Kommunikation

Microservices

Msc. Informatik

13. April 2021

Stand der Dinge

Was geschehen wird:

- Festlegung der API
- Ausarbeitung der Kommunikationspatterns
- Entwurf der Architektur der Services

Kommunikation

Problemstellung

Services...

- werden ohne Kenntnis voneinander entwickelt
- existieren als Black Box ohne Anschluss
- benötigen Daten von anderen Services
- werden von Clients genutzt

Ziele

Anforderung an Microservices:

- Einheitliche Schnittstellen
- Erweiterbar
- Unterstützt Breaking Changes

Anforderung an das Backend als Ganzes:

- Dokumentierte Schnittstelle
- Nutzbar ohne Kenntnis des Ecosystems

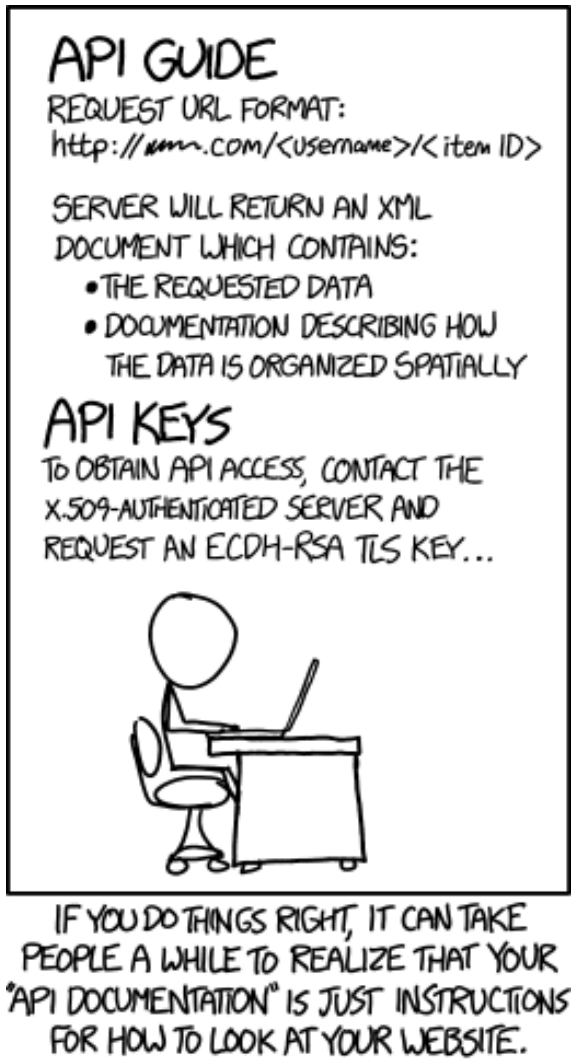


Abbildung: xkcd - api

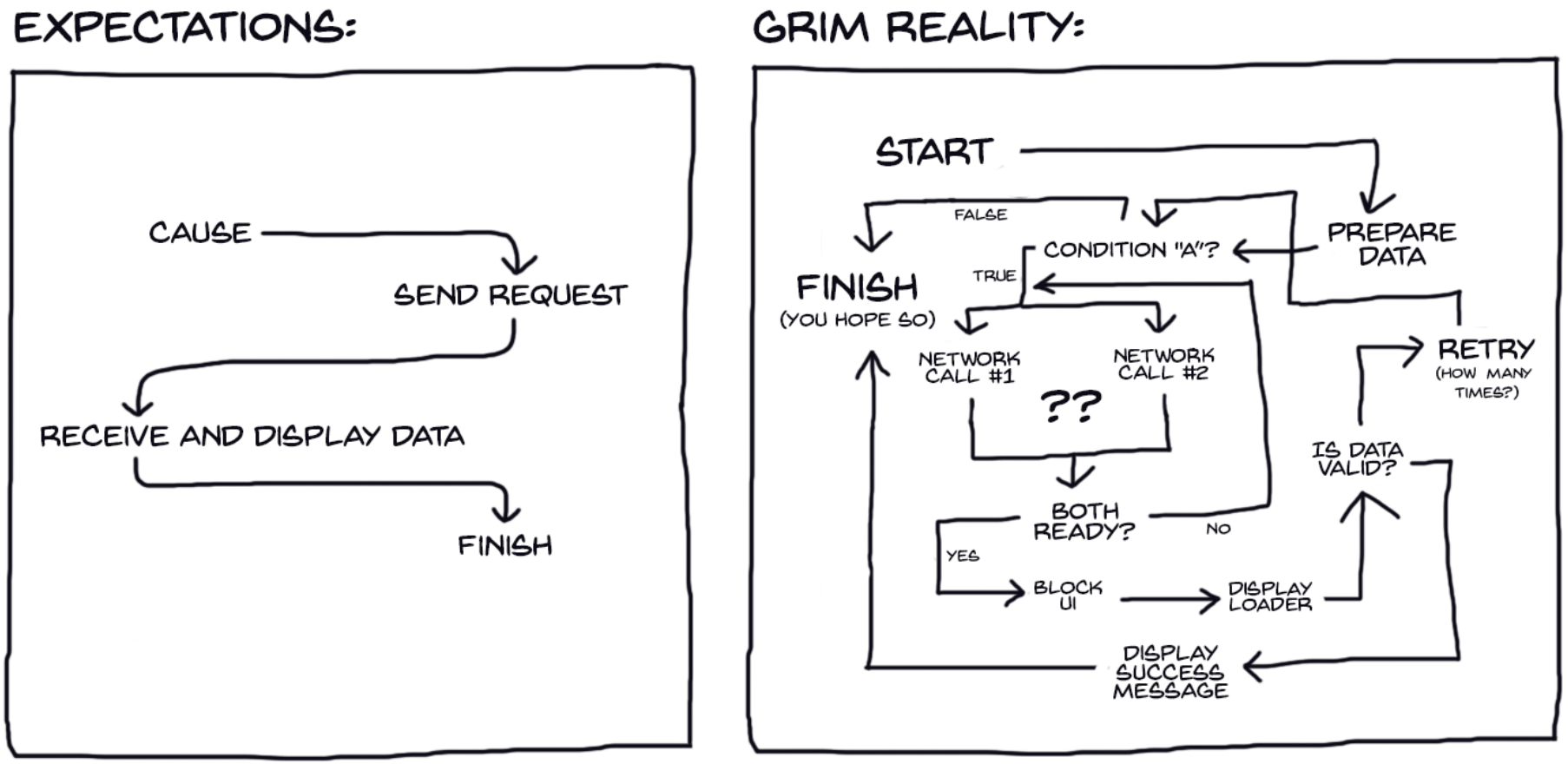


Abbildung: Expectation vs. Reality

Verschiedene Optionen

Auch bei der Kommunikation existieren viele Möglichkeiten:

- REST + HTTP Requests
- gRPC (google Remote Procedure Calls)
- Actor-System
- Custom-Protocoll

REST

Service-Typen

Was ist nun mit den REST Services in einem Microservices-System?

Alternativen

Intra-Microservices-Based-System-Communication

Es existieren valide Alternativen zu REST als Schnittstelle für die Kommunikation:

- gRPC
- Actors
- Custom Protocol

gRPC

Googles Open Source Remote Procedure Call System

- Transport geschieht über HTTP/2
- Nutzt **Protocol Buffer** als Interface Description Language
- Unterstützung für die gängigsten Sprachen
- hochperformant
- Streaming mit Bidirektion und Authentication (über HTTP/2)

Protocol Buffer

Deklarativer Ansatz, Services zu definieren

- Protocol Buffers serialisieren strukturierte Daten
- Messages mit Attributen werden definiert
- Entwickelt als schnellere und schlankere Alternative zu XML
- Vereinfacht API Versionierung, da Messages Versionsnummern enthalten
- Wird primär zwischen Services verwendet

Actors

Actors als Model für *concurrent computation*

- Nachrichtenbasierte Kommunikation
- Aktoren verarbeiten Nachrichten basierend auf ihrem Zustand
- Sind keine Threads/Prozesse, werden nur von diesen belebt
- Ressourceneffizient und ideal für Nebenläufigkeit

Custom-Protocol

Ach, einfach mal selber machen.

- Spezielle Anpassung an Use-Case
- Eventuelle Performanceoptimierung durch Reduzierung des Overheads
- Starke Kopplung zwischen Client und Frontend

In der Regel ist davon abzuraten, das Rad neu zu erfinden..

API Gateway

Wie kommunizieren Clients mit dem Microservices Ecosystem?

Anforderungen:

- Mehrere Clients sollen unterstützt werden
- Clients sollen möglichst von Microservices-Logik befreit werden
- Teams sollen ihre API unabhängig voneinander entwickeln können

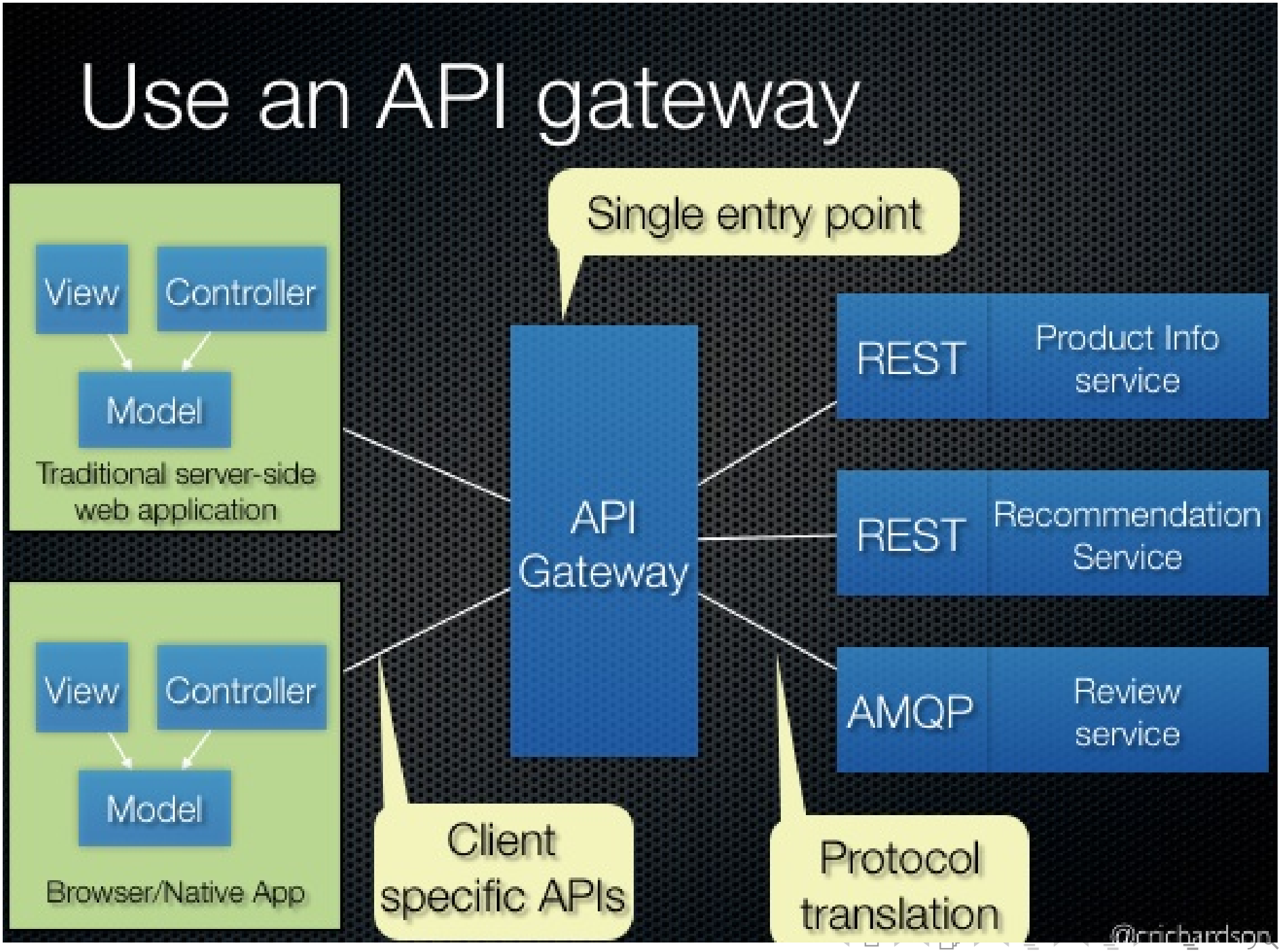
Unterschiede

Allgemein nutzbarer (Web)Dienst:

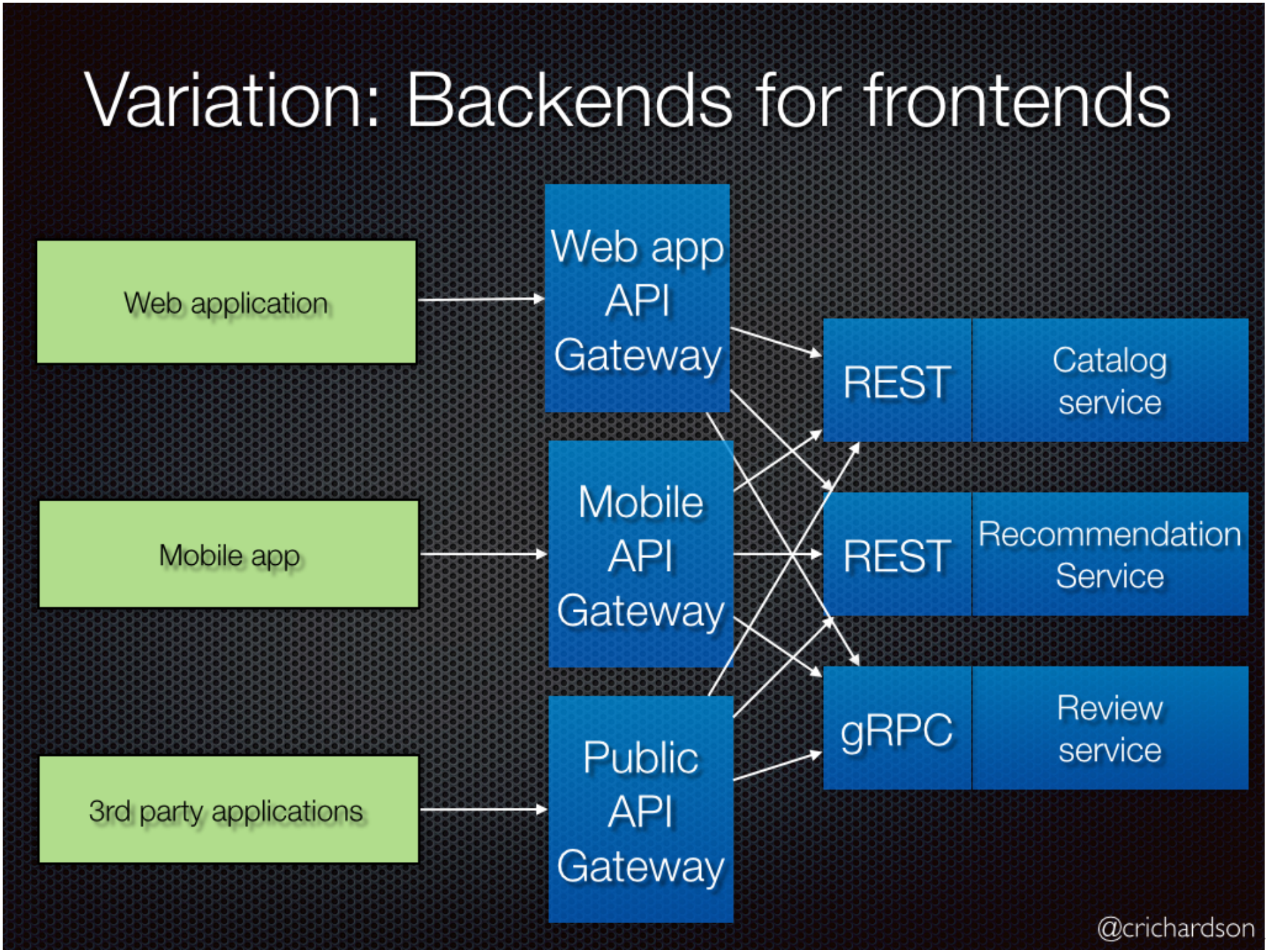
- Microservices sind extern erreichbar
- Lastverteilung muss im Backend geschehen

Starke Kopplung zwischen Client und Backend:

- Client kann Architektur des Backends kennen
  - Lokalität der Services
  - Aufteilung der Business Logic auf die Microservices
  - Erreichbarkeit/Auslastung aller Instanzen


Use an API gateway
Single entry point
View
Controller
Model
Traditional server-side web application
View
Controller
Model
Browser/Native App
API Gateway
REST
Product Info service
REST
Recommendation Service
AMQP
Review service
Client specific APIs
Protocol translation
@crichardson

Variation: Backends for frontends



API Gatway

Aufgaben des Gateways:

- Routing von Requests an den richtigen Service
- Verwalten von eventuellen User-Service-Bindungen
- Datenaggregation
- Authentication
- Letzte Bastion bei Systemausfall

Das Pattern wird beim Thema Service Mesh nochmal eine Rolle spielen