertes Datenmanagement

- Dezentralisierung bei Microservices betrifft auch die Datenbanken
- Es wird bevorzugt pro Service eine Datenbank zu haben
- Eine der Probleme von dezentralisiertem Datenmanagement ist Update Management
- Es ist schwierig die Datenkonsistenz zu garantieren
- Mögliche Probleme werden von kompensierenden Operationen abgefangen (Vorgänge werden rückgängig gemacht)

Charakteristiken von Microservices

Infrastructure Automation

- Testautomatisierung und Ansätze wie Continuous Integration und Continuous Delivery helfen bei der Entwicklung von stabiler und hochwertiger Software

Charakteristiken von Microservices

Design for failure

- Ein Service sollte immer gegen mögliche Fehlerfälle ausgiebig getestet werden
- Komplexer bei Microservices als bei monolithischen Design
- Da nicht alle möglichen Probleme abgedeckt und vorhergesehen werden können, kommt Echtzeit-Monitoring zum Einsatz

Charakteristiken von Microservices

Evolutionäres Design

- Eine Software sollte so entworfen werden, dass statt es bei größeren Änderungen zu verwerfen, diese weiterentwickelt werden kann
- Die wichtigsten Eigenschaften eines Komponenten sind Austauschbarkeit und Erweiterungsfähigkeit
- Oftmals wird ein Service verworfen anstatt auf lange Sicht überarbeitet zu werden

Microservices: Vor- und Nachteile

Vorteile	**Nachteile**
Kompakter Quellcode	Größerer Ressourcenbedarf
Leichterer Einstieg ins Entwicklerteam	Use Cases nicht eindeutig
Erhöhte Einsatzfähigkeit	Sehr komplex
Kürzere Entwicklungszeiten	Überhang an Services möglich
Autonome Services	Testsfälle werden komplexer
Kleine dedizierte Teams	Abhängig von Schnittstellen
Resistent gegen Ausfall	Probleme mit Datenkonsistenz
Business Case abhängig	Aufwändig zu managen
Technologie unabhängig	Großer Kommunikationsaufwand pro Team
Wiederverwendbare Services	

Humane Registries

- Humane Registry ist eine automatisierte Dokumentation für Microservices
- Designt um Informationen automatisch in menschlich lesbaren Form zu schreiben und zu aktualisieren
- *Verständlichkeit*: Format sollte für alle lesbar und verständlich sein
- *Automation*: Entwickler haben selten Zeit eine Dokumentation zu pflegen
- *Einfachheit*: Erweiterung der Informationen sollte unkompliziert sein
- Dokumentationswerkzeug durchsucht den Quellcode des Systems und stellt detaillierte Informationen darüber bereit

Serverless

- Es wird komplett auf die serverseitige Logik verzichtet und stattdessen werden Cloud-Services von Drittanbietern integriert
- Zu solchen Cloud-Services zählen Datenbanken und Authentifizierungsmechanismen: **Backend as a Service**
- Ein anderer Bereich von Serverless ist **Function as a Service**
- Serverseitige Logik immer noch vom Entwickler geschrieben, jedoch verpackt in Container und von einer Cloud gemanagt

Micro Frontends

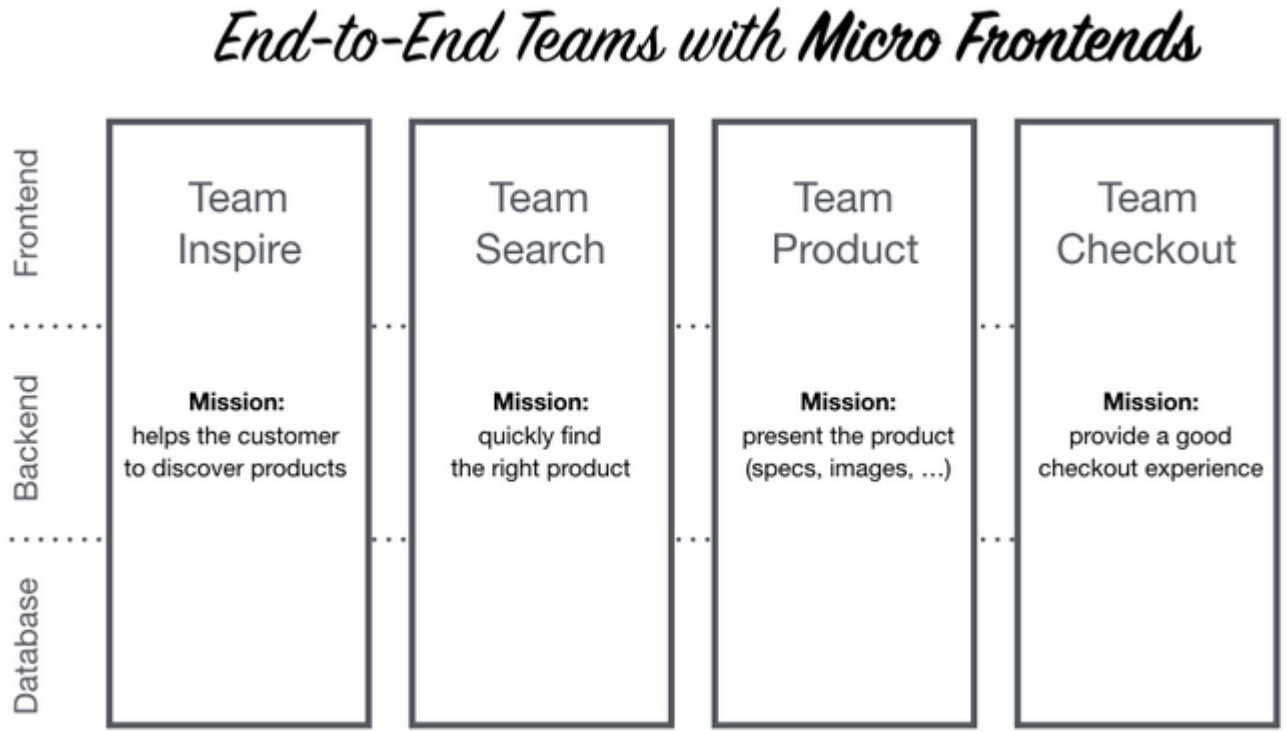
- Für Webanwendungen gewinnt Front-End immer mehr an Bedeutung
- Der Trend geht in Richtung einer 90 zu 10 Aufteilung zu Gunsten von Front-End
- Der aktuelle Trend heißt "Micro Front-Ends" und Unternehmen, wie Spotify und Zalando sind schon umgestiegen

Micro Frontends

Vorteil	**Nachteil**
Unabhängig	Erhöhter Betriebsaufwand
Einfacher einzusetzen	Erhöhte Komplexität (z.B. Infrastruktur, Kommunikation)
Hohe Testbarkeit	Schlechtere Performance
Unabhängige Technologiestacks	Restrukturierung kann sehr komplex werden
Unabhängig im Fehlerfall	
Parallele Entwicklung möglich	

Micro Frontends

- Vertikale Aufteilung



Self-contained-Systems

- Self-contained-System (SCS) ist ein Architekturmuster mit Fokus auf Separation einer Funktionalität in mehrere unabhängige Systeme
- Aus der Kollaboration dieser Systeme entsteht ein neues logisches System
- Probleme von Monolithen werden umgangen, die durch ständiges Wachstum unwartbar werden
- Ähnlichkeit zu Microservices
  - Unabhängige Einheiten
  - Technologische Diversität
  - Dezentralisierte Infrastruktur

Self-contained-Systems

- Ein Microservices ist üblicherweise kleiner als ein SCS
- Weniger SCSs als Microservices
  - Ein E-Commerce Shop kann 5-25 SCSs haben oder bis zu 100 Microservices.
- SCSs besitzen ein UI, während Microservices es üblicherweise auslagern

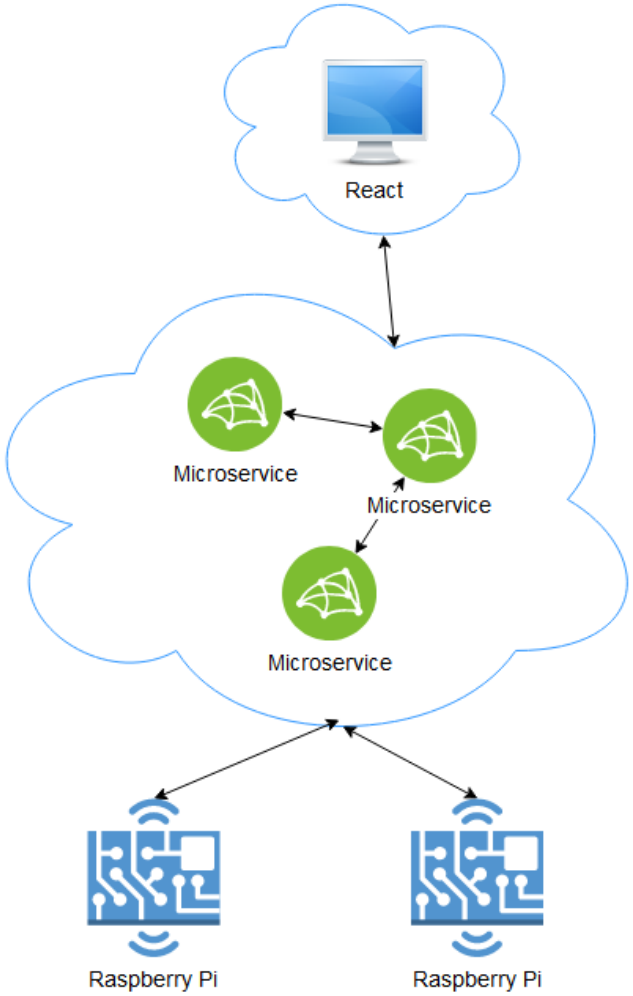
Demo

Story

- Smart Home Demo
- Zwei Raspberry Pis mit Sensoren und LEDs
- Microservices für Sensoren und LEDs
- Microservices sind in AWS Cloud
- React Dashboard zeigt Temperatur, Helligkeit, Luftfeuchtigkeit und kann die LEDs steuern

Architektur

- cote: Node.js Bibliothek
- Microservices in Javascript



Code

https://github.com/janzse/microservices

Quellen

- Abbildung Folie 3: http://microservices.io/patterns/microservices.html

Fragen und Feedback