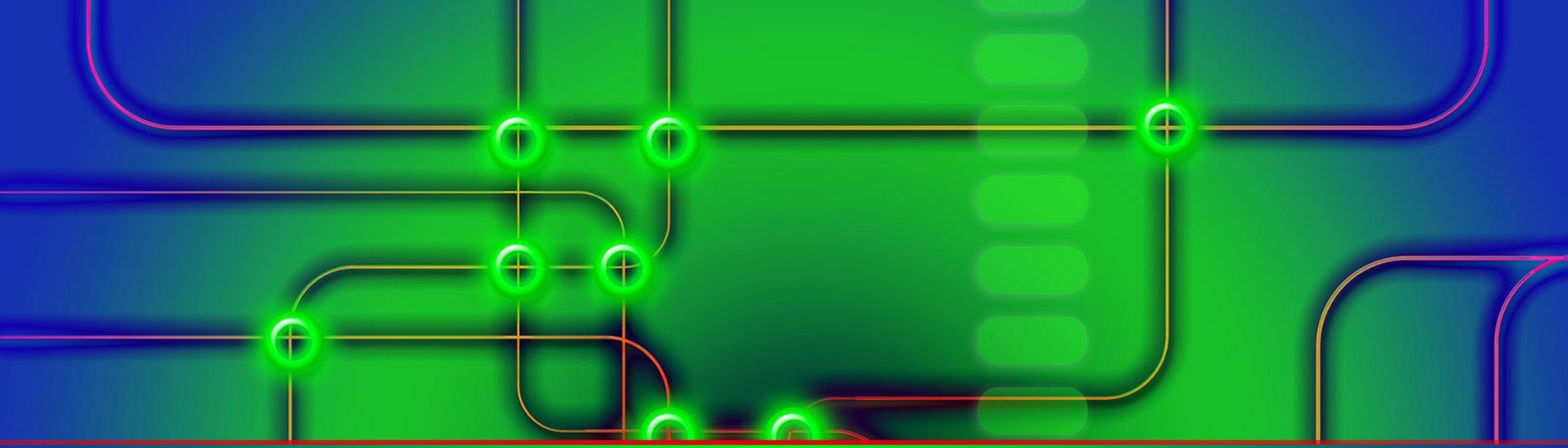
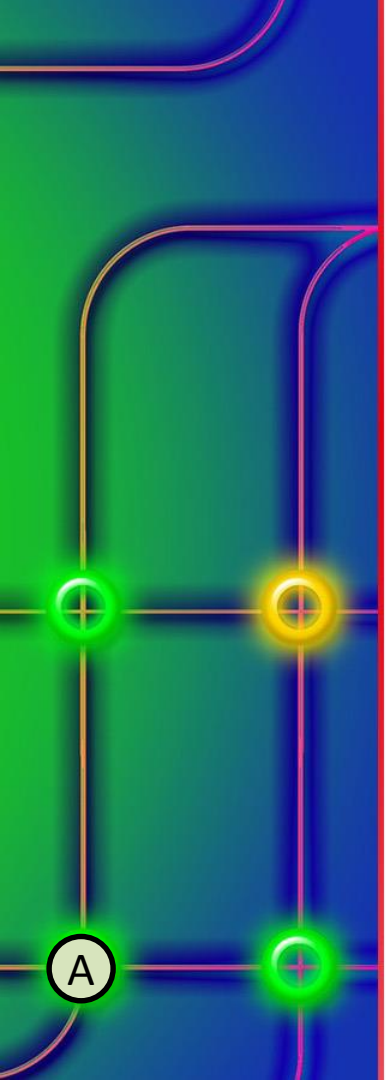


Microservices unter der Motorhaube

DEVTOPS

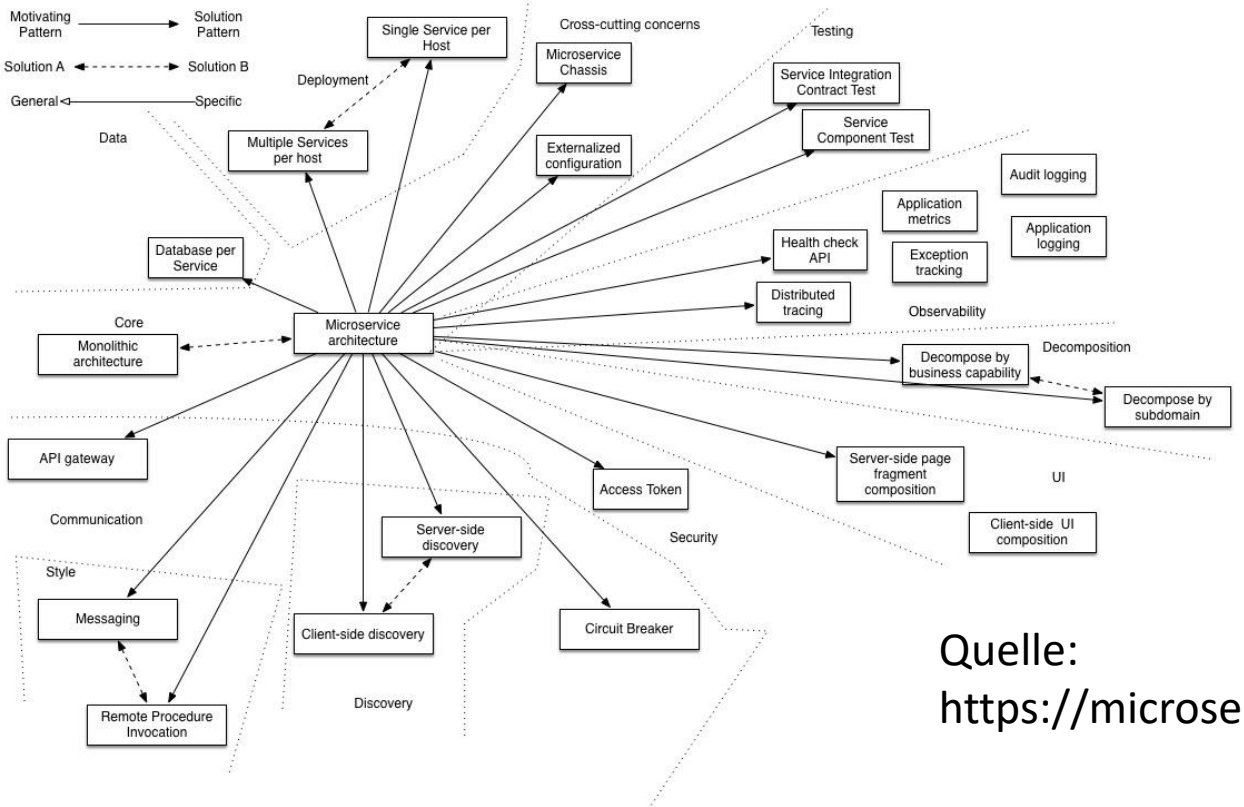
Andreas Chatziantoniou, Matthias Fuchs



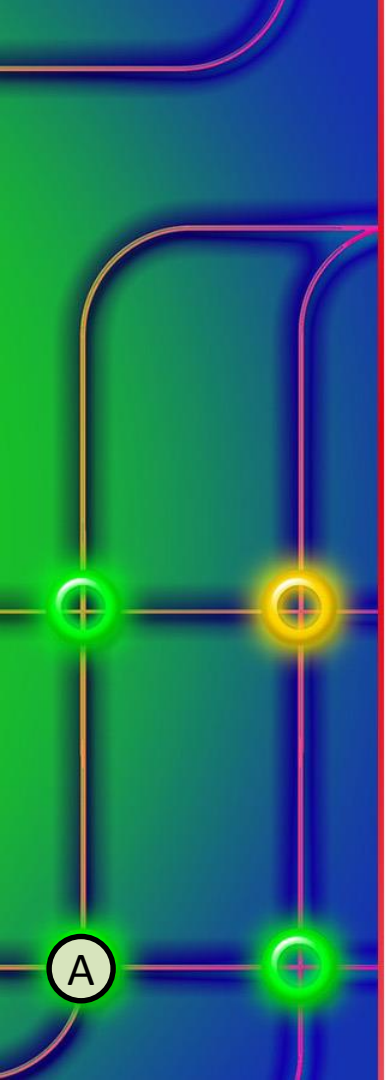
Agenda

- Microservices - Architektur
- Gegenüberstellung Microservices und SOA Suite
- Ist nun alles Gold was glänzt?

Microservices - Architektur

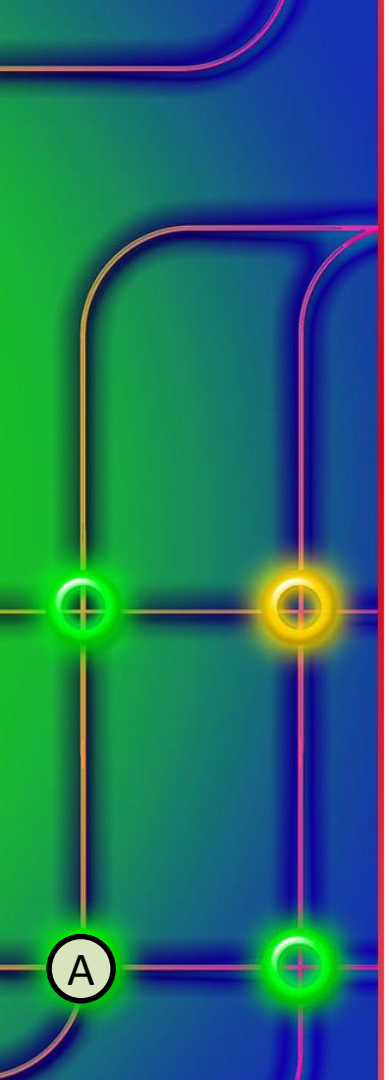


Quelle:
https://microservices.io



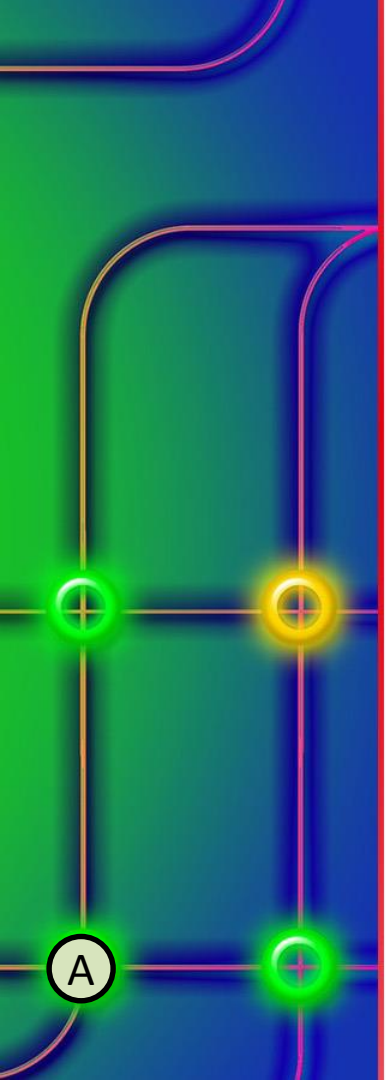
Komponenten

- Core
- Communication
- Data
- Deployment
- Testing
- Observability
- Decomposition
- UI
- Discovery
- Security
- …



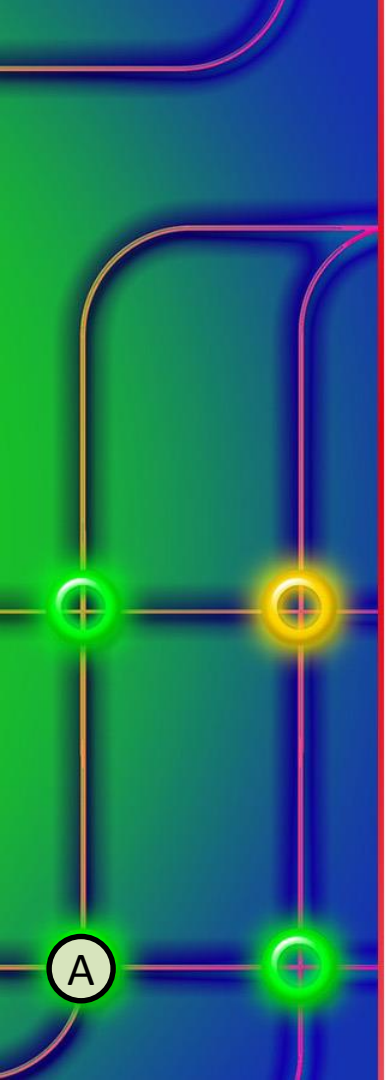
Komponenten

- Core
- **Communication**
- **Data**
- **Deployment**
- Testing
- Observability
- Decomposition
- UI
- Discovery
- **Security**
- …



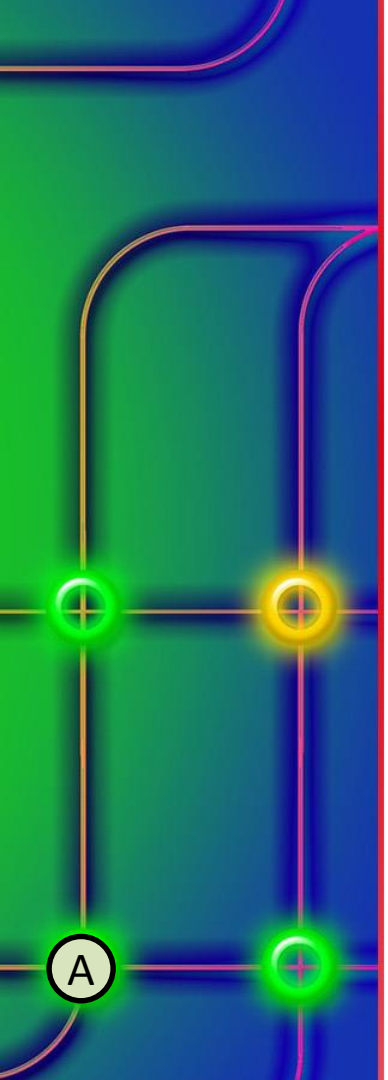
Communication (Client Access)

- In einer Microservice Architektur wird prinzipiell für jeden Client eine eigene Version des UI entwickelt
  - Browser: HTML5/JavaScript
  - Smartphone: REST Aufrufe
  - 3rd Party: REST Aufrufe



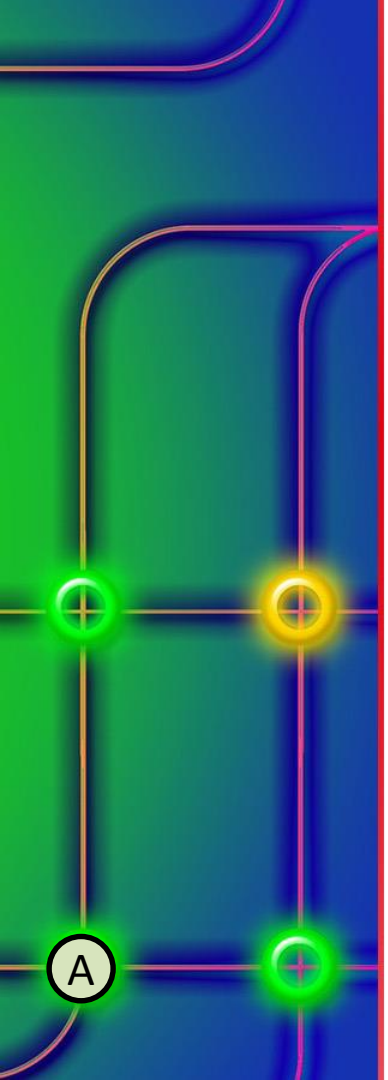
Communication (Client Access)

- Beispiel: Hotelbuchung
  - Zimmerbeschreibung
  - Verfügbarkeit
  - Preis
  - Reviews
  - Package-Deals (z.B. Restaurant + Zimmer)



Communication (Client Access)

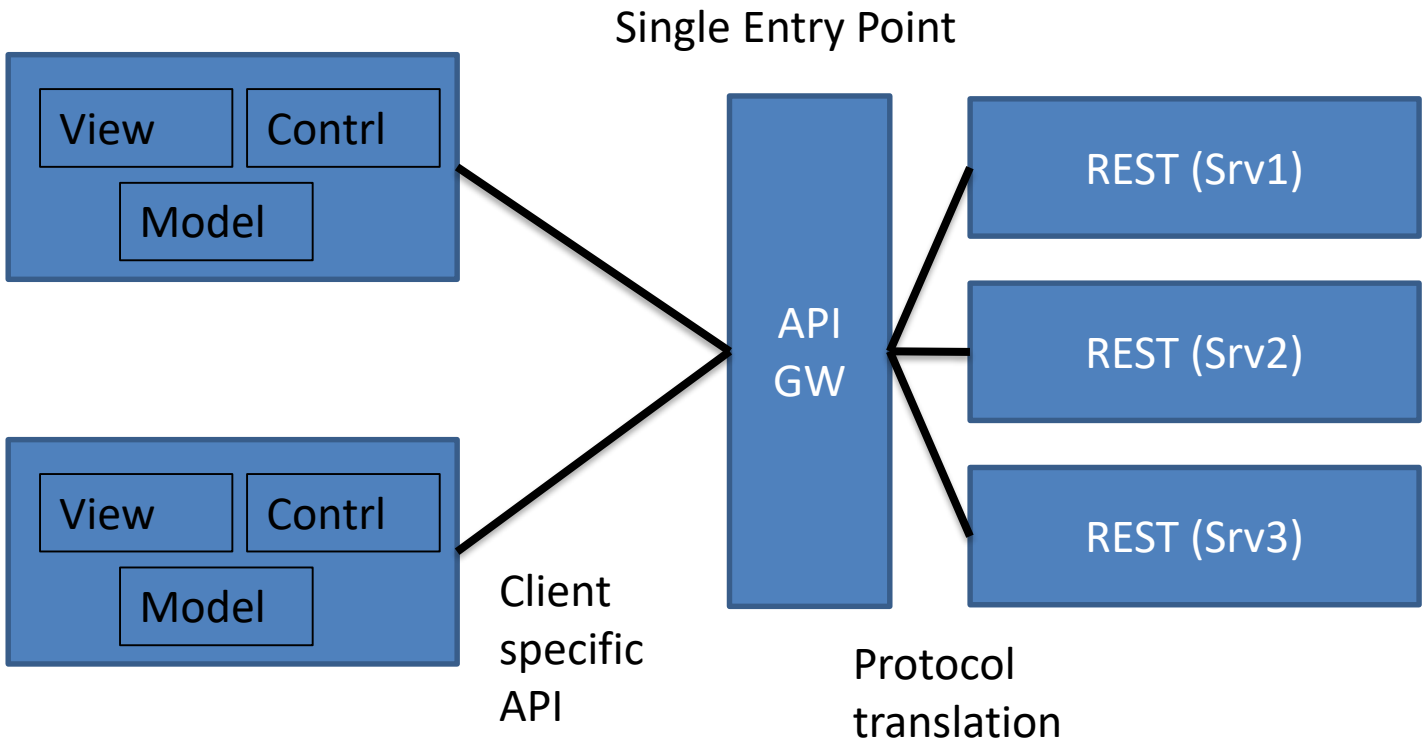
- Informationen kommen aus verschiedenen anderen Services
  - Zimmerbeschreibung → Content Mgmt
  - Verfügbarkeit → Reservierungssystem
  - Preis → AI-Engine zur Preisbestimmung
  - Reviews → Review Service / CRM
  - …

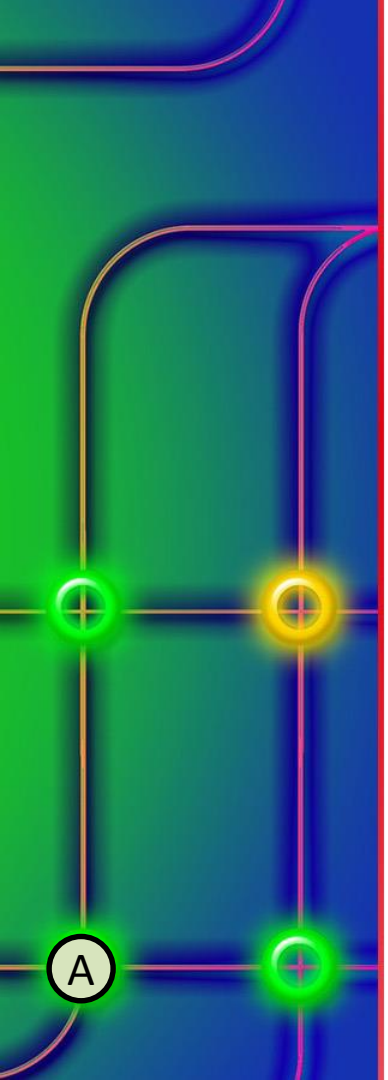


Communication (Client Access)

- Granularität der API ist feiner, als der Client dies benötigt
- Per Client werden andere Daten benötigt
- Netzwerk – Latency beeinflusst Design
- Extra Services können dazu kommen; Client sollte dies nicht merken
- Services benutzen verschiedene Protokolle

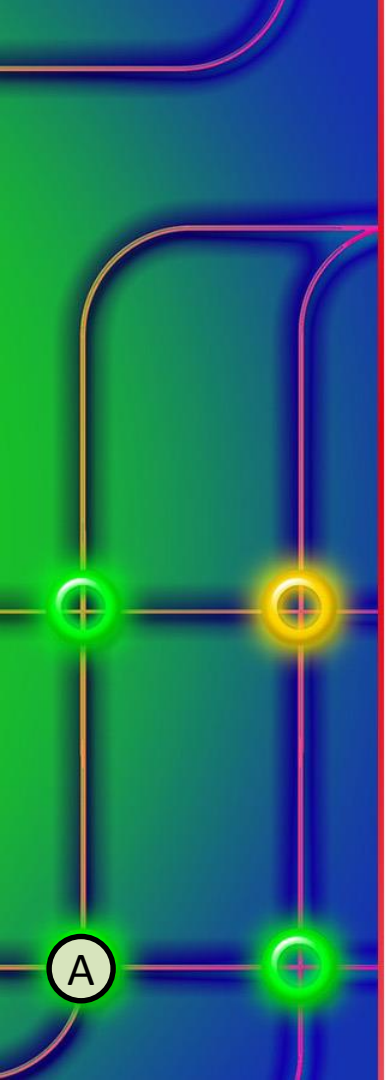
Communication (API Gateway)





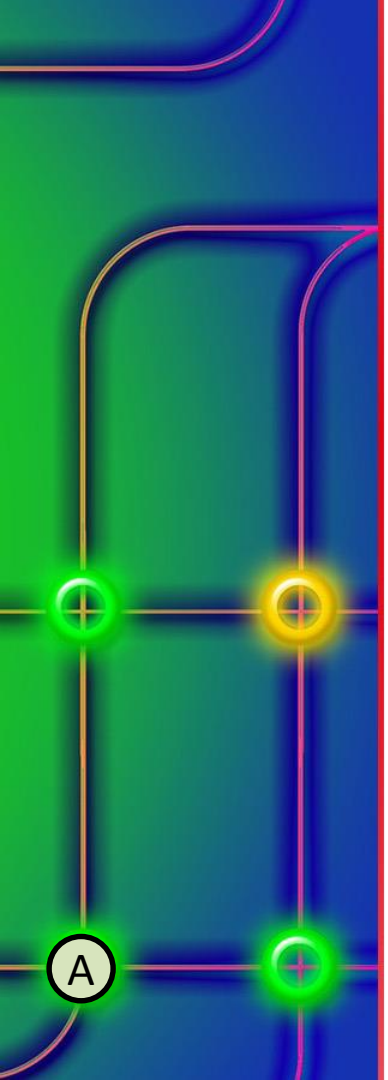
Communication (API Gateway)

- Vorteile
  - “ESB”: Client muss nicht wissen wo der Service ist
  - Verschiedene Clients werden gut unterstützt
  - “Sammelt” Calls zu verschiedenen Services
- Nachteile
  - Extra Software
  - Responsetime wird größer
  - Skalierbarkeit?



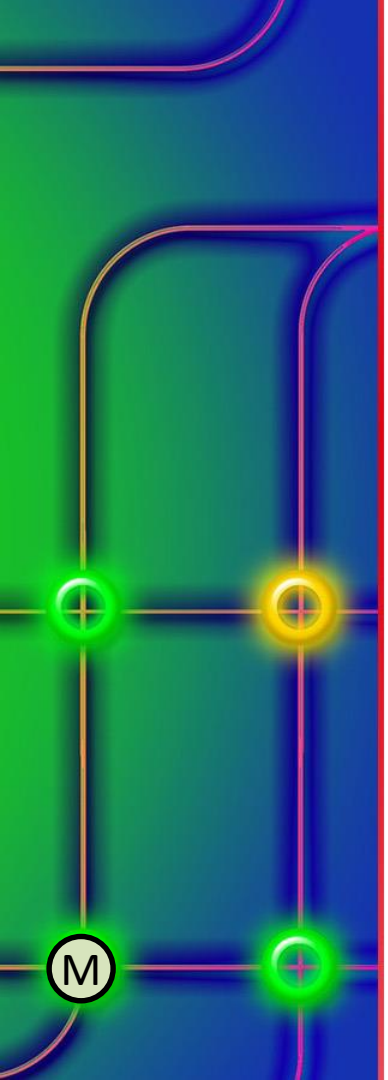
SOA - Communication

- Innerhalb der SOA wird (fast) durchgehend SOAP benutzt
- Webservices statt API
- Goldene Regel:
  - BPEL für Orchestrierung
  - ESB für Routing & Transformation



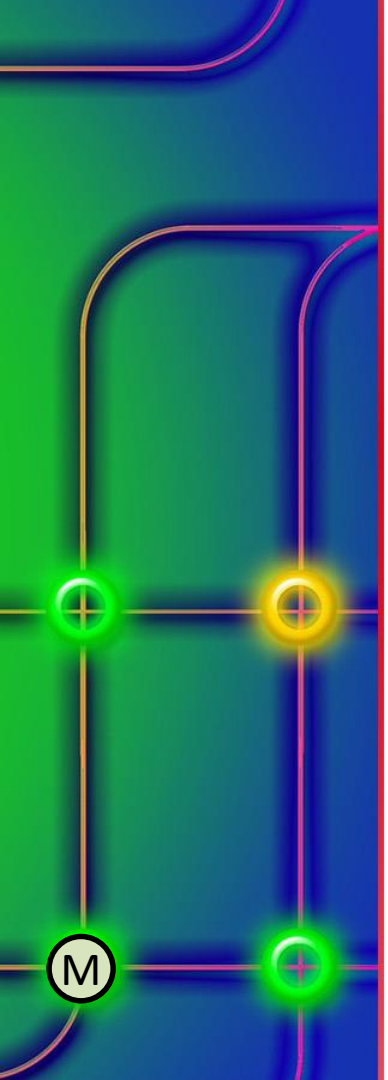
SOA / Microservices - Communication

- Orchestrierung
  - Externer Prozess regelt den Ablauf
- Choreography
  - “Service weiß, was zu tun ist, und regelt dies selbst”



Data

- Wie jedes IT System müssen auch bei Microservices Daten persistiert werden
- Wie sieht dies in Rahmen von Microservice aus?



Data - Beispiel

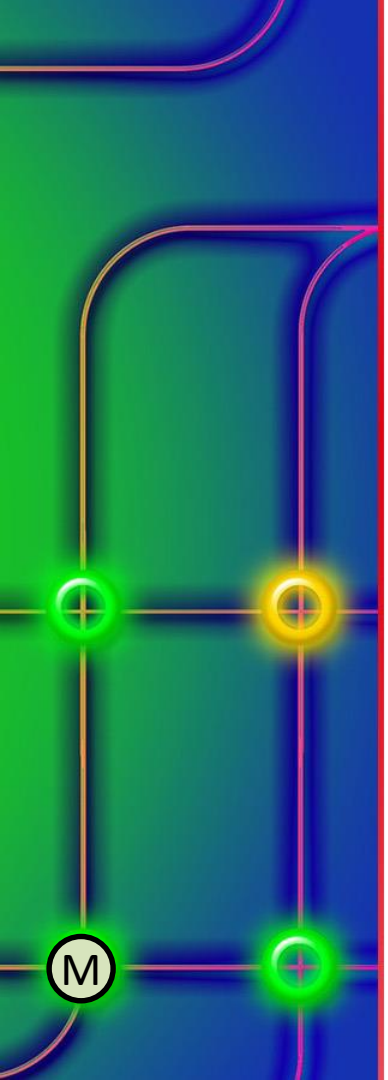
- Folgendes Beispiel:
  - Warehouse Service
    - Daten der Güter im Lager
  - Shipment Service
    - Daten der Güter von Sendungen

Data – Warehouse Table

ID	MagazineID	ProductID	NrOfSKU	LastOrderDate
1	NBG1R14	A654	4	12-NOV-2018
2	NBG1R03	B128	1099	27-AUG-2017

Data – Shipment Table

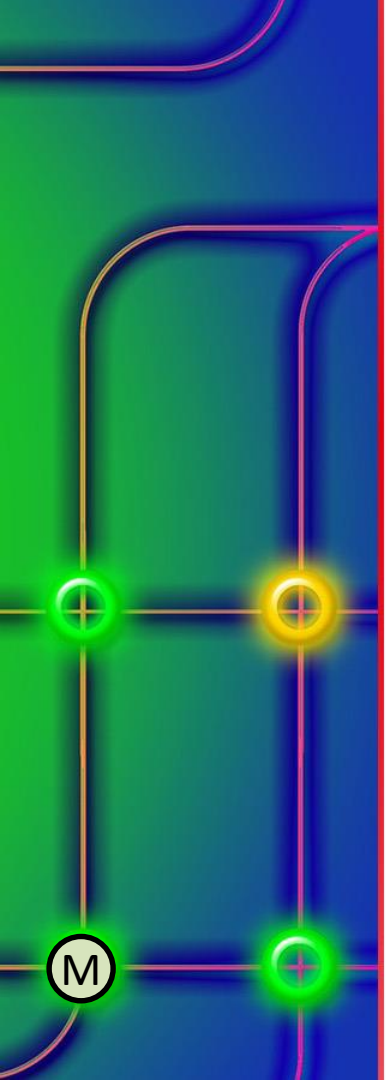
ID	ShpmtID	CstID	NrOfSKU	OrderPickID
10	23N1698	C7554	HS32817	OP553BT
11	23N1699	B421	LY5299	KE2399X



Data – Per Microservice

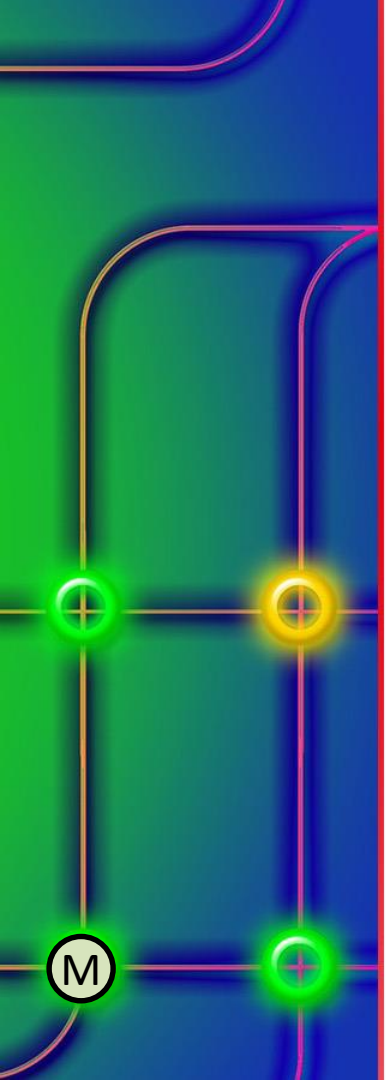
- Services sind “loosely-coupled” – welchen Einfluß hat dies auf die Daten?
- Wie kann die Architektur sicherstellen, dass die Daten konsistent sind?
- Was passiert bei Transaktionen, die sich über mehrere Services erstrecken?
- Welchem Service gehören zusammengesetzte Daten?

Data – Integrität



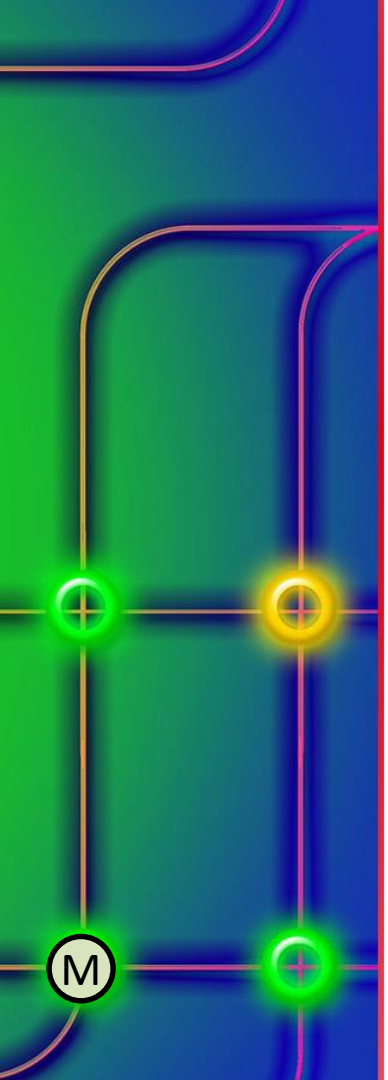
Data - Architekturlösung

- Daten bleiben einem Microservice zugeordnet
- Diese Daten sind nur über die API des Microservice zu erreichen



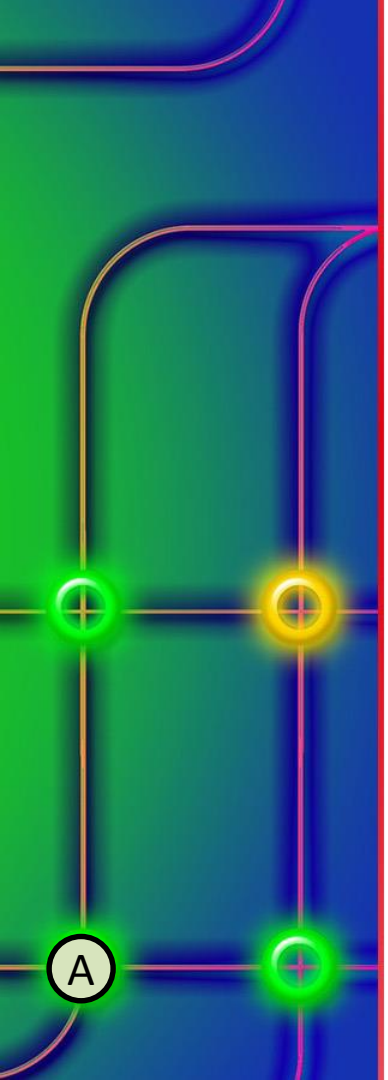
Data - Architekturlösung

- Vorteil: “loosely coupled” Microservices
  - Änderungen an einer Datenbank haben keinen Einfluß auf andere Service
  - Transactionen die mehrere (Data)Services benutzen ist komplex
- Implementierung von Queries die Daten zusammenfügen (verschiedene DBs) ist komplex
  - SAGA Pattern
    - Service publiziert Event bei Änderung, andere Services abonnieren sich



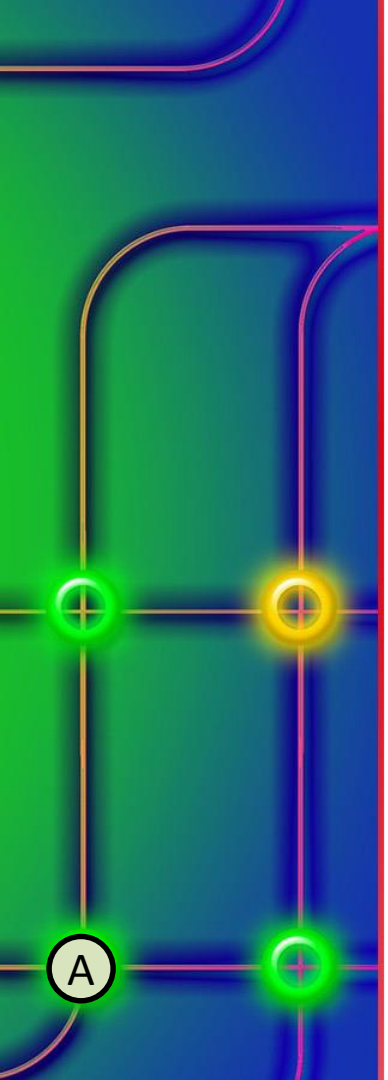
Data – SOA Suite

- DB und SOA Design können zu denselben Problemen führen



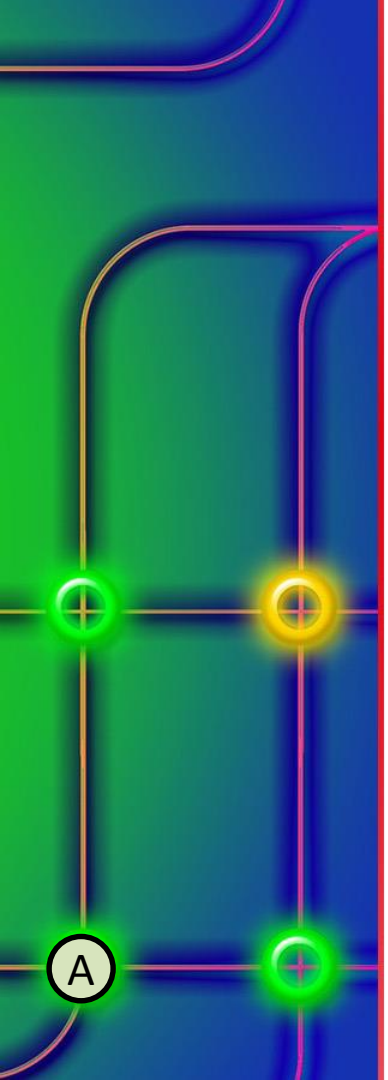
Deployment

- Mit Microservices sind diese Deployment Strategien möglich
  - Single Service Instance per host
  - Multiple Service Instance per host



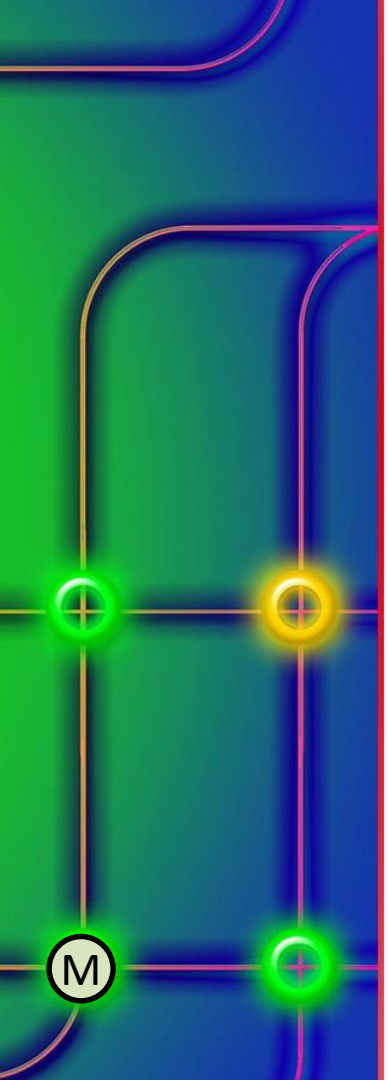
Deployment

- Beide Strategien haben dasselbe Problem:
  - Resourcemanagment
    - Zu viel Resources pro Microservice (single Instance)
    - Zu wenig Resources pro Microservice (multiple instances)
- VM/Docker/Serverless
  - Isolierung
  - Extra Management



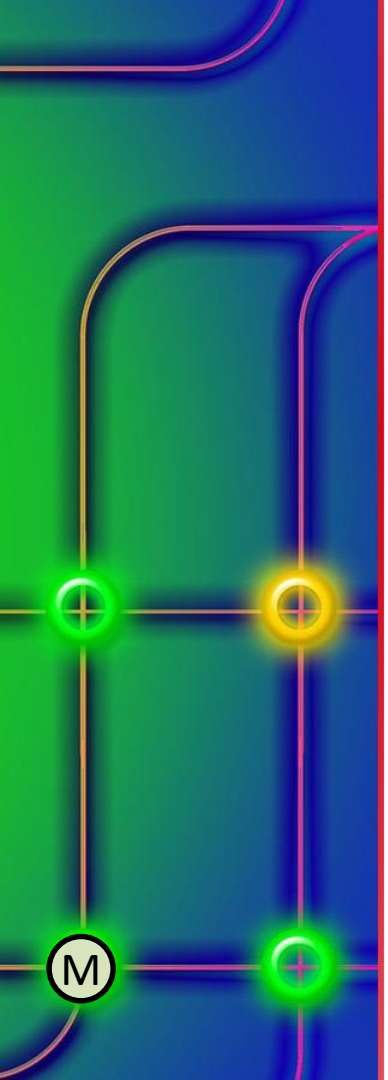
Deployment – SOA Suite

- Durch die Bindung zu Application Servers ist die Deploymentstrategie festgelegt
- Application Server
  - Resource sharing
  - Größerer Bedarf an Resources / Management



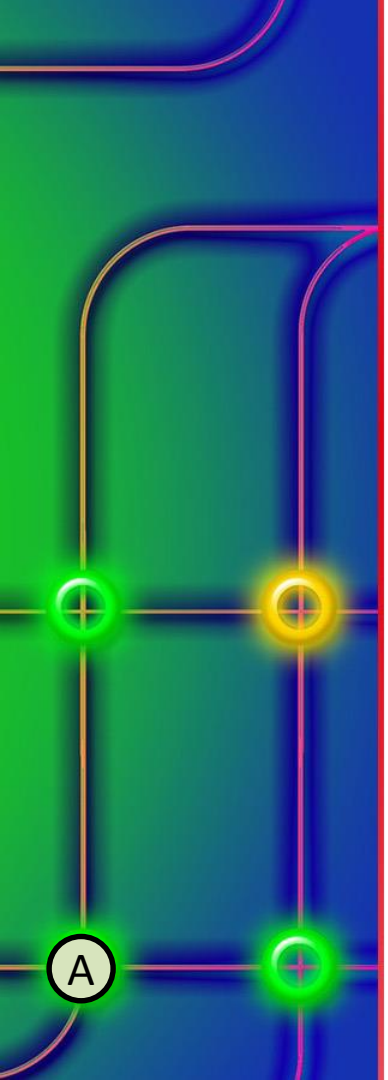
Security

- Direkte Security bei Microservices ist eher selten der Fall
  - Nutzung API Gateway
- API Gateway authentifiziert Anfragen und leitet diese and andere Services weiter
- Einsatz von JWT
- Wie verhält sich die Nutzung von vorhandener Sicherheitssoftware



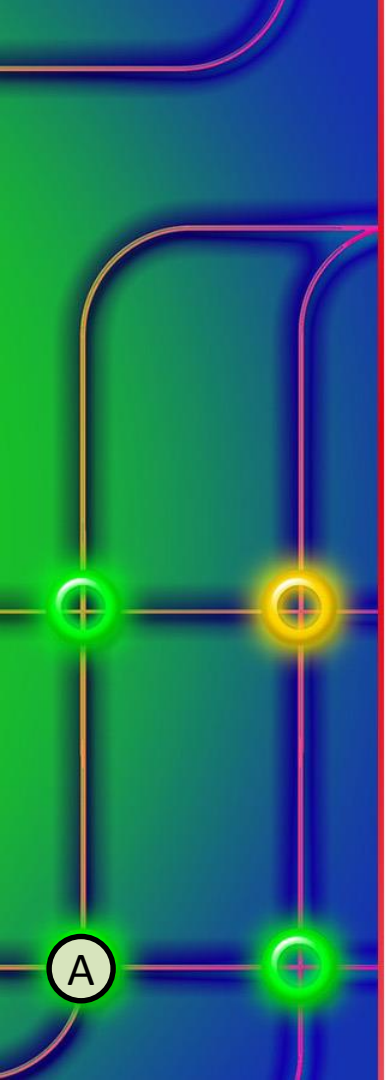
Security – SOA Suite

- In der SOA Suite wird im Bundling OWSM geliefert
- Weiterhin sind die Securityfeatures von WLS einsetzbar



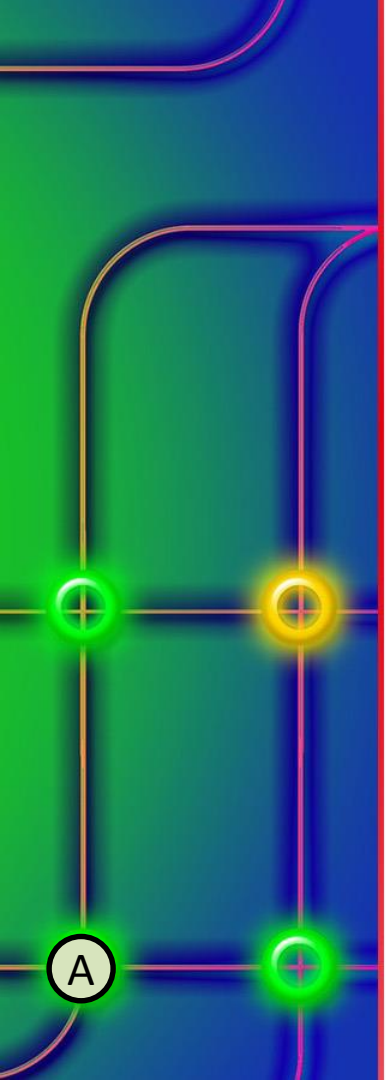
Monitoring

- Bei Microservices muss das Monitoring extra geregelt warden
- Einbindung von Frameworks ist möglich, erfordert aber Extra Aufwand



Istio

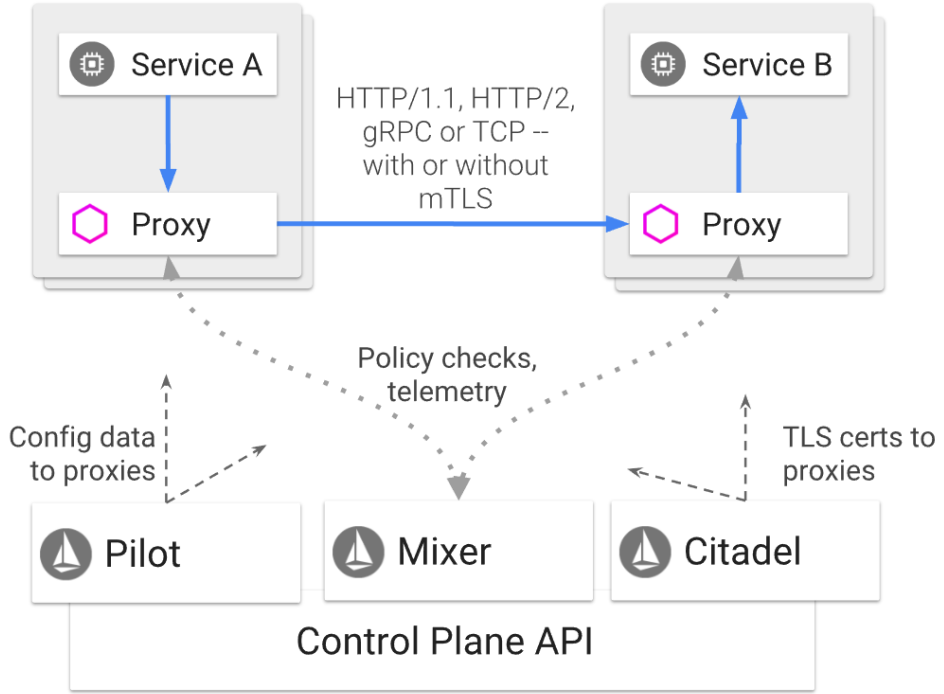
- Eine Möglichkeit um Microservices zu betreiben (monitoring, connectivity, securing) ist Istio (https://istio.io)
- Istio ist ein *service mesh*
  - Netwerk von Microservices, das Anwendungen und Integrationen darstellt
  - Wenn ein service mesh wächst (Anzahl und Komplexität) wird es schwieriger um es zu managen
  - Ein *service mesh* hat oft komplexe operationelle Anforderungen
    - A/B testing, canary releases, rate limiting, access control und end-to-end authentication



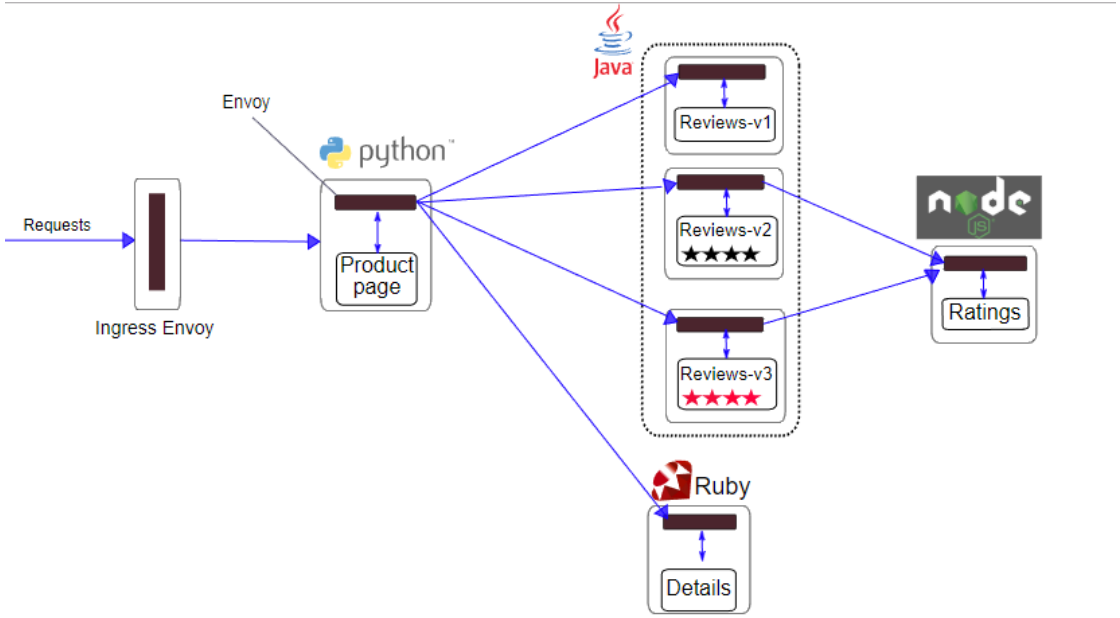
Was bietet Istio?

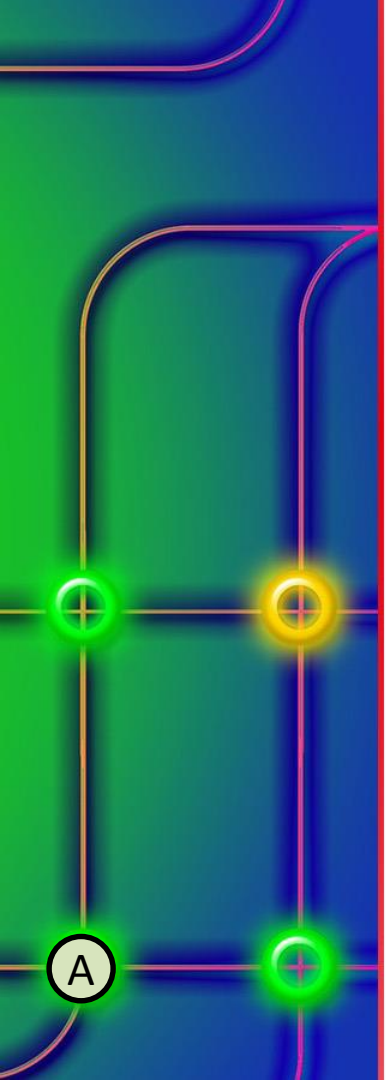
- Istio erlaubt das Anlegen eines Netzwerks von deployed Microservices mit load balancing, service-to-service authentication, monitoring und mehr, ohne Änderungen im Microservice Code
- Istio wird zu Microservices hinzugefügt in dem ein *sidecar proxy* in der Umgebung ausgerollt wird
  - Dieser *sidecar* fängt die Netzwerkkommunikation zwischen den Microservices ab
  - Danach kann Istio mit eine *Control Plane* die Umgebung betreiben

Istio - Architektur



Istio - Beispiel

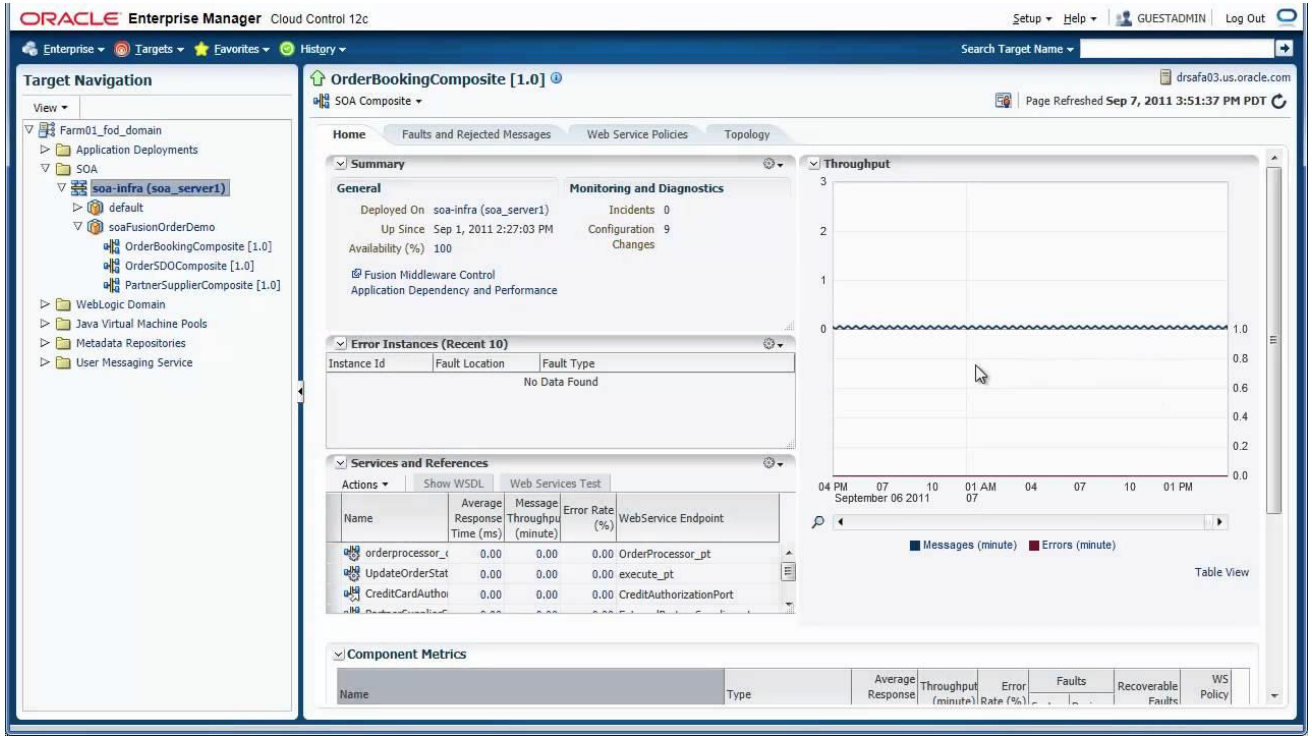




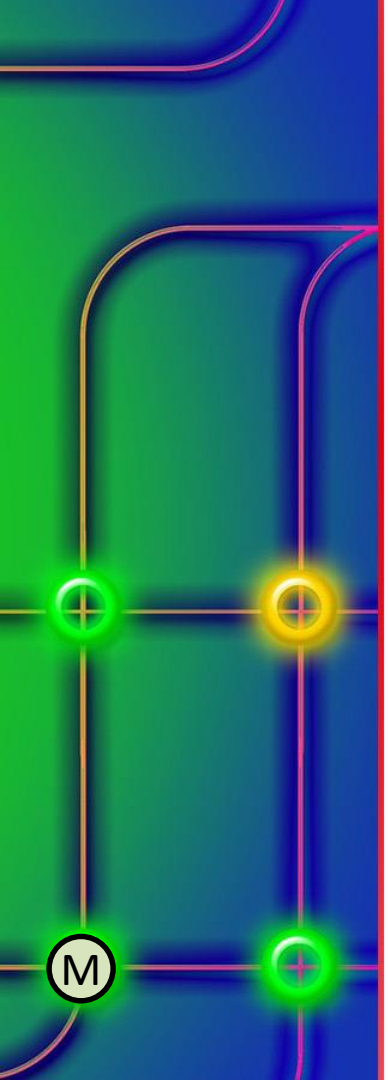
SOA Suite – Cloud Control

- In eine non-Microservices Oracle FMW Umgebung (z.B. SOA Suite) würden wir hierzu Oracle Cloud Control (inkl. Mgmt Packs) einsetzen

SOA Suite – Cloud Control

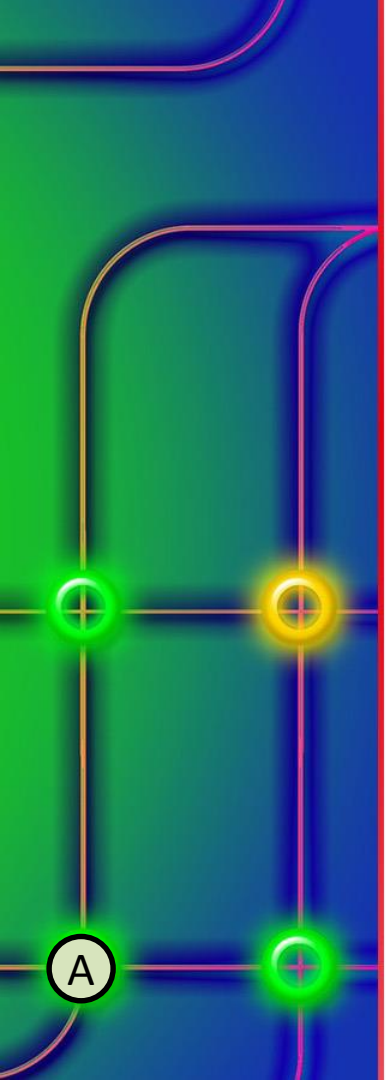


Ist nun alles Gold was glänzt?



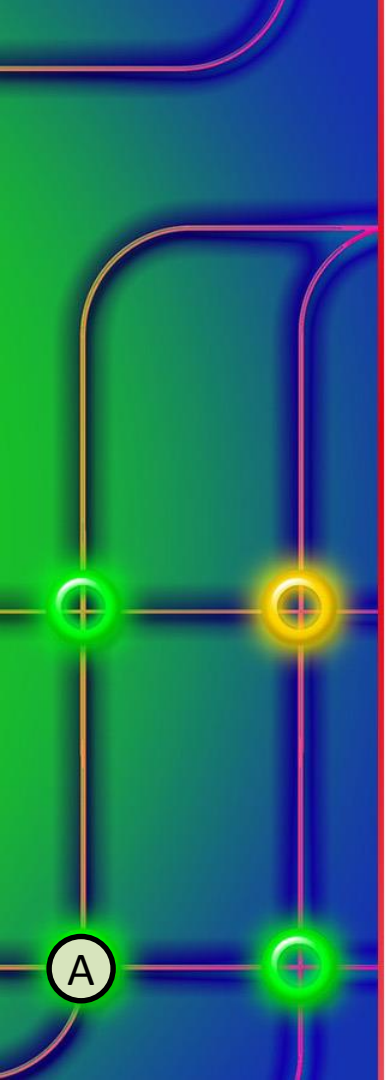
Vorteile Microservices

- (erfordert) Konvention für gutes SW Design
- Höhere Produktivität
- Unabhängig (Entw/Deploy/Patch/…)
- Gute Integration mit CI Tools (Hudson/Jenkins/bamboo)
- Gute Wartbarkeit, weil klein und übersichtlich



Nachteile Microservices

- Eigentlich nur möglich mit DevOps
- Verteilte Systeme sind schwieriger zu managen (auch im Monitoring)
- Security ist aufwendig und resourcen-intensive
- Troubleshooting ist schwierig
- Performance (?)
- End-to-end Tests sind komplex



Fazit

- Microservices können sehr sinnvoll genutzt werden
- Beachten der Rahmenbedingungen
  - Übersichtlichkeit/Design/Performance/Security/…
- Microservices werden sehr wahrscheinlich zum Problem, wenn die Organisation nicht Bereit zum Umdenken ist