MINUTEN

CLOUD-NATIVE

CLOUD NATIVE COMPUTING

Unit:

Microservices

(1) Vor- und Nachteile

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 12

Microservice Architekturen

12.1 Eigenschaften von Microservices

12.2 Integrationsmuster für Microservices

- Datenbank-basierte Integration
- gRPC, REST
- API Versioning
- Event-Driven Architectures

12.3 Architekturelle Sicherheit

- Circuit-Breaker, Bulkhead
- Idempotente API-Operationen

12.4 Skalierung von Microservices

- Load Balancing
- Messaging
- Scaling for Reads/Writes
- Command Query Responsibility Segregation (CQRS)
- Caching

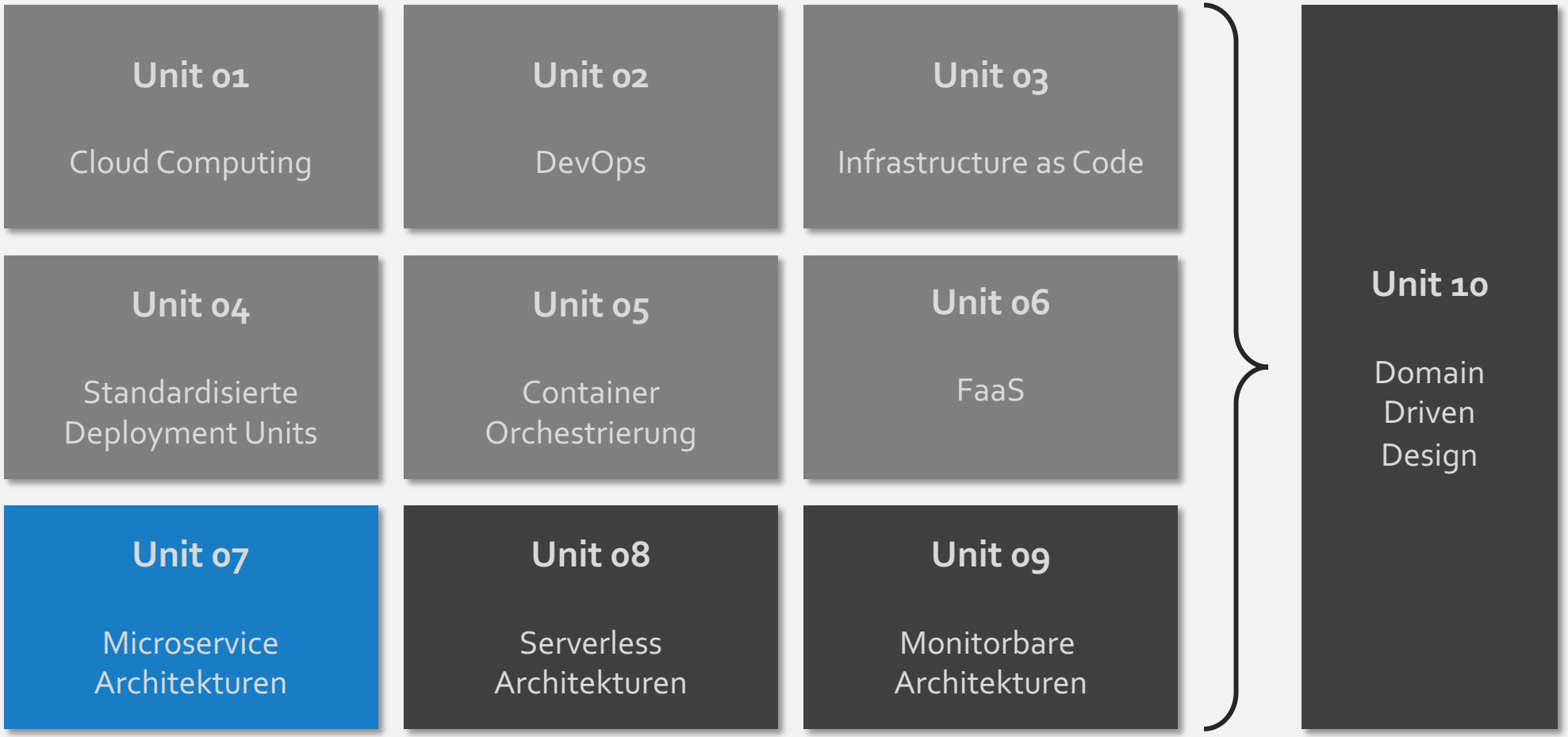
12.5 Prinzipien zur Entwicklung von Microservices

12.6 Serverless-Architekturen

- Architekturelle Konsequenzen von Serverless-Limitierungen
- Das API-Gateway Pattern
- Abgrenzung zu Microservices

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

Microservices

- Was ist das für ein Architekturstil?
- Vor- und Nachteile von Microservices

Randbedingungen

- Lose Kopplung und hohe Kohäsion
- Bounded Context und Domain Driven Design
- Conways Law
- Service Ownership und Team-Strukturen

Integration

- Shared Databases, Sync vs. Async, RPC
- REST, Services as State Machines (HATEOS)
- Versioning

Skalierung von Microservices

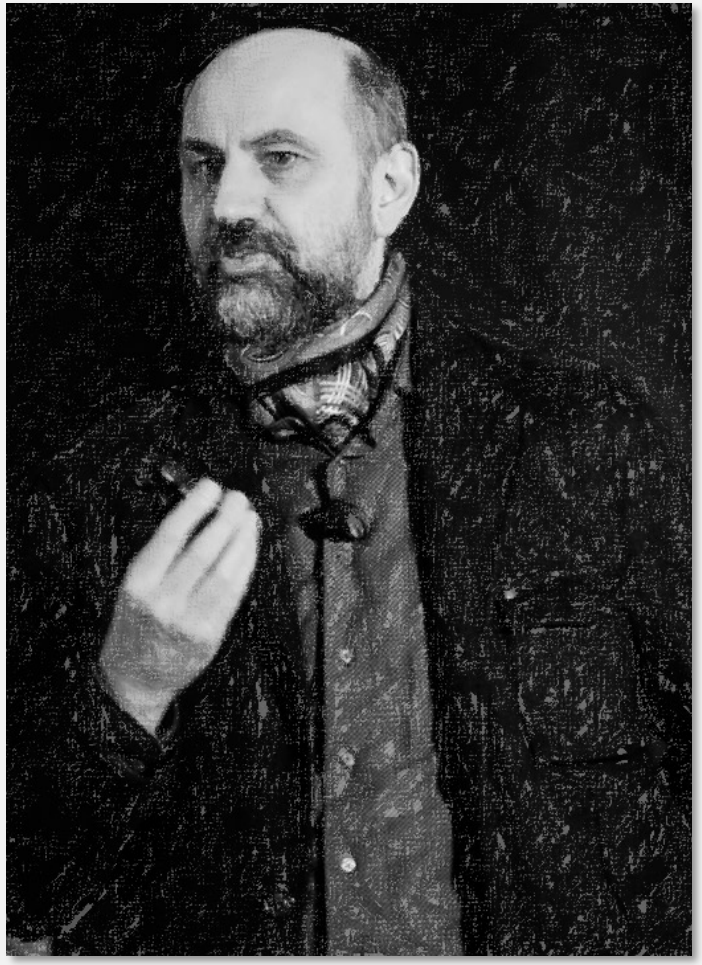
- Failure is Everywhere
- Architectural Safety Measures
- Scaling
- Caching

7 Prinzipien von Microservices

- Model Around Business Concepts
- Adopt a Culture of Automation
- Hide Internal Implementation Details
- Decentralize All the Things
- Independently Deployable
- Isolate Failure
- Highly Observable

WAS SIND MICROSERVICES?

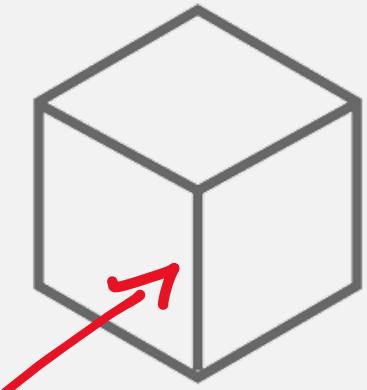
- Microservice **architectural style** is an approach to developing a single application
- Application is **composed of small services**, each running in its own process
- Services communicate with lightweight mechanisms, often an **HTTP resource API**
- Services are built around business capabilities and are **independently deployable**
- **Minimal centralized** management
- Services are written in different programming languages and use different data storage technologies
  - **Polyglot programming**
  - **Polyglot persistency**



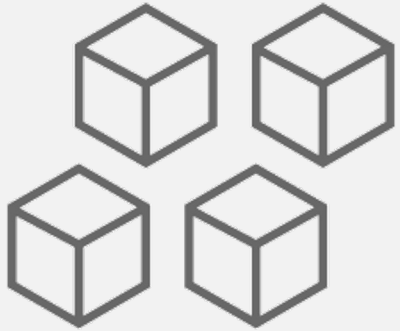
Martin Fowler, 2014

Quelle: https://martinfowler.com/articles/microservices.html

MONOLITHEN ⇢ MICROSERVICES



MONOLITHIC
Single unit



SOA
Coarse-grained

MICROSERVICES
Fine-grained

Eine monolithische Anwendung führt ihre gesamte Funktionalität **in einem einzigen Prozess** aus und skaliert diesen durch Replikation des Monolithen auf mehrere Server.

Bei einer Microservices-Architektur wird jedes Funktionselement **in einem separaten Service (Prozess)** untergebracht und skaliert, indem diese Services auf verschiedene Server verteilt und bei Bedarf **unabhängig voneinander repliziert** werden.

Quelle: https://martinfowler.com/articles/microservices.html

DEKOMPOSITION MITTELS DIENSTEN

- Eine Komponente ist eine Einheit von Software, die **unabhängig voneinander austauschbar** und aufrüstbar ist.
- Microservice-Architekturen verwenden Bibliotheken, aber ihre eigene Software wird in erster Linie durch die **Aufteilung in Services** vorgenommen.
- Bibliotheken sind Komponenten, die in ein Programm eingebunden sind und über **in-memory Funktionsaufrufe** aufgerufen werden.
- Services sind **out-of-process Komponenten**, die über einen Mechanismus wie eine HTTP-Requests oder Remote Procedure Calls kommunizieren.

Quelle: https://martinfowler.com/articles/microservices.html

CLOUD NATIVE COMPUTING

Merke:

Komponenten:
Unabhängig austauschbar und aktualisierbar.

Libraries:
In-Process Komponenten

Services:
Out-of-Process Komponenten

DEKOMPOSITION MITTELS DIENSTEN

Vor- und Nachteile

Vorteile:

- Services als Komponenten führen zu expliziteren Komponentenschnittstellen.
- Programmiersprachen verfügen oft nicht über einen Mechanismus zur Definition explizit veröffentlichter Schnittstellen.
- Oft verhindern nur Dokumentation und Disziplin, dass die Kapselung einer Komponente aufgebrochen wird.
- Dies kann zu engen Kopplung zwischen Komponenten führen.
- Services erfordern hingegen explizite Remote-Call Mechanismen und sind daher disziplinierend.

Nachteile:

- Remote-Calls sind „teurer" als prozessinterne Aufrufe.
- Daher sind solche APIs meist weniger feingranular und somit umständlicher zu verwenden.
- Änderung von Zuständigkeiten zwischen Komponenten ist komplexer.
- Komponenten-übergreifende Anpassungen sind oft schwieriger umzusetzen.
- Es müssen bei der Interaktionen zwischen Komponenten Prozessgrenzen überschritten werden.

Quelle: https://martinfowler.com/articles/microservices.html

DEKOMPOSITION MITTELS DIENSTEN

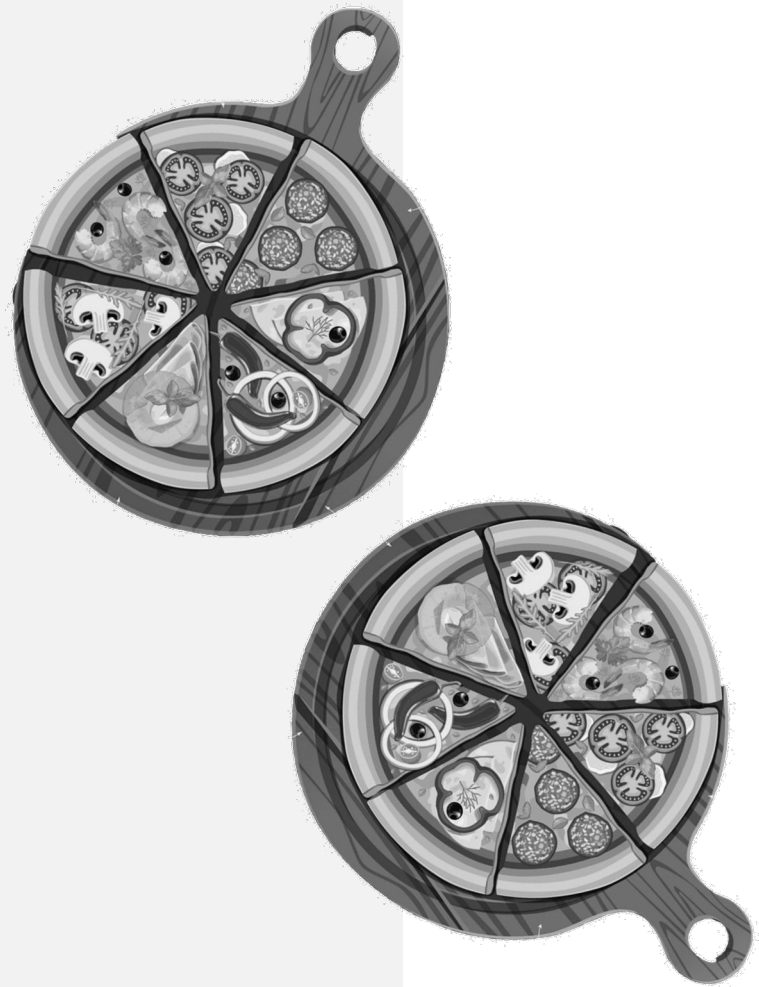
Wie groß ist ein Microservice?

- Obwohl "Microservice" zu einem beliebten Namen für diesen Architekturstil geworden ist, führt der Name zu einer unglücklichen Fokussierung auf die Größe des Dienstes und zu Diskussionen darüber, was "Mikro" ausmacht.

- In der Praxis findet man eine **Bandbreite von Servicegrößen**.

  - Am oberen Ende findet man etwa Amazons Konzept des **Zwei-Pizza-Teams** (d. h. ein gesamte Team kann von zwei *amerikanischen* Pizzen ernährt werden), also nicht mehr als ein Dutzend Personen.

  - Am unteren Ende der Skala findet man Teams von einem halben Dutzend Personen, die etwa ein halbes Dutzend Services unterstützen (**1 Personen "Teams"**).



Quelle: https://martinfowler.com/articles/microservices.html

DEKOMPOSITION MITTELS DIENSTEN

Produkte anstatt von Projekten

- Anwendungsentwicklungen basieren oft auf einem Projektmodell
- Anders beim Microservices-Modell: Ein Team betreut einen Service im Sinne eines Produkts über dessen gesamte Lebensdauer
- Entwickler haben so mehr Kontakt zu Benutzern und sammeln Erfahrung aus dem Betrieb (bspw. im Rahmen ihrer Support-Aufgaben)
- Fokus auf Geschäftsfunktionen fördert Software, die Benutzern bei der Verbesserung der Geschäftsfähigkeit unterstützt

Quelle: https://martinfowler.com/articles/microservices.html

CLOUD NATIVE COMPUTING

Software wird gerne in Projekten entwickelt (ähnlich einem Haus).

Verstehen Sie Services lieber als Produkte, die dauerhaft gewartet werden müssen (eher das Selbstverständnis eines Kathedralen-Baumeisters).

“ You build it, you run it.

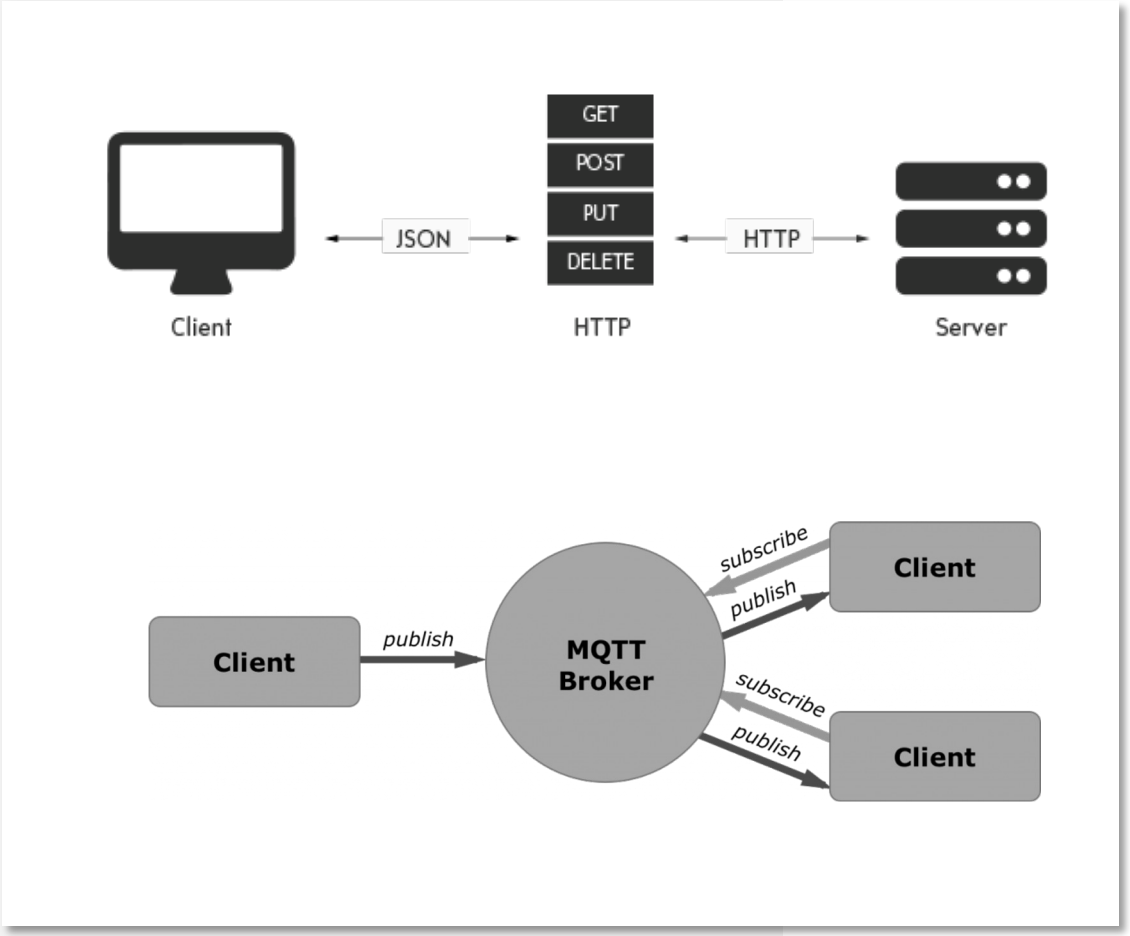
- Werner Vogels (CTO, Amazon)

DEKOMPOSITION MITTELS DIENSTEN

CLOUD NATIVE COMPUTING

Smart endpoints + Dumb pipes

- Fokus auf **Kommunikationsmechanismus** bei Aufbau von Kommunikationsstrukturen zwischen Prozessen
- Beispiel: Enterprise Service Bus (ESB) Technologien enthalten oft umfangreiche Funktionen für Nachrichten-Routing, Choreografie, Transformation und Anwendung von Geschäftsregeln
- Microservices sollen hingegen möglichst **entkoppelt** und **kohärent** sein
- Microservices agieren eher als kombinierbare **Filter** (im Sinne der Unix-Philosophie) und besitzen eigene **Domänenlogik**
- Choreografie erfolgt oft über einfache **REST-Protokolle**
- **HTTP Request-Response**-basierte Ressourcen-APIs und **Lightweight Messaging** sind am häufigsten verwendete Microservices-Protokolle



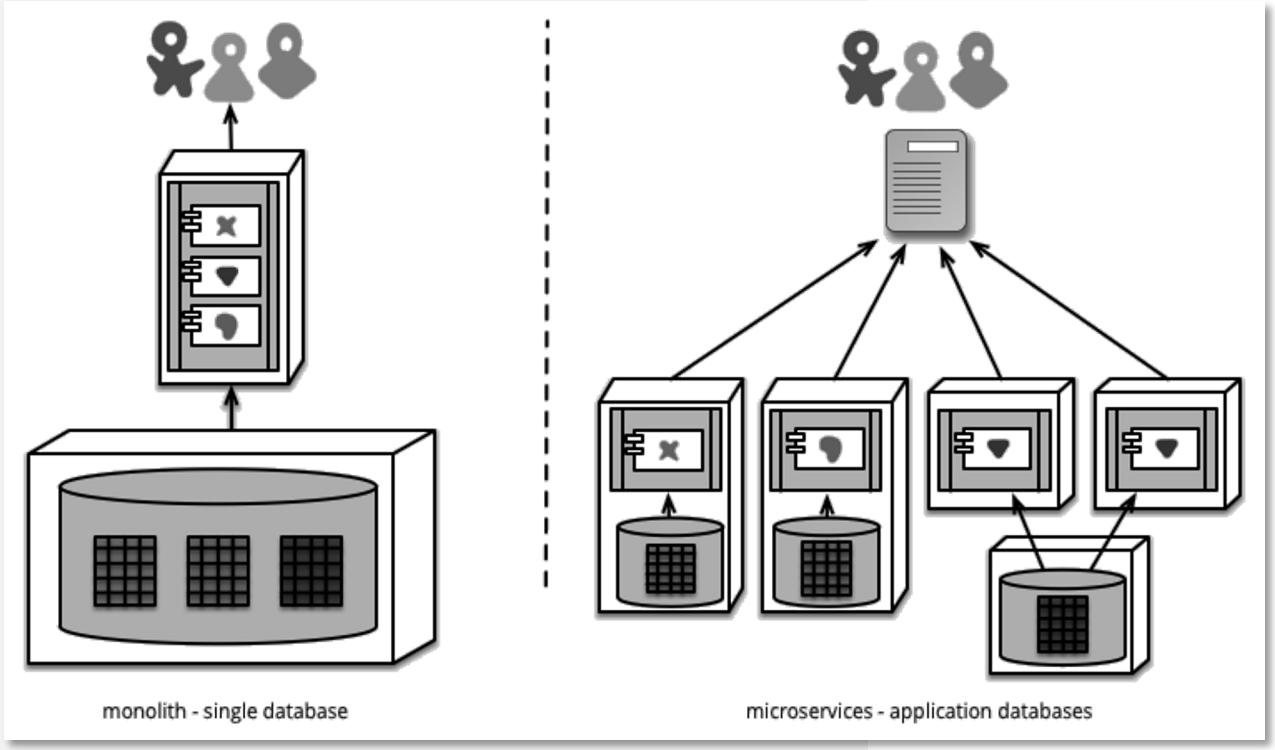
Quelle: https://martinfowler.com/articles/microservices.html

DEKOMPOSITION MITTELS DIENSTEN

CLOUD NATIVE COMPUTING

Dezentralisierte / Polyglotte Persistenz

- Konzeptionelle Modelle der Welt unterscheiden sich zwischen Komponenten
- Monolithische Systeme machen oft Kompromisse und binden sich an eine DB-Technologie
- Microservices dezentralisieren hingegen oft Entscheidungen über Datenspeicherung
- **Polyglotte Persistenz** bedeutet (im Extremfall) jeder Service hat seine eigene Datenbank
- Polyglotte Persistenz erhöht allerdings auch die Komplexität und erfordert technisches Know-how (über mehr DB-Systeme)
- Microservice Architekturen kommen daher oft mit Herausforderungen aufgrund der polyglotten Persistenz einher



Quellen: https://martinfowler.com/articles/microservices.html, https://martinfowler.com/bliki/PolyglotPersistence.html

DEKOMPOSITION MITTELS DIENSTEN

Evolutionäres Design

- Microservice-Architekturen folgen einer **evolutionären Design-Philosophie**.
- Die Schlüsseleigenschaft von Komponenten sollte deren **unabhängigen Austauschbarkeit** und Erweiterbarkeit sein.
- Komponenten sollten einem **modularen Design** folgen. Hierbei sollte Funktionalität, die gleichzeitig geändert werden muss, im selben Modul bleiben.
- Bei einem Monolithen erfordert jede Änderung eine vollständige Erstellung und Bereitstellung der gesamten Anwendung.
- Bei Microservices muss man nur den/die **geänderten Service(s) neu bereitstellen**.
- Der Nachteil besteht darin, dass zu beachten ist, dass Änderungen an einem Service dessen Nutzer (Consuming Services) beeinträchtigen können.

Quellen: https://martinfowler.com/articles/microservices.html

INHALTE

CLOUD NATIVE COMPUTING

Microservices

- Was ist das für ein Architekturstil?
- Vor- und Nachteile von Microservices

Randbedingungen

- Lose Kopplung und hohe Kohäsion
- Bounded Context und Domain Driven Design
- Conways Law
- Service Ownership und Team-Strukturen

Integration

- Shared Databases, Sync vs. Async, RPC
- REST, Services as State Machines (HATEOS)
- Versioning

Skalierung von Microservices

- Failure is Everywhere
- Architectural Safety Measures
- Scaling
- Caching

7 Prinzipien von Microservices

- Model Around Business Concepts
- Adopt a Culture of Automation
- Hide Internal Implementation Details
- Decentralize All the Things
- Independently Deployable
- Isolate Failure
- Highly Observable

KONTAKT

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de