Microservices

by

Dr. Günter Kolousek

Ausgangslage

- Monolithisches System
  - untrennbare Einheit
  - unabhängig von anderen Systemen
  - single-tiered
    - UI, Database access, Business logic
- $\rightarrow$ SOA
  - höhere Komplexität durch Entkopplung der Dienste
  - höhere Komplexität durch WS-* Spezifikationen (wenn diese eingesetzt werden)
  - erschwertes Debugging, Logging und Testen

Microservices

- Jedes Service stellt einen abgegrenzten Teil der Anwendung dar
  - im Gegensatz zu SOA $\rightarrow$ Anwendung setzt sich aus einzelnen Diensten (wie z.B. ”Abrechnung erstellen”) zusammen
  - $\rightarrow$ DDD (domain driven design): bounded context
- ”Do one thing and do it well”
  - $\rightarrow$ Unix Philosophie
    - Größe: klein
      aber nicht zu klein $\rightarrow$ #Nachrichten, Latenz, Fehler
  - $\rightarrow$ SRP (single responsiblity principle)
- ”should have a universal interface”
  - Unix Philosophy: Textstream
  - HTTP: uniform interface

Microservices – 2

- Anordnung: verteilt
  - Prozesse, die über Netzwerk miteinander kommunizieren
  - plattformübergreifende Protokolle
  - eigenständige Implementation
    - verschiedene Programmiersprachen, DBMS, HW- und SW
- trotzdem: keine genaue Definition von Microservice
- Service Contract
  - verbindliche Vereinbarung zwischen Service und Clients
  - Contract Versioning
- Service Typen
  - Functional services
  - Infrastructure services (nicht öffentlich sichtbar)
    - Authentifizierung, Autorisierung, Geheimhaltung, Integrität, Logging, Monitoring

Vorteile

- überschaubar für jedes Teammitglied
  - …und besser auf die Organisation abstimmbar
  - Teamgröße: 5–9
- steigert Kohäsion, verringert Kopplung
  - und damit die Zusammensetzbarkeit (composability)
- können unabhängig von einander entwickelt werden
- Continuous Delivery einfacher
  - eine einzige Codezeile ändern…
- austauschbar
- Technologiestack kann leichter aktuell gehalten werden
  - jedes Microservice eigenen Technologiestack
- können unabhängig von einander skaliert werden

Nachteile

- Testen und Softwareverteilung (deployment) komplexer und aufwändiger
- höhere Komplexität auf Grund verteilter Architektur
- Overhead
  - Schnittstellen und Schnittstellendefinition
  - Laufzeit (Speicher, Netzwerk, Serialisierung/Deserialisierung)
  - (Netzwerk, Speicher)
- Gesamtkomplexität wird nicht geringer

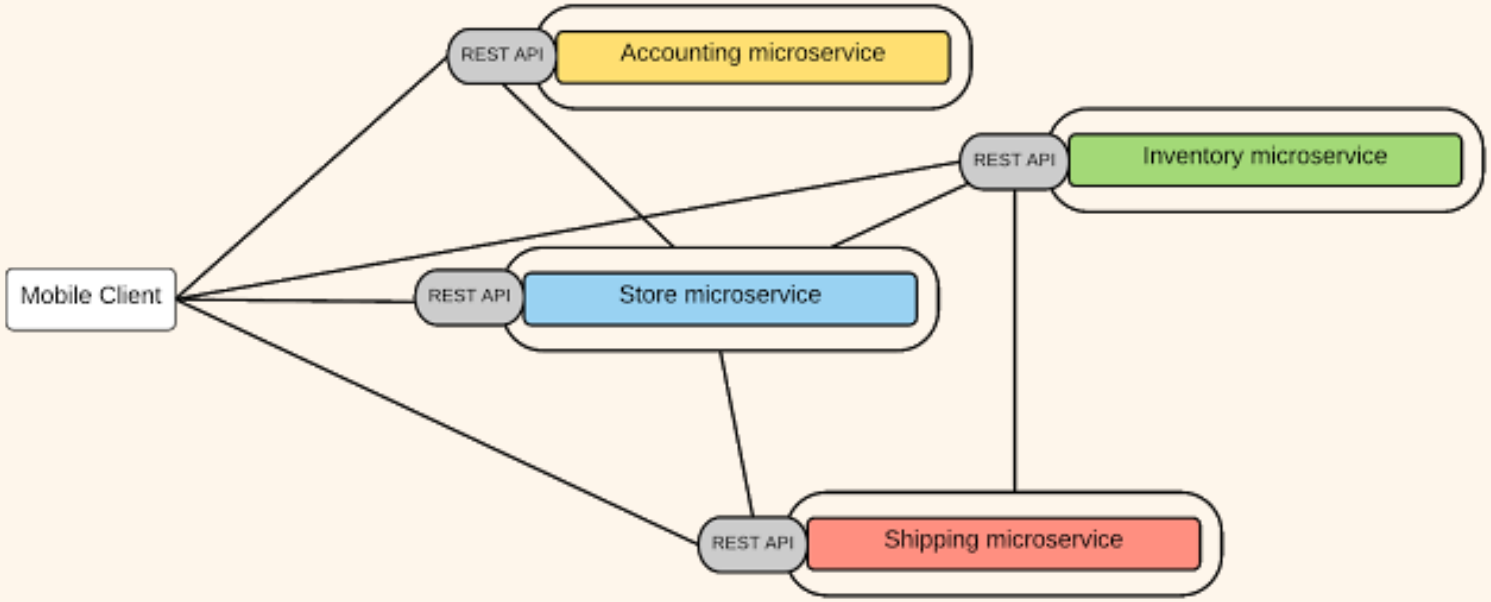
  You can move it about but it's still there!

  Robert Annett

Kommunikation

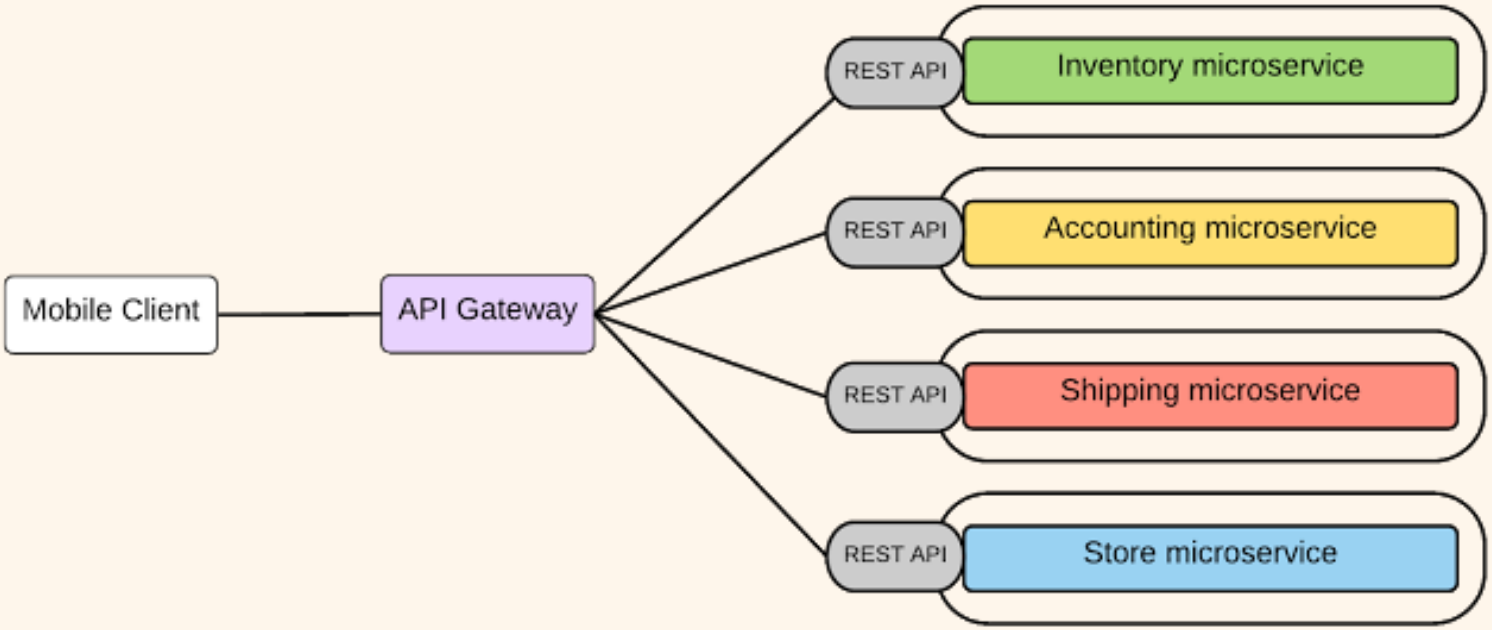
- synchron: WS-* (SOAP,…), REST, grpc, RMI, .Net,…
- asynchron: AMPQ, MQTT,…
- Nachrichtenformat: protobuf, JSON, XML,…
- Kommunikationsstile
  - Point-to-point Stil: direktes Aufrufen des Service
  - API Stil: unter Verwendung eines Gateways
    - Client kommuniziert mit einem Gateway
    - Gateway: leitet weiter; Sicherheit; Monitoring; Transformation
  - Message Broker Stil: Kopplung der Services über Publish/Subscribe
    - Entkopplung
    - besser skalierbar

Point-to-point Stil



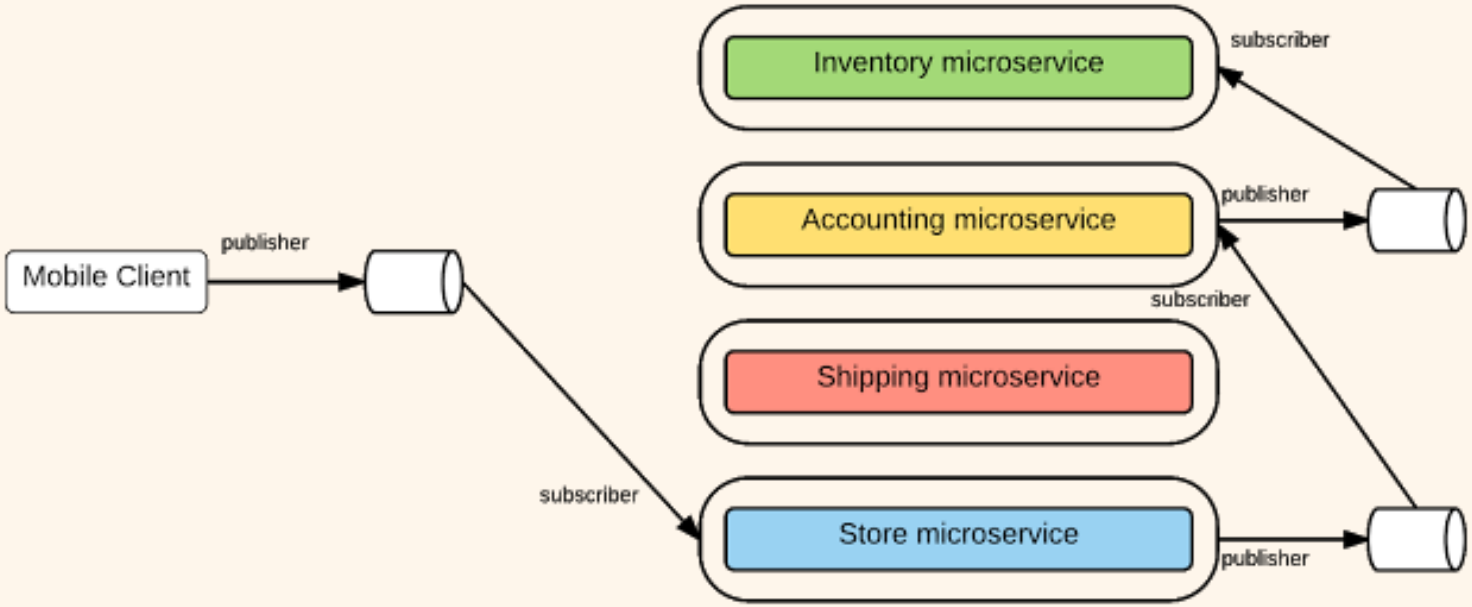
Quelle: https://dzone.com/articles/microservices-in-practice-1

API Stil



Quelle: https://dzone.com/articles/microservices-in-practice-1

Message Broker Stil

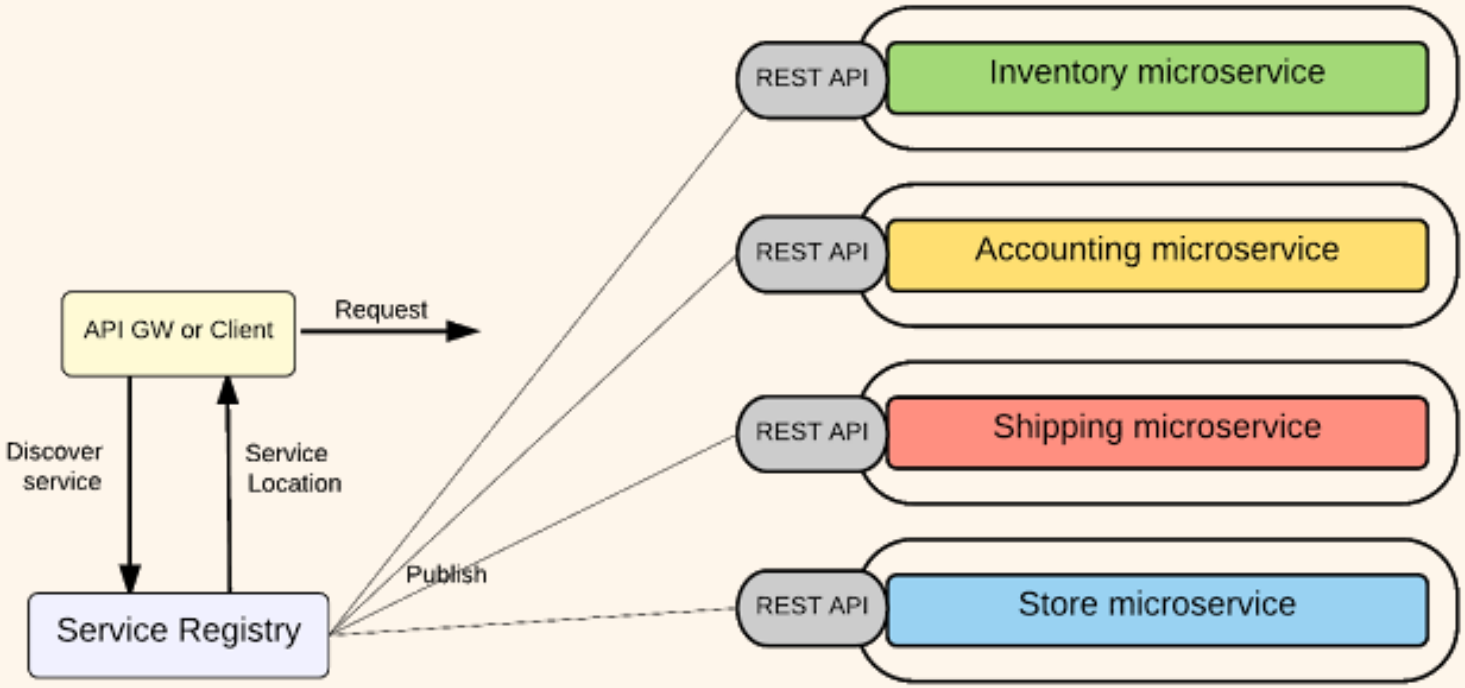


Quelle: https://dzone.com/articles/microservices-in-practice-1

Service Discovery

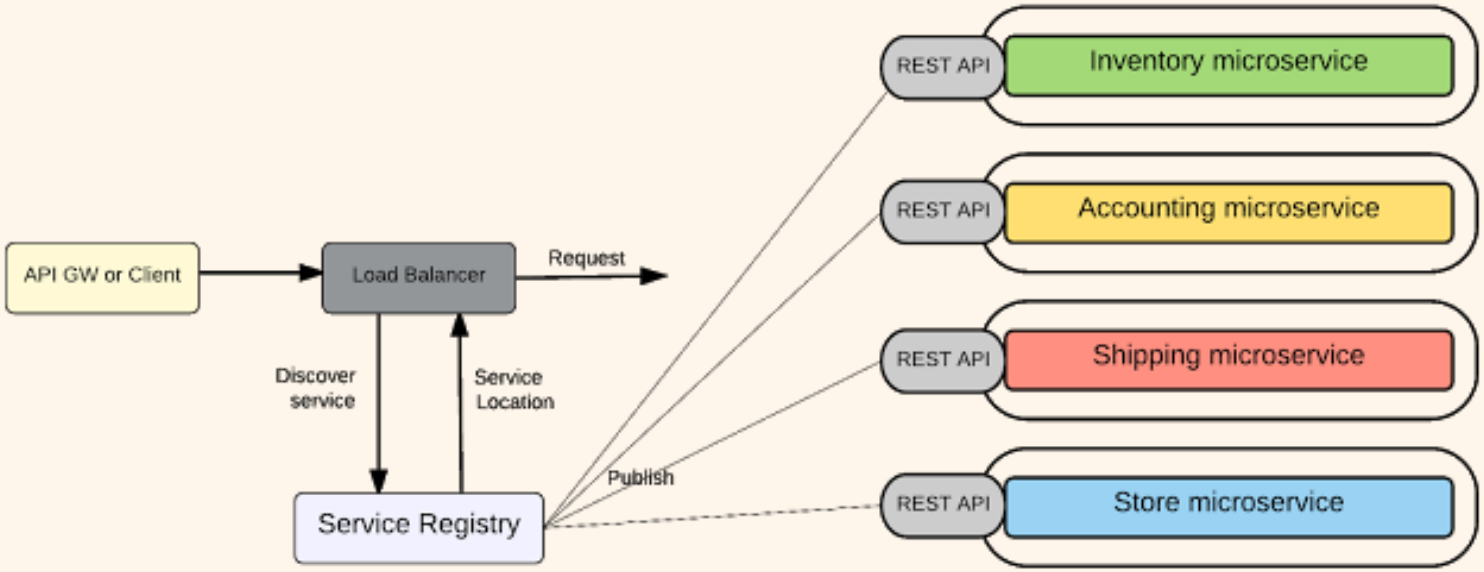
- Client-side: Client bzw. API GW stellt Anfrage an Service Registry
- Server-side: Client bzw. API GW an Komponente wie Load-Balancer (mit well-known Adresse) und diese Komponente stellt Anfrage an Service Registry

Client-side Discovery



Quelle: https://dzone.com/articles/microservices-in-practice-1

Server-side Discovery



Quelle: https://dzone.com/articles/microservices-in-practice-1

Schichtenarchitektur

Jedes Microservice besteht prinzipiell aus

- Presentation Layer
  - z.B. HTML mit Links zu anderen Microservices $\rightarrow$ REST
- Application Layer
- Data Layer
  - $\rightarrow$ dezentrale Datenhaltung!

Dezentrale Datenhaltung

- jedes Microservice
  - …ist für eigene Datenhaltung verantwortlich!
  - …hat eigene Datenbank
- Kopplung nur über API
  - ändern von Daten in anderem Microservice nur durch Schnittstelle, die das jeweilige Microservice anbietet
  - vorteilhaft: Message Broker Stil
- $\rightarrow$ Datenhaltung dezentralisiert
  - jedes Microservice eigene DB
  - ansonsten Microservice nicht unabhängig!

Dezentrale Datenhaltung – 2

- Probleme
  - …bei Transaktionen über mehrere Microservices!
    - Sichtweise: wenn notwendig, dann prinzipiell ein Entwurfsfehler
    - SRP $\rightarrow$ Fehler, dann müssen Operationen der vorhergehenden Microservices rückgängig gemacht werden!
  - …bei Konsistenz der Daten!
    - Größe des Microservice groß genug, dass konsistente Daten in einem Microservice!

Deployment

- prinzipiell schwieriger, da hohe Anzahl an Microservices
- Möglichkeiten
  - Virtuelle Maschinen: z.B. HyperV, VirtualBox, KVM
  - Container: z.B. Docker, Kubernetes, LXC
  - Prozesse

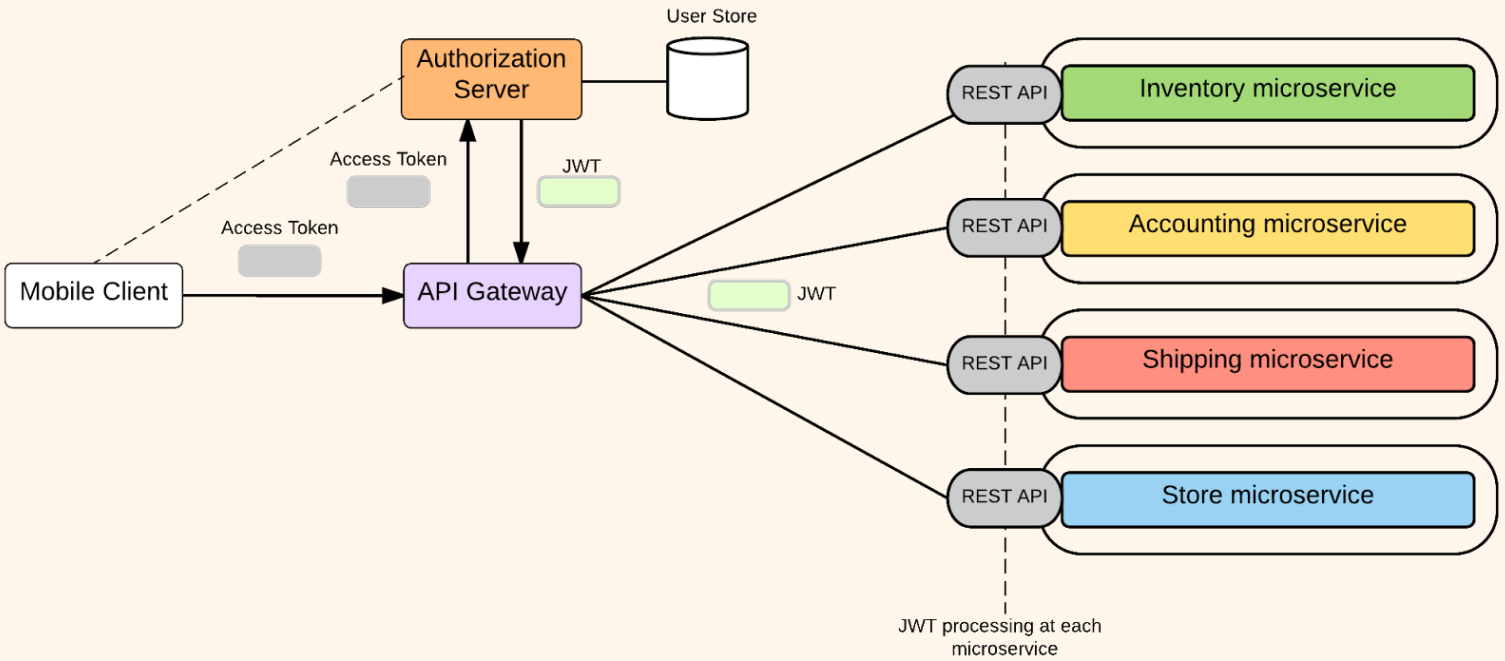
Sicherheit

- Authentifizierung und Autorisierung
- OpenID
  - Protokoll zur dezentrale Authentifizierung
  - OpenID Provider verifiziert Identität $\rightarrow$ Token
- OAuth2
  - Protokol, um Anwendungen Zugriff auf eine Ressource zu erlauben
- SAML
  - Security Assertion Markup Language

Sicherheit – 2

- OpenID Connect
  - Authentifizierung für OAuth2
  - verwendet → JWT
  - → häufige Verwendung für SSO (single sign-on)
- JWT (JSON Web Token) → IETF RFC 7519
  - `header.payload.signature` jeweils Base64URL kodierte JSON Objekte
  - Struktur der JSON Objekte festgelegt und auch frei belegbar
  - baut auf
    - JSON Web Signature (JWS) → RFC7515
    - JSON Web Encryption (JWE) → RFC7516

Sicherheit – 3



Quelle: https://dzone.com/articles/microservices-in-practice-1