ABB Technikerschule

Technik. Informatik. Wirtschaft. Management →

Dominik Meyer
Software Engineering

(Container) Architektur

Ziel

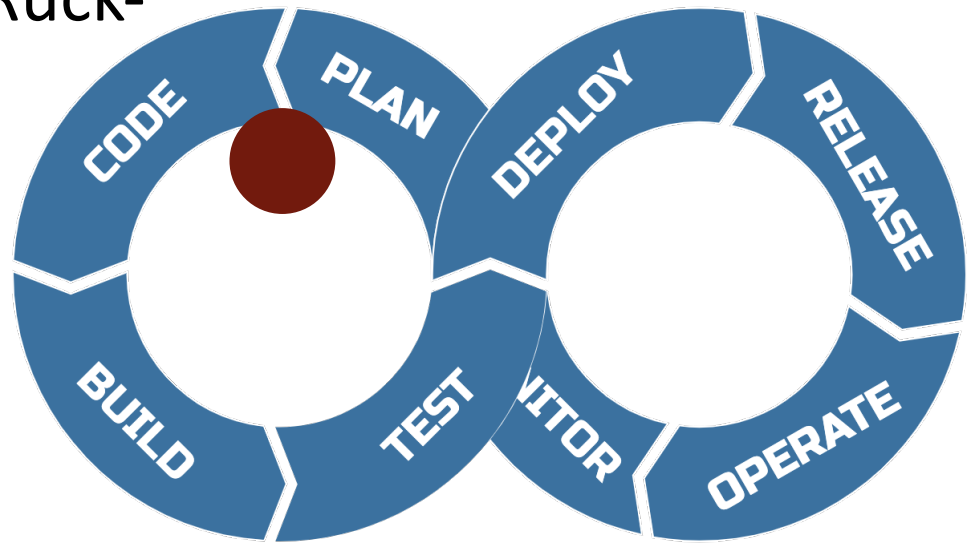
Nach der Lektion erläutern die Studierenden den Unterschied zwischen Monolith, Microservices und Serverless Architekturen.

Nach der Lektion legen die Studierenden dar, warum wir mit einer Monolith Architektur arbeiten.

(Container) Architektur

Input
– Container Diagramm aber mit Lücken (Architektur & Design)
– Anforderungen (RME)

Output
– Container / High Level / «Enterprise» Architektur
– C4 Diagramme, welche nun auch Rück-
  sicht auf Architektur nehmen
– Architektur Vorschläge



⚠ High Level, Programmiersprache etc. spielt noch keine Rolle

Agenda

- Ziel
- Was ist «Architektur»?
- Monolith
- Microservices
- Serverless
- Unser Fokus
- Netflix™ über Microservices
- Zielkontrolle

Was ist Architektur?

His conclusion was that **"Architecture is about the important stuff. Whatever that is"**. On first blush, that sounds trite, but I find it carries a lot of richness. It means that the heart of thinking architecturally about software is to decide what is important, (i.e. what is architectural), and then expend energy on keeping those architectural elements in good condition. For a developer to become an architect, they need to be able to recognize what elements are important, recognizing what elements are likely to result in serious problems should they not be controlled.

← Martin Fowler

Was ist Architektur?

Software architecture

[software architecture is the process of converting software characteristics such as flexibility, scalability, feasibility, reusability, and security into a structured solution that meets the technical and the business expectations] Mohamed Aladdin, Medium

L14 – L15

Software design

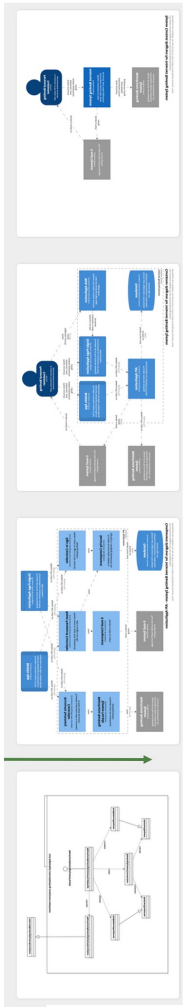
[While software architecture is responsible for the skeleton and the high-level infrastructure of a software, the software design is responsible for the code level design such as, what each module is doing, the classes scope, and the functions purposes, etc.]

Also Mohamed

L22+ – Design

Unsere Flughöhen

Wir lösen unser Huhn-Ei Problem aus den vorherigen Lektionen!



Kontextdiagramm

Containerdiagramm

Komponentendiagramm

Code-Diagramme

High Level Architektur «Container Architecture»

L14

Architektur «Component Architecture»

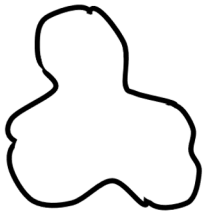
L15

Design

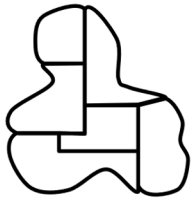
L22 +

Architektur *angewandt*

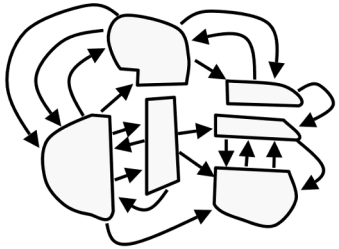
The product



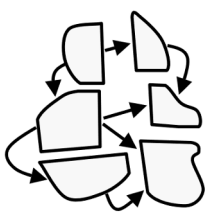
Its parts



Communication 1



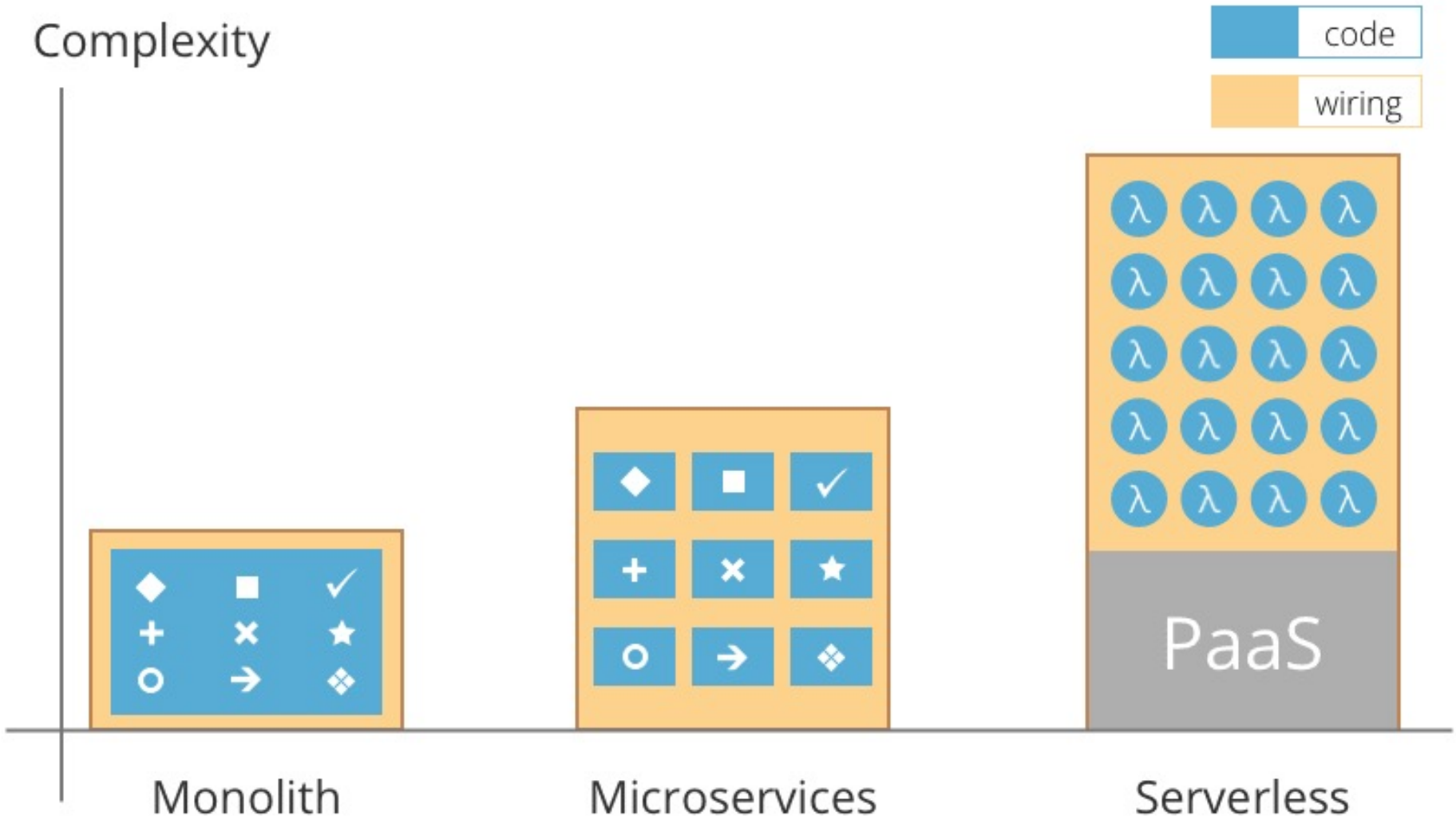
Communication 2



Design considerations

- Compatibility - With others or an older self
- Extensibility - Adding capabilities without major changes
- Maintainability - Documentation, DevOps, Transparency, clean
- **Modularity - isolated components, exchangeable, division**
- Reliability - functions under load for a specific time
- Reusability - Don’t repeat yourself DRY, reuse in other designs
- Robustness - Operate under stress / faulty data, fail graceful
- *Security - Withstand hostile acts and influence, keep data safe* ✅
- Usability - Emotions, help, experience, journeys

Vergleich der 3



Monolith

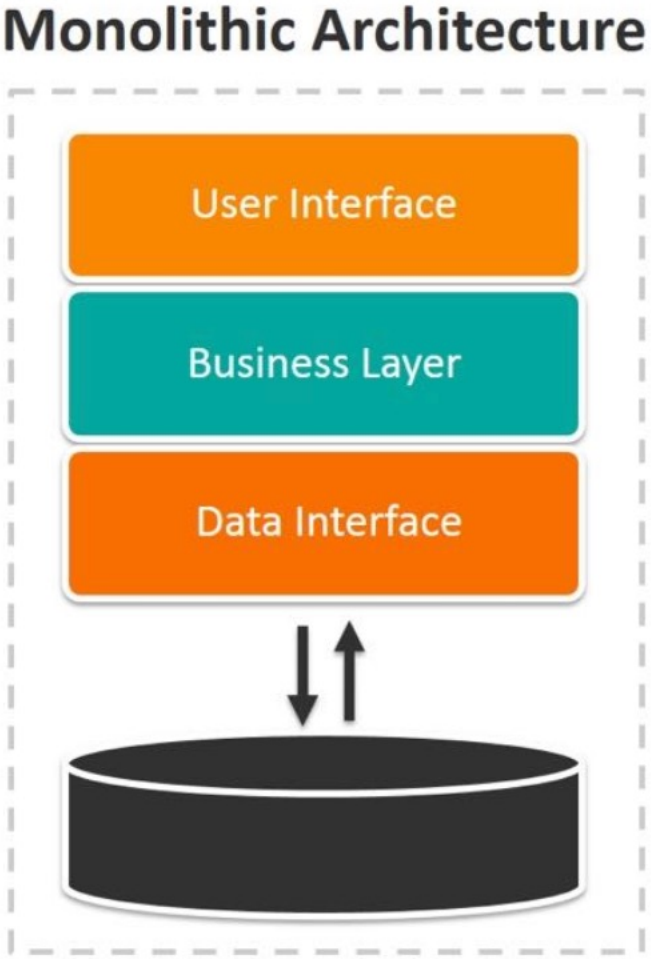
- Ein einzelner Service
  - alle Logik, alle «Domänen»
- horizontal skalierbar
- für 90% der MA und DA geeignet

+

- Geeignet für kleine Projekte
- Jegliche DevOps Aspekte (Test, Deploy, Operate etc.) werden einfacher
- Kein Overhead bei Kommunikation
- Wenn Spielregeln (Design Anforderungen) eingehalten werden ein legitimer Anfang

−

- Sobald Modularität verletzt → Spaghetti → 🪦
  - «Split the monolith» muss klar definiert und kommuniziert sein
- Ab gewisser Projektgrösse organisatorisch und technisch ungeeignet
  - zwei Teams
  - verschiedene Sprachen oder Plattformbedürfnisse

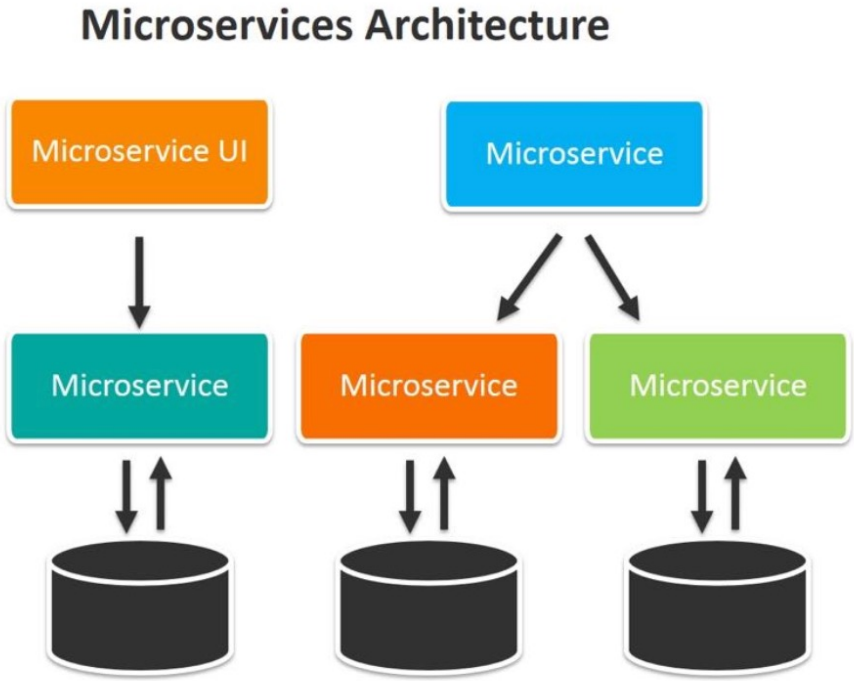


src

Ein Frontend und ein Backend sind eher zwei Monolithen als Microservices.

Microservices M$

- Viele kleine Services
  - getrennt nach Domäne (DDD)
- horizontal & vertikal skalierbar
- Komplexität hoch, für MA & DA eher herausfordernd



src

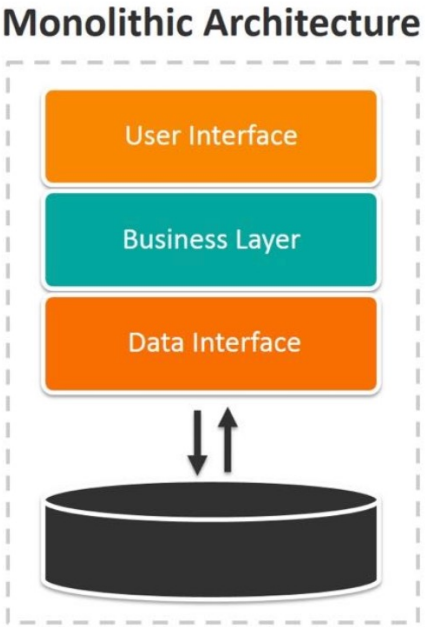
+

- Kleinere Services = schnellerer Entwicklungszyklus
- Einfachere CI/CD, einfachere Tests
- Programmiersprachen mischen
- geeignet für stabile und «erwachsene» Geschäftsmodelle mit viel «Traffic»

–

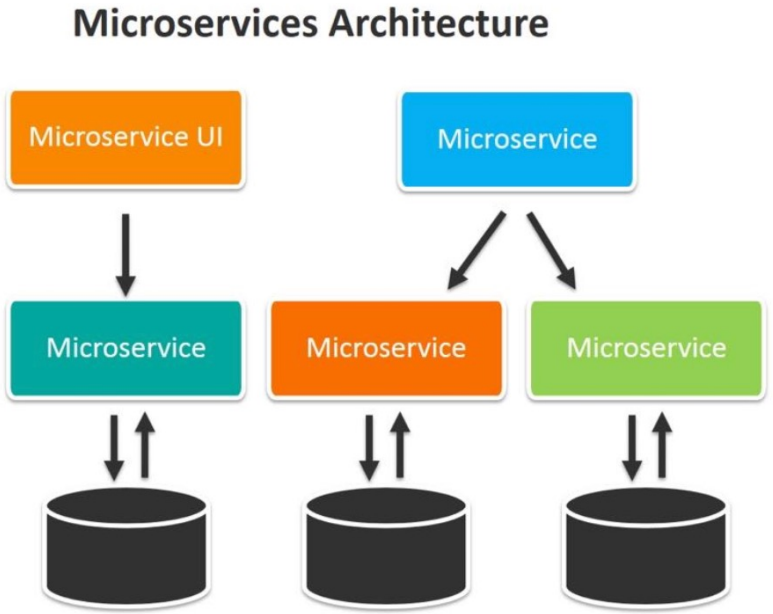
- höhere Komplexität in jeglicher Hinsicht (nach DevOps)
- Testing mehrerer Services wird kompliziert
- Kommunikation zwischen M$ erhöht Latenz und Aufwand

«Split the monolith»



Wenn Design Considerations eingehalten werden, ist dieser Split möglich!

Insbesondere und vor allem «Modularität».



Serverless Computing (PaaS)

- Komplett anderer Use Case
- Komplexität +- M$ Level
- Wir definieren nur Funktion (Code)
- Die Ausführung überlassen wir komplett jemand anderem (FaaS)

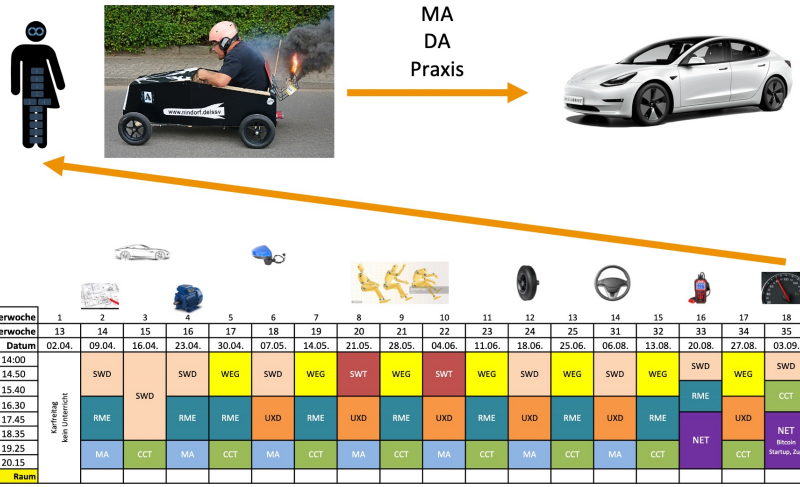
+

- Noch schnellere Entwicklung (weniger Konfiguration)
- Skalierung wird anderen überlassen
- «Pay as you go», Ressourcen on Demand
- Keine Operations nötig (FaaS)
- Eher für Spezialfälle oder Aufgaben als für den täglichen Gebrauch
  - Mails senden
  - Etwas irgendwo asynchron erledigen
  - «One time» Aktionen ohne Zustand

–

- kennt keinen «richtigen» Zustand
  - ausser durch neue Zusätze wie Caches
- Limitierte Laufzeit
  - AWS Lamda z.B. 15min max

Unser Fokus

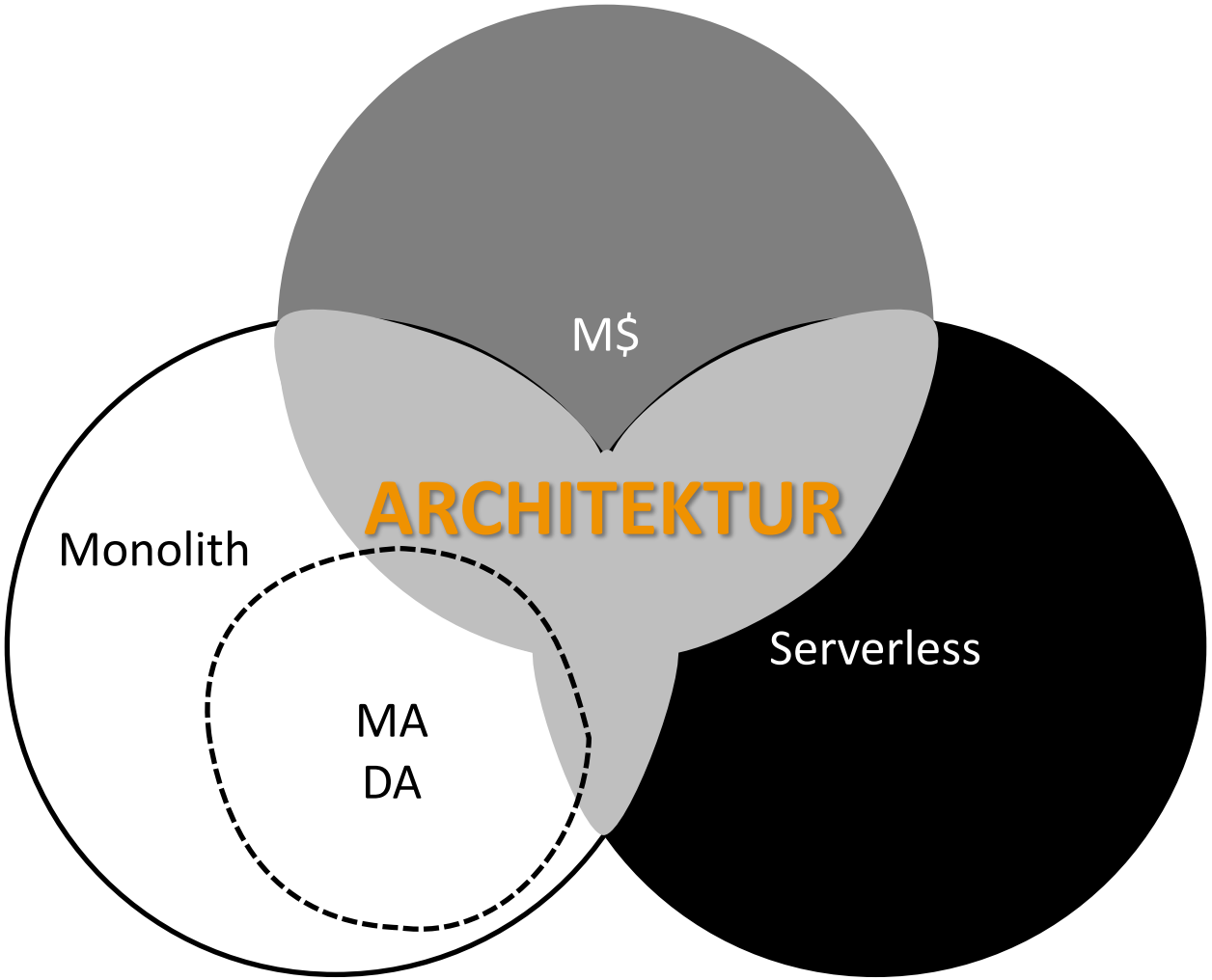


- Monolith (spring-starter)
  - Einfach zu entwickeln
  - Testen
  - Kein Overhead
  - Kleinste Komplexität
- 1 Komponente, WEG die andere
  - «quasi» zwei Monolithen
- Design erlaubt «Split the monolith»
- Breiter Horizont
  - Gedanklich nicht festgefahren

- Lektion 15: Komponenten Architektur
  - Modularity → «Split the monolith»

- Compatibility - With others or an older self
- Extensibility - Adding capabilities without major changes
- Maintainability - Documentation, DevOps, Transparency, clean
- Modularity - isolated components, exchangeable, division
- Reliability - functions under load for a specific time
- Reusability - Don't repeat yourself DRY, reuse in other designs
- Robustness - Operate under stress / faulty data, fail graceful
- Security - Withstand hostile acts and influence, keep data safe
- Usability - Emotions, help, experience, journeys

Nicht schwarz-weiss - Grau



Task

Lesen & Verstehen

AUFTRAG
⏰ 10min

https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

Lesen und verstehen Sie diesen Beitrag.

Zielkontrolle

AUFTRAG
⏰ 5min

https://forms.office.com/r/ZgDpVAmFSE

🔐 ABB Login

WEITER WISSEN.

Wir begleiten Sie!