Architekturstile und -prinzipien

in Zeiten von Containern und der Cloud.

STEFAN ZÖRNER // EMBARC

JUG Saxony Day 2022

Radebeul, 23. September 2022

Architekturstile und -prinzipien in Zeiten von Containern und der Cloud

Zusammenfassung

Container und Orchestrierungslösungen wie Kubernetes halten Einzug in Unternehmen und Organisationen. Manche Experten bezeichnen die Cloud schon länger als "das neue Normal". Hat das Auswirkung auf den Anwendungsentwurf? In diesem Vortrag diskutiere ich etablierte und aufstrebende Architekturstile und -prinzipien im Licht der neuen technischen Möglichkeiten.

Ganz konkret lernt Ihr, was Architekturstile überhaupt sind, welche Ihr kennen solltet und wie Ihr diese bei Anwendungen in Containern mit Prinzipien und Best Practices sinnvoll ausprägt. Ich diskutiere, welche technischen Fragestellungen Ihr bei zeitgemäßen Stilen zuallererst beantworten solltet. Und Ihr erfahrt, wie Ihr Euren Lösungsentwurf geeignet absichert, um Risiken früh aufzudecken und alle Beteiligten mitzunehmen.

Stefan Zörner

- Softwareentwickler + -architekt bei embarc in Hamburg
- Vorher oose, IBM, Mummert + Partner, Bayer AG, …

Schwerpunkte:

- Softwarearchitektur (Entwurf, Bewertung, Dokumentation)
- Cloud- und Java-Technologien

Stefan.Zoerner@embarc.de

@StefanZoerner

➔ xing.to/szr

embarc
Software Consulting GmbH

Mission Statement (für diesen Vortrag)

Im Anschluss

- … versteht Ihr, was Architekturstile und -prinzipien überhaupt sind.
- … könnt Ihr wichtige Stile erklären und unterscheiden.
- … kennt Ihr wichtige Fragestellungen rund um die Detaillierung eines Stils.
- … wisst Ihr, wie sich die Stile bei Anwendungen in Containern sinnvoll ausprägen lassen.
- … versteht Ihr die Wechselwirkung zwischen Stil und Zielumgebung.

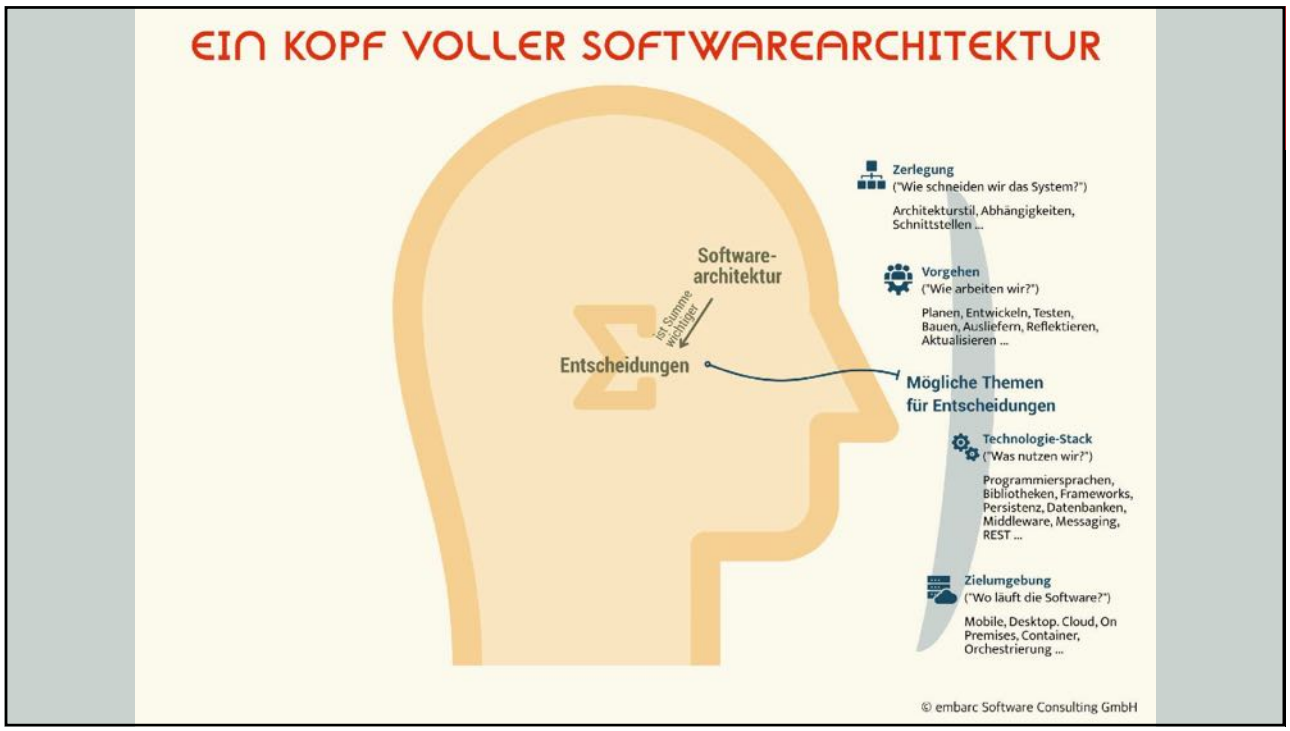
Agenda

1. Architekturstile und -prinzipien
2. Kleine Stilkunde der Softwarearchitektur
3. Einen Stil ausprägen
4. Kubernetes und die Cloud
5. Kein Fazit aber ein Fahrplan
6. Weitere Informationen

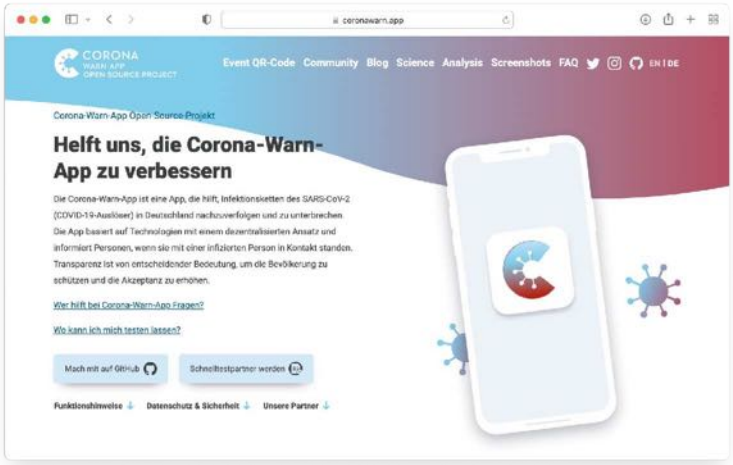
Agenda

1. Architekturstile und -prinzipien
2. Kleine Stilkunde der Softwarearchitektur
3. Einen Stil ausprägen
4. Kubernetes und die Cloud
5. Kein Fazit aber ein Fahrplan
6. Weitere Informationen

1



Beispiel: Die deutsche Corona-Warn-App



Steckbrief

Thema:
Contact Tracing App

Veröffentlicht:
Juni 2020

Downloads (iOS + Android):
46,9 Millionen

Bereitgestellte Testergebnisse:
216,2 Millionen

(Stand Anfang September 2022)

Ausgewählte Entscheidungen der Corona-Warn-App

Zerlegung

- Ganz grob: Apps als Clients, serverseitiges Backend
- Fachliche Zerlegung des Backends in Vertikalen

Zielumgebung

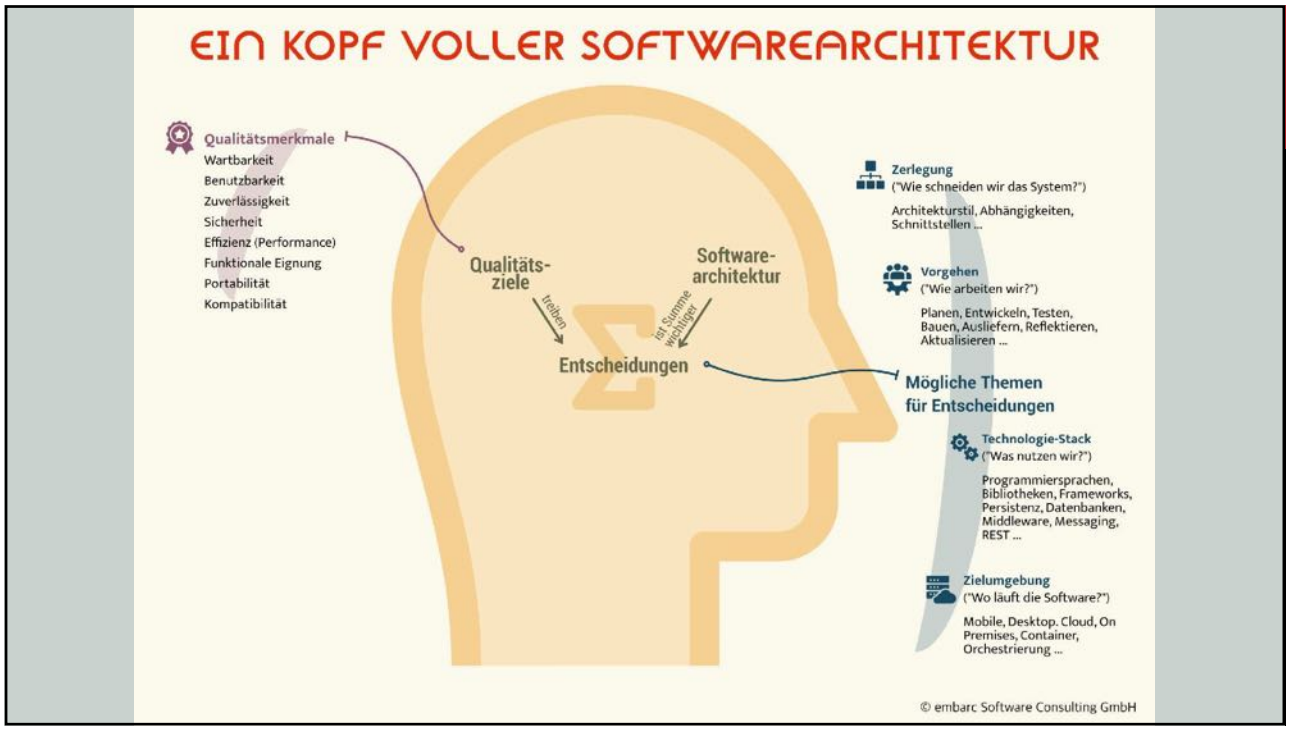
- Apps in entsprechenden Stores
- Backend: Kubernetes in Open Telekom Cloud (Public Cloud)

Technologie-Stack

- Mobile Apps in Swift bzw. Kotlin
- Backend in Java mit Spring Boot
- PostgreSQL

Vorgehen

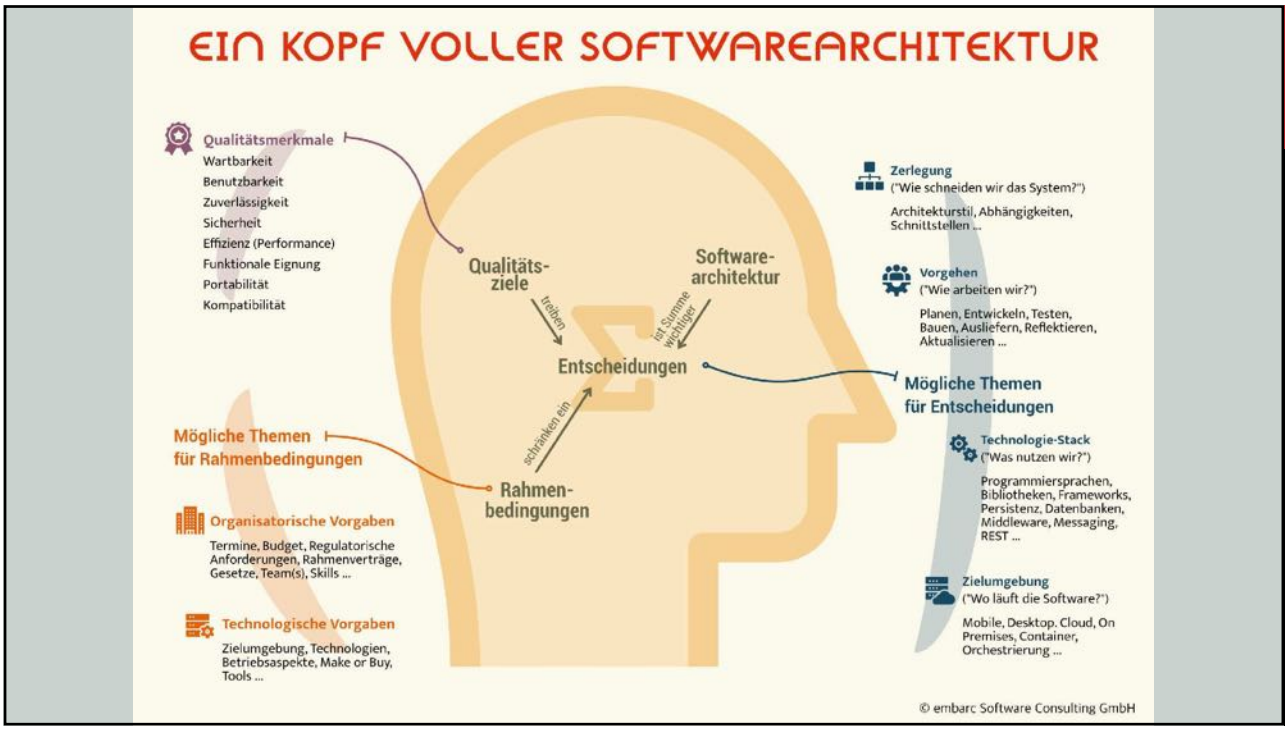
- Prinzip der Datensparsamkeit
- Entwicklung Open Source (GitHub)



Beispiel: Top-Qualitätsziele Corona-Warn-App

Die vorrangigen Architekturziele der CWA in der Reihenfolge ihrer Wichtigkeit.

	Ziel	Beschreibung
	Höchster Datenschutz	Der Schutz der personenbezogenen Daten hat oberste Priorität. *(Sicherheit)*
	Effektive Warnfunktionalität	Die App ist ein effektiver Baustein bei der Pandemie-Bekämpfung. *(Funktionale Eignung)*
	Attraktive Lösung für App-Nutzer	Die App ist leicht zu installieren sowie intuitiv und effizient zu bedienen. *(Benutzbarkeit)*
	Hohe Zuverlässigkeit	Die Lösung geht mit Lastspitzen wegen hoher Nutzer- oder Infektionszahlen ebenso souverän um, wie mit Störungen und böswilligen Angriffen. *(Zuverlässigkeit)*
	Gute Wartbarkeit	Die Software lässt sich leicht anpassen, wenn z. B. Nutzer/-innen, Politik oder neue Forschungsergebnisse es erfordern. *(Wartbarkeit/Erweiterbarkeit)*

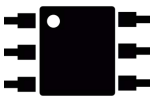


Beispiel: Rahmenbedingungen CWA

Ausgewählte Maßgaben an Entwicklung und Betrieb sowie einige Informationen zum politischen Umfeld.

Technische Vorgaben

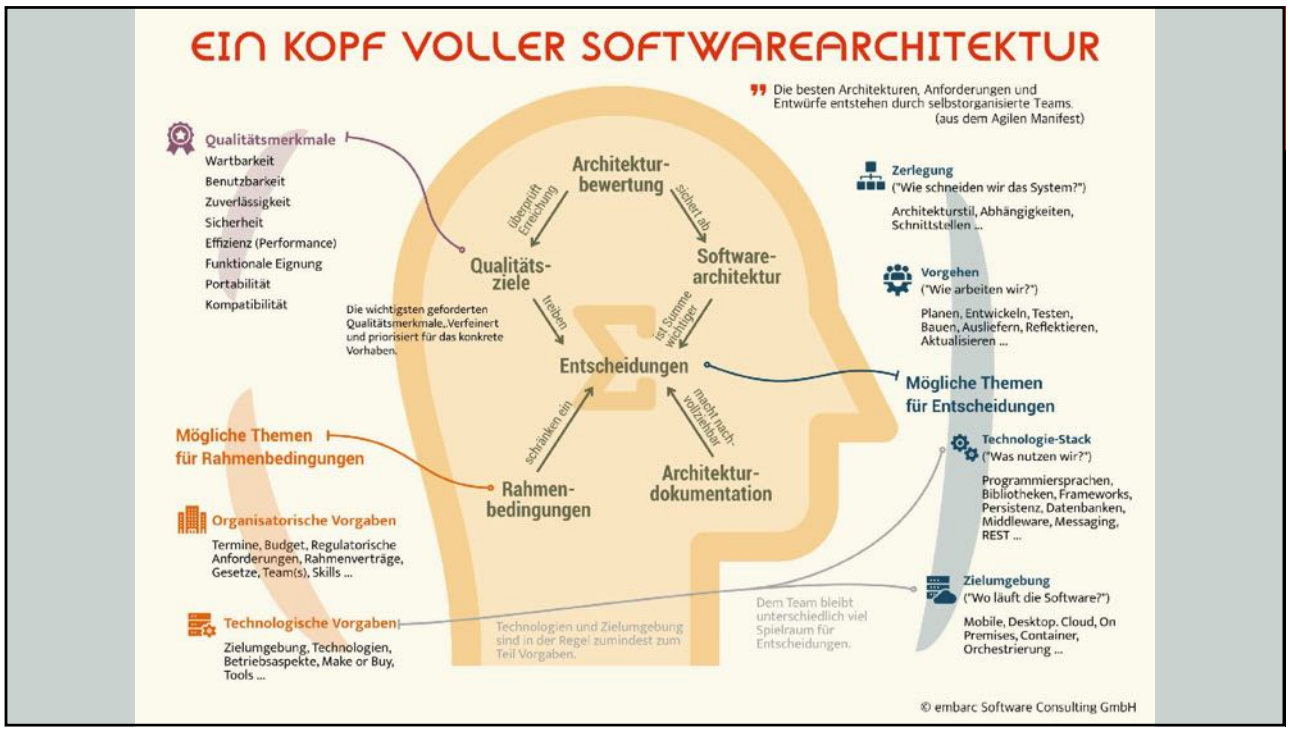
- Entwicklung von nativen, mobilen Clients für Android- und iOS-Smartphones
- Verfolgen eines dezentralen Ansatzes für die Datenspeicherung
- Einsatz des Exposure Notification Framework von Google und Apple
- Betrieb der Backend-Komponenten bei einem deutschen Provider in deutschen Rechenzentren



Rahmenbedingungen CWA (Fortsetzung)

Organisatorischer Rahmen und Umfeld

- Auftraggeber: Deutsche Bundesregierung
- Entwicklung und Betrieb durch ein Konsortium aus zwei Auftragnehmern (SAP und Deutsche Telekom)
- Start der Entwicklung 04/2020
- enger Zeitrahmen (Apps zum Download verfügbar ab 06/2020)
- hoher politischer Druck, viele Parteien involviert (Ministerien, Behörden, RKI ...)
- große Medienaufmerksamkeit
- gewisse Skepsis innerhalb der Bevölkerung


EIN KOPF VOLLER SOFTWAREARCHITEKTUR
Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
(aus dem Agilen Manifest)
Qualitätsmerkmale
Wartbarkeit
Benutzbarkeit
Zuverlässigkeit
Sicherheit
Effizienz (Performance)
Funktionale Eignung
Portabilität
Kompatibilität
Architektur-bewertung
überprüft Erreichung
sichert ab
Qualitäts-ziele
Software-architektur
treiben
ist Summe wichtiger
Die wichtigsten geforderten Qualitätsmerkmale. Verfeinert und priorisiert für das konkrete Vorhaben.
Entscheidungen
schränken ein
macht nach-vollziehbar
Rahmen-bedingungen
Architektur-dokumentation
Mögliche Themen für Rahmenbedingungen
Organisatorische Vorgaben
Termine, Budget, Regulatorische Anforderungen, Rahmenverträge, Gesetze, Team(s), Skills ...
Technologische Vorgaben
Zielumgebung, Technologien, Betriebsaspekte, Make or Buy, Tools ...
Technologien und Zielumgebung sind in der Regel zumindest zum Teil Vorgaben.
Dem Team bleibt unterschiedlich viel Spielraum für Entscheidungen.
Zerlegung
("Wie schneiden wir das System?")
Architekturstil, Abhängigkeiten, Schnittstellen ...
Vorgehen
("Wie arbeiten wir?")
Planen, Entwickeln, Testen, Bauen, Ausliefern, Reflektieren, Aktualisieren ...
Mögliche Themen für Entscheidungen
Technologie-Stack
("Was nutzen wir?")
Programmiersprachen, Bibliotheken, Frameworks, Persistenz, Datenbanken, Middleware, Messaging, REST ...
Zielumgebung
("Wo läuft die Software?")
Mobile, Desktop. Cloud, On Premises, Container, Orchestrierung ...
© embarc Software Consulting GmbH


Kölner Dom

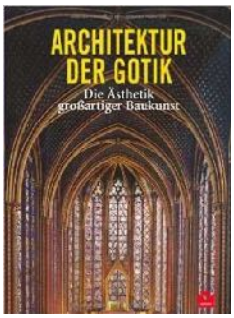
Baustil: Gotik.

"Der Kölner Dom ist eine der größten Kathedralen im gotischen Baustil."

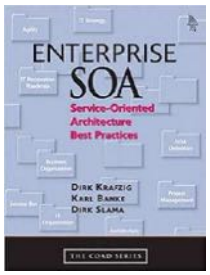
(Wikipedia)

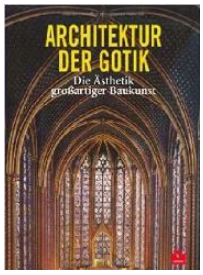
Charakteristische Eigenschaften der Gotik

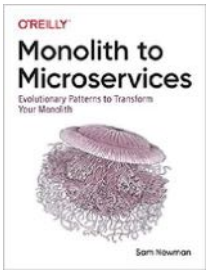
- Betonung der Vertikalen
- weitgehende Durchbrechung der Außenwandflächen durch Fenster mit Spitzbögen
- Reduzierung der Wandstärken
- …



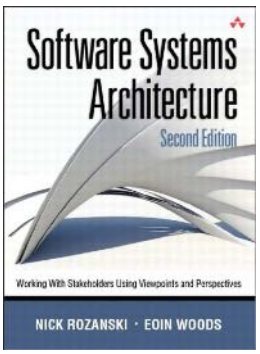
Ausgewählte Bücher zu Architekturstilen







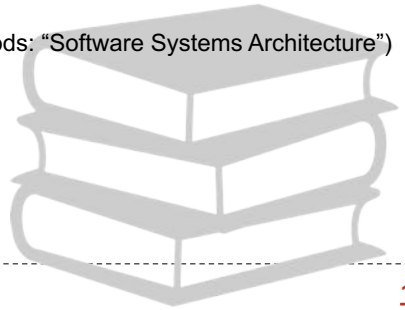
Was ist ein Architekturstil (in Software)?



*"Ein Architekturstil drückt ein **grundlegendes Organisationsschema** für Softwaresysteme aus. Er stellt eine Reihe vordefinierter **Elementtypen** bereit, spezifiziert ihre Verantwortlichkeiten und enthält **Regeln und Richtlinien** für die Organisation der Beziehungen zwischen ihnen."*

(N. Rozanski + E. Woods: "Software Systems Architecture")

Im Englischen Original: "An Architectural style expresses a fundamental organizational schema for software systems. It provides a set of predefined element types, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them."



Ein Beispiel: Microservices

*"Kurz gesagt, der Microservice-Architekturstil ist ein Ansatz zur Entwicklung einer **einzelnen Anwendung** als einer **Familie kleiner Services**, die jeweils in einem eigenen Prozess laufen und mit leichtgewichtigen Mechanismen kommunizieren."*

(James Lewis, Martin Fowler, 2014)
➔ https://martinfowler.com/articles/microservices.html

Charakteristische Eigenschaften

(verdichtet aus dem Blog-Beitrag von Lewis und Fowler)

- Zerlegung in relativ kleine (fachliche) Services
- Services sehr lose gekoppelt
- Services einzeln installierbar und upgradebar
- Dezentrale Datenhaltung
- Hoher Freiheitsgrad bei Technologieauswahl

Warum brauche ich einen Architekturstil?

- **Erfahrungswissen**
  ... bewährte Lösungen nutzen.
- **Kommunikation**
  ... anderen die Grundidee(n) der Lösung vermitteln.
- **Strukturierung**
  ... Rahmen für die Entwicklung schaffen.

Architekturstile und -prinzipien

in Zeiten von Containern und der Cloud.

STEFAN ZÖRNER // EMBARC

JUG Saxony Day 2022

Radebeul, 23. September 2022

Was ist ein Prinzip (ganz allgemein)?

„Ein Prinzip (Plural: Prinzipien; von lat. principium = Anfang, Beginn, Ursprung, Grundsatz) ist das, aus dem ein anderes seinen Ursprung hat. ... Allgemeinsprachlich handelt es sich bei einem Prinzip um einen Grundsatz, eine feste Regel, an die man sich hält. “

(Wikipedia)

Nicht-IT-Beispiele für Prinzipien

Philosophie, Psychologie

- Kategorischer Imperativ (Kant)
- Lustprinzip (Freud)

Naturwissenschaften

- Survival of the Fittest (Darwin)
- Zufallsprinzip
- Archimedisches Prinzip
- Pauli-Prinzip

Prinzipien in der Softwareentwicklung (Beispiele)

Allgemeine Prinzipien (Entwurf, Arbeitsweise)

KISS (Keep it simple, stupid)

YAGNI (You Aren't Gonna Need It)

DRY (Don't repeat yourself)

SOLID-Prinzipien für den (objektorientierten) Entwurf

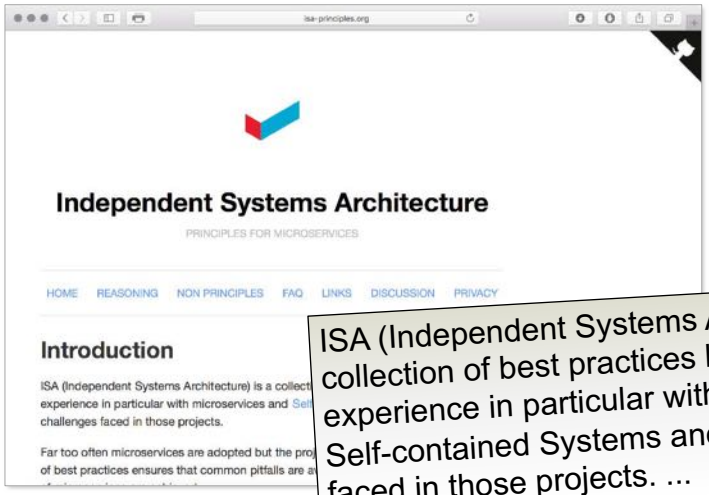
- ✓ [S]ingle Responsibility Principle
- ✓ [O]pen/Closed Principle
- ✓ [L]iskov Substitution Principle
- ✓ [I]nterface Segregation Principle
- ✓ [D]ependency Inversion Principle

Weitere prominente IT-Prinzipien

Geheimnisprinzip („Information Hiding", D. Parnas 1972)

Conway's Law (M. E. Conway 1968)

ISA Principles



ISA (Independent Systems Architecture) is a collection of best practices based on experience in particular with microservices and Self-contained Systems and the challenges faced in those projects. ...

➔ https://isa-principles.org

ISA Principle 1 (/9)

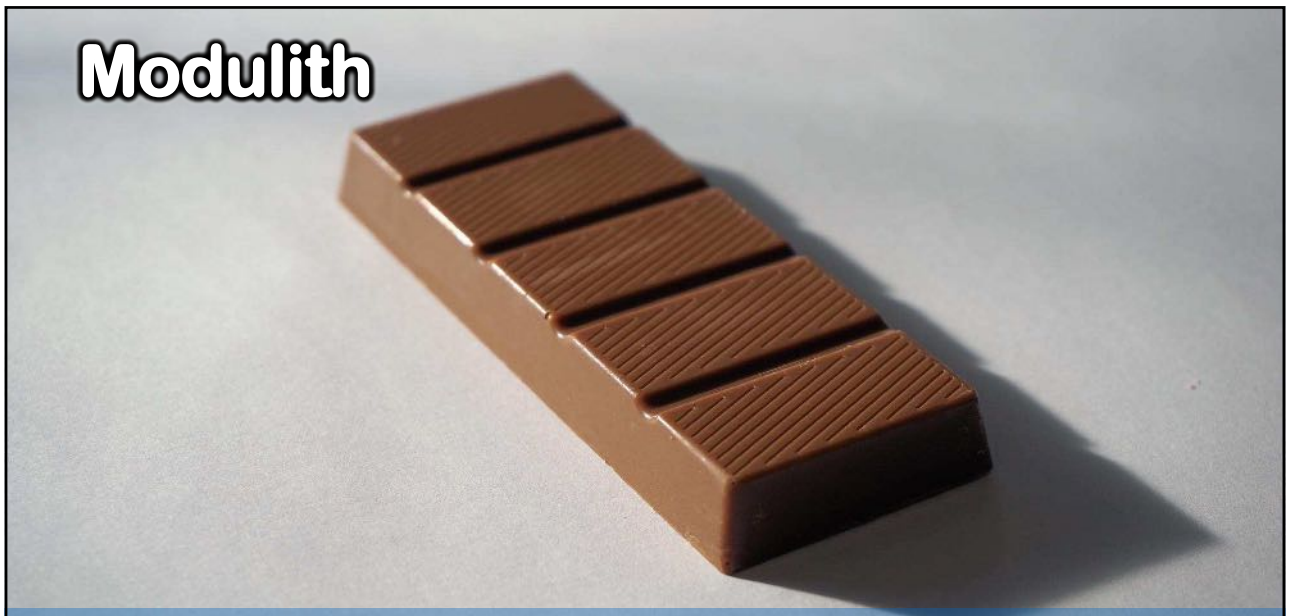
*"The system must be divided into **modules** that provide **interfaces**. Access to other modules is only possible through these interfaces. Therefore, modules must not directly depend on implementation details of other modules, e.g. the internal data representation in a database.*
The rest of the principles define how modules might be implemented, and how ISA differs from other modularization approaches."

→ https://isa-principles.org

Agenda

2

Modulith



„Strukturiert gebaut, in einem Stück geliefert.“

Steckbrief: Modulith

Klare Strukturierung der Anwendung in Module, geplante Abhängigkeiten, einzelne Deployment-Einheit.

- „Strukturiert gebaut, in einem Stück geliefert.“
- (Deployment-)Monolith,
  als Anti-Pattern: Big Ball of Mud (wenn unstrukturiert)
- Modul (auch: Komponente, Subsystem)
- Atlassian Confluence
- Container Ansatz: Ein Image als Deployment-Format für die komplette Anwendung.
  (+) Einheitliches Format für Auslieferung
  (+) ggf. Vereinheitlichung im Entwicklungsprozess / Betrieb

Legende

- Slogan des Stils, Leistungsversprechen
- synonyme oder nah verwandte Stile, „auch“
- Strukturierendes Element (Bauteil)
- Beispielanwendung(en) für den Stil, Architektur-Ikonen
- Ansatz mit Containern, mögliche Vorteile

Steckbrief: Mehrschichtarchitektur

Gliederung der Anwendung in technische und/oder logische Schichten. Zugriff jeweils nur auf die nächste (tiefere) Schicht.

„Leicht zu portieren durch Abstraktion."

N-Tier, Client-Server

Schicht (im Englischen: Layer, Tier)

SAP R/3, WordPress

Ansatz mit Containern: einzelne Schichten in separate Container.
(z.B. komplette Geschäftslogik)
(+) Einheitliches Format für Auslieferung
(+) ggf. Vereinheitlichung im Entwicklungsprozess / Betrieb
(+) Bei geeigneter Implementierung (grob) skalierbar und aktualisierbar

Legende

- Slogan des Stils, Leistungsversprechen
- synonyme oder nah verwandte Stile, „auch"
- Strukturierendes Element (Bauteil)
- Beispielanwendung(en) für den Stil, Architektur-Ikonen
- Ansatz mit Containern, mögliche Vorteile

Steckbrief: Microservices

Anwendung besteht aus einzeln deploybaren, sehr lose gekoppelten Services. Typisch: technologische Vielfalt

„Flexibilität durch kleine, unabhängige Teile“

Self-contained Systems (SCS)

Service (auch: Vertikale, SCS)

Netflix, Xing (für SCS)

Natürlicher Ansatz: Services jeweils als Container.
(+) Bei geeigneter Implementierung fein skalierbar und aktualisierbar

Legende

- Slogan des Stils, Leistungsversprechen
- synonyme oder nah verwandte Stile, „auch“
- Strukturierendes Element (Bauteil)
- Beispielanwendung(en) für den Stil, Architektur-Ikonen
- Ansatz mit Containern, mögliche Vorteile

Steckbrief: Serverless

Geschäftslogik deployt als kleine, zustandslose Funktionen.
Ressourcen-Verbrauch nur bei Aufruf/Ausführung.

„Der erste wahre Cloud-Native-Stil“

Function as a Services (FaaS)

Funktion

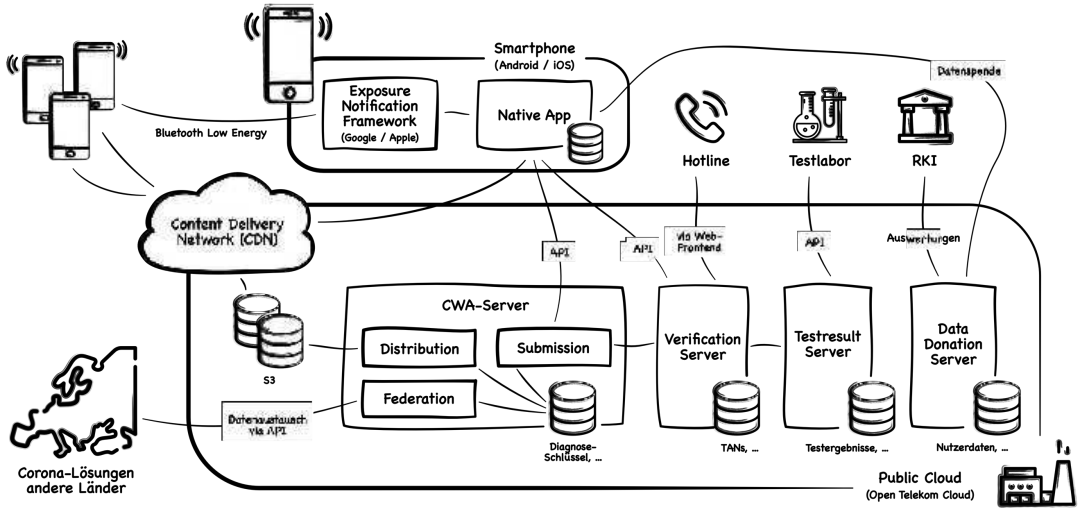
Alexa Skills

Verschiedene Container-Lösungen für Serverless z.B. in Kubernetes verfügbar. Container werden davon wegabstrahiert.
(-) Wirken schwergewichtig und gefährden teilweise Serverless-Vorteile.
(+) Lastspitzen mit Containern auffangen

Legende

- Slogan des Stils, Leistungsversprechen
- synonyme oder nah verwandte Stile, „auch“
- Strukturierendes Element (Bauteil)
- Beispielanwendung(en) für den Stil, Architektur-Ikonen
- Ansatz mit Containern, mögliche Vorteile

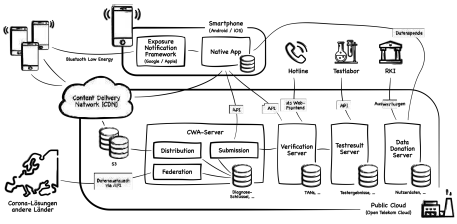
Corona-Warn-App: Informelles Überblicksbild



Quelle der Abbildung: S. Zörner, F. Sippach: „So gehen Architektur-Reviews! Entlang der Corona-Warn-App“, OOP 2021

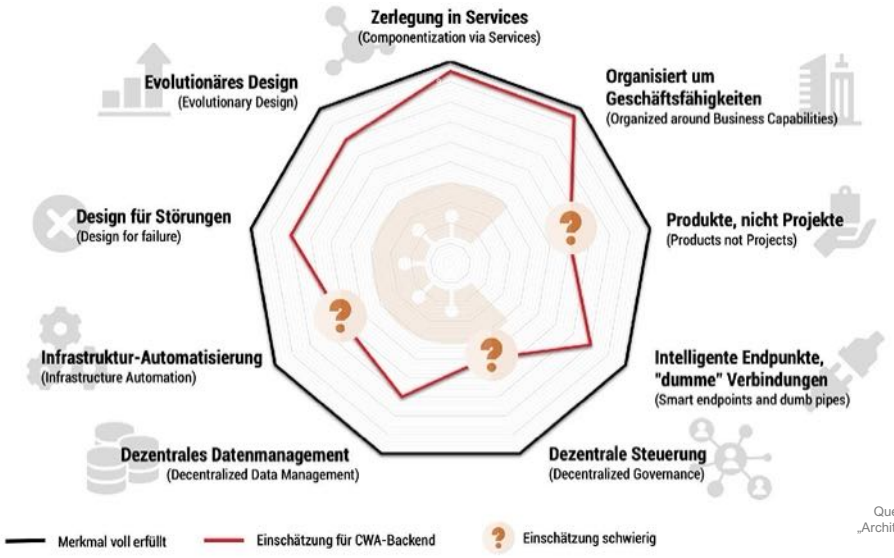
Kleines Quiz …

Welchen Architekturstil hat die Corona-Warn-App? Kreuzt die am besten passende Antwort an!



- ○ Monolith / Modulith
- ○ Mehrschichtarchitektur / Client/Server
- ○ Microservices / SCS
- ○ Serverless
- ○ Das ist eine Trickfrage

Einschätzung bezüglich Microservices



Quelle der Abbildung: S. Zörner: „Architekturstile und –prinzipien …", Informatik Aktuell 2021

Agenda

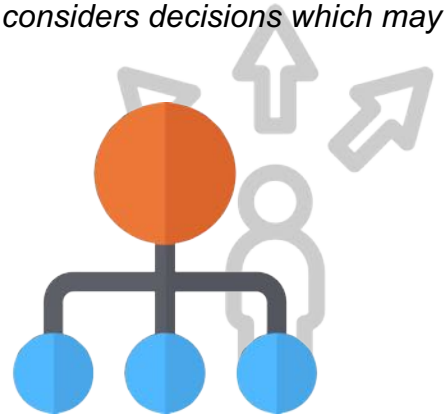
1 Architekturstile und -prinzipien
2 Kleine Stilkunde der Softwarearchitektur
3 **Einen Stil ausprägen**
4 Kubernetes und die Cloud
5 Kein Fazit aber ein Fahrplan
6 Weitere Informationen

3

ISA Principle 2 (/9)

*" … 2. The system must have two clearly separated levels of architectural decisions: **The Macro Architecture** comprises decisions that cover all modules. All further principles are part of the Macro Architecture.*
*The **Micro Architecture** considers decisions which may be taken individually for each module. …"*

→ https://isa-principles.org



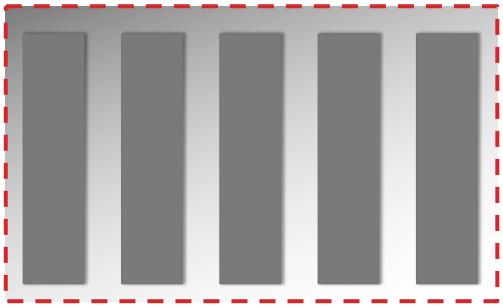
Ein Spannungsfeld

***Eine* Anwendung**
So sollte es sich für den Benutzer auch anfühlen.

Technologische Vielfalt in den "Vertikalen"
- Paradigmen
- Programmiersprachen
- Bibliotheken und Frameworks
- ...

z.B. in der Persistenz (polyglott)

Die UI-Frage

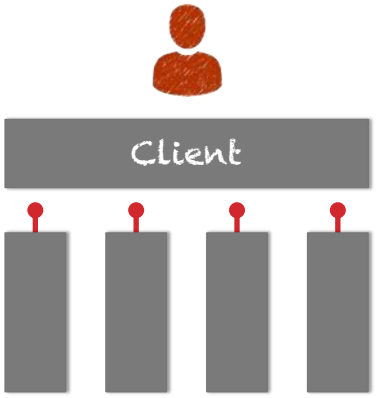


Anforderung:
Trotz mehrerer Teile präsentiert sich die Anwendung dem Benutzer “aus einem Guss“.

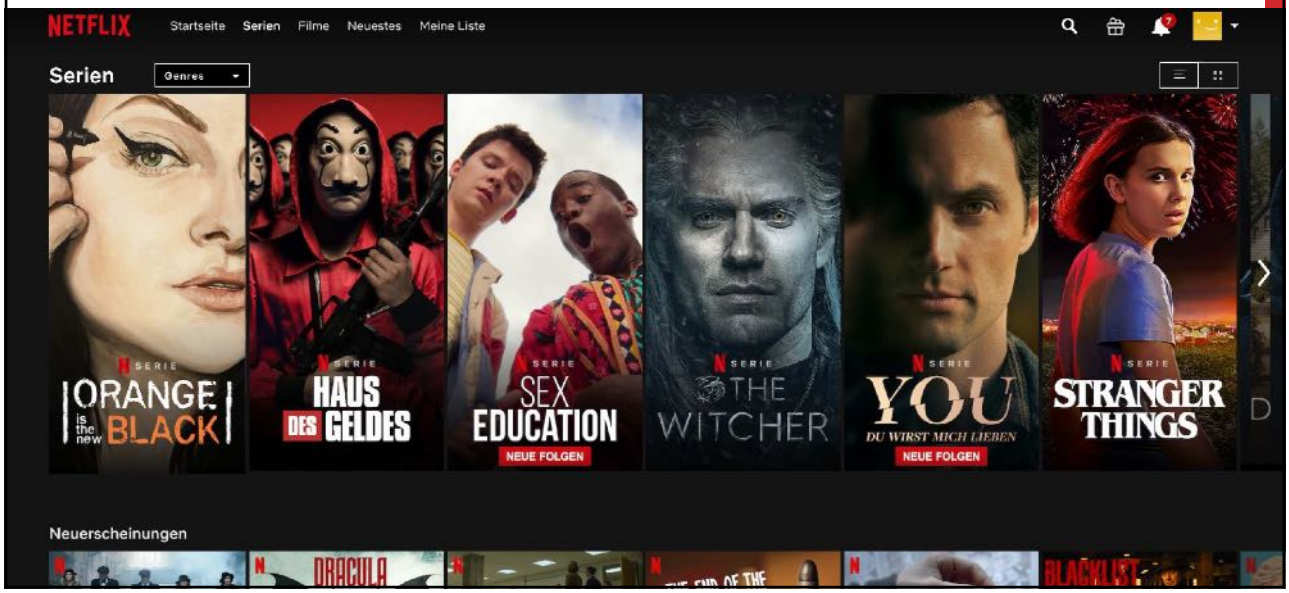
Frage:
Wie realisieren wir mit mehreren Teilen *ein* UI?

Antworten: (Extreme) Option A

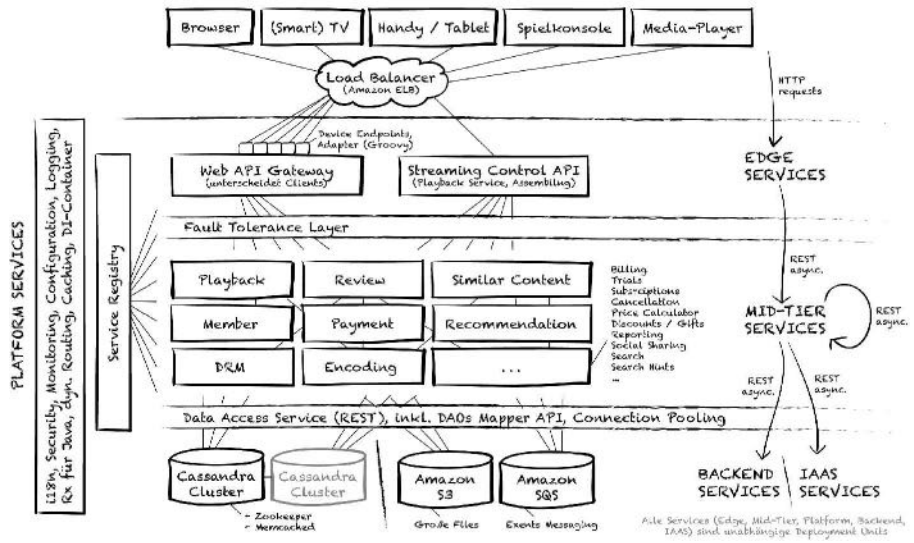
Microservices (allgemeiner: Vertikalen) haben keinen UI-Anteil, sondern nur eine (z.B. REST)-Schnittstelle. Ein gemeinsamer Client greift auf alle Services zu.



“Netflix is the king of online streaming, using more global bandwidth than cat videos and piracy combined.”



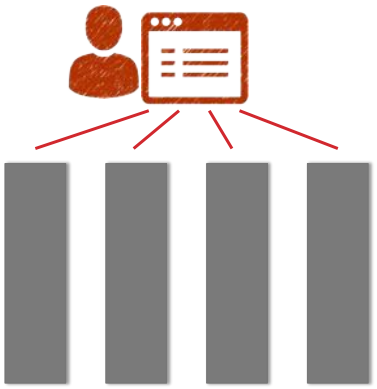
Beispiel: Netflix



Quelle: Stefan Toth, Stefan Zörner “Gut das ist? Umgekehrte Architekturbewertung eines Internetgiganten”

Antworten: (Extreme) Option B

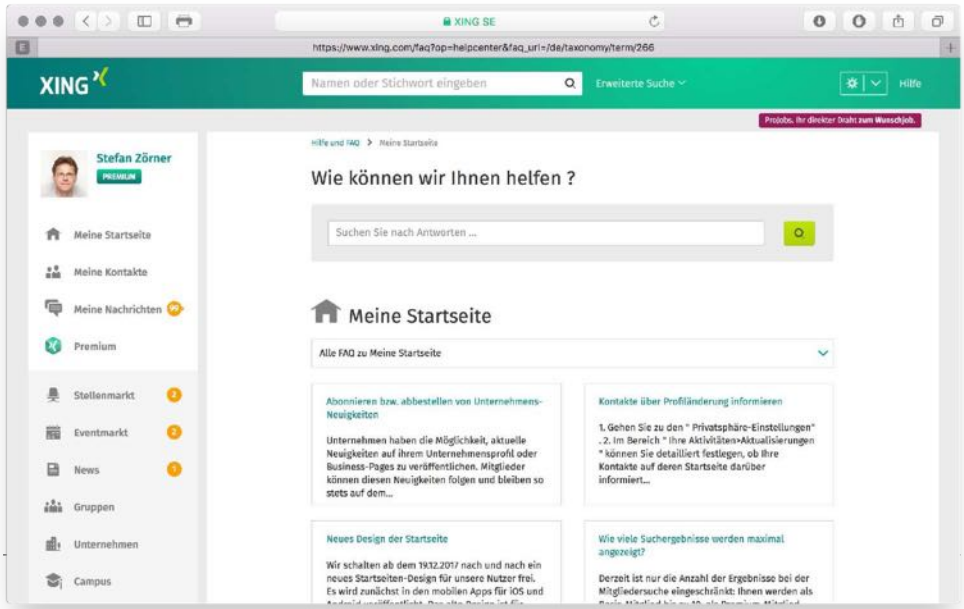
Vertikalen (z.B. Microservices) bringen jeweils die komplette UI für „ihr Thema“ mit. Die Integration erfolgt z.B. über Links im Browser.



Eine Vertikale ist für die gesamte Seite im Browser verantwortlich.

Microservices

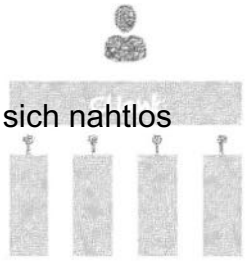
Beispiel: Xing



Besondere Stärken der Ansätze

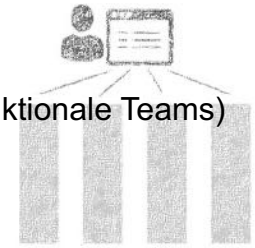
Gemeinsames UI für UI-lose Vertikalen

- Inhalte und Funktionen aus verschiedenen Themen lassen sich nahtlos integrieren – keine Brüche
- Einheitliches User Interface ist leicht erreichbar
- Spezialisiertes Team für eine optimale UX denkbar

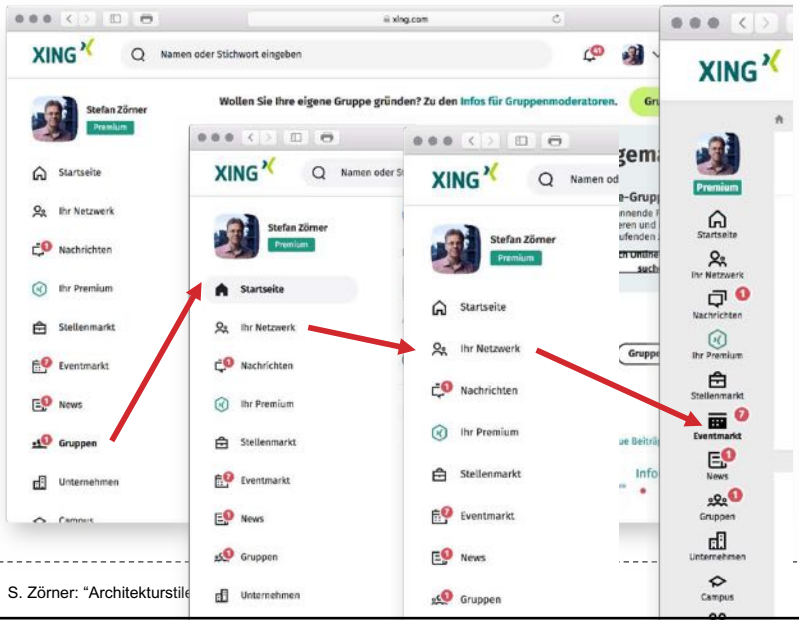


Separate UIs für die Vertikalen

- Teams vollumfänglich für Thema verantwortlich (Cross-funktionale Teams)
- Technologische Freiheiten im UI-Bereich
- Unabhängiges Arbeiten der Teams leichter möglich



Beispiel: Uneinheitliche Navigation in Xing



Gruppen

Startseite

Ihr Netzwerk

Eventmarkt

Kompromisse ...

Begriffe nach ISO 25010

Benutzbarkeit (Usability)
Ist die Software intuitiv zu bedienen, leicht zu erlernen, attraktiv?

Portabilität (Portability)
Ist die Software leicht auf andere Zielumgebungen (z.B. anderes OS) übertragbar?

Funktionale Eignung (Functional Suitability)
Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen? ...

Effizienz (Performance)
Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch? ...

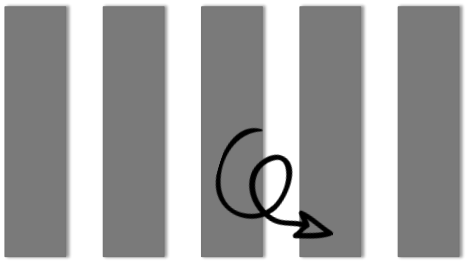
Kompatibilität (Compatibility)
Ist die Software konform zu Standards, arbeitet sie gut mit anderen zusammen?

Zuverlässigkeit (Reliability)
Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt? ...

Sicherheit (Security)
Ist das System sicher vor Angriffen? Sind Daten und Funktion vor unberechtigtem Zugriff geschützt? ...

Wartbarkeit (Maintainability)
Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden? ...

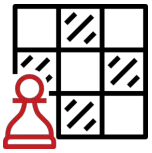
Die Kommunikationsfrage

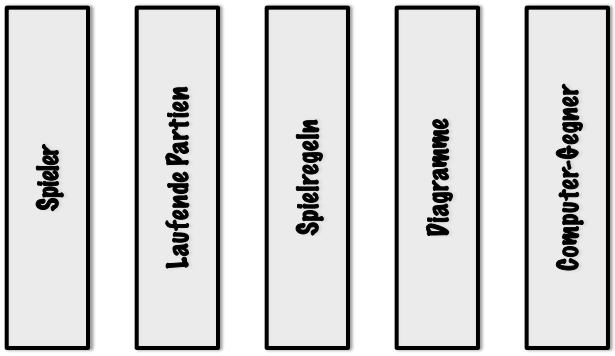


Anforderung:
Ein Service benötigt Funktionalität und/oder Daten eines anderen Services, um seine Aufgabe zu erledigen.

Frage:
Dürfen Services miteinander reden, und wenn ja wie?
(Und wenn nein – wie realisieren wir dann obige Anforderung?)

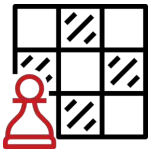
Beispiel: Eine Online-Schachplattfform

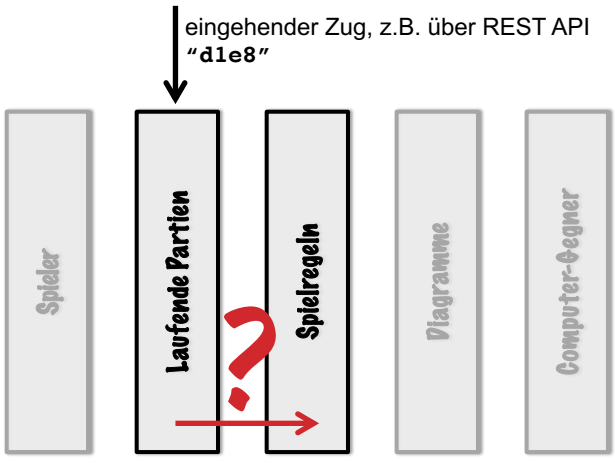




Nutzung der Spielregeln aus den laufenden Partien, um eingehende Züge zu prüfen.

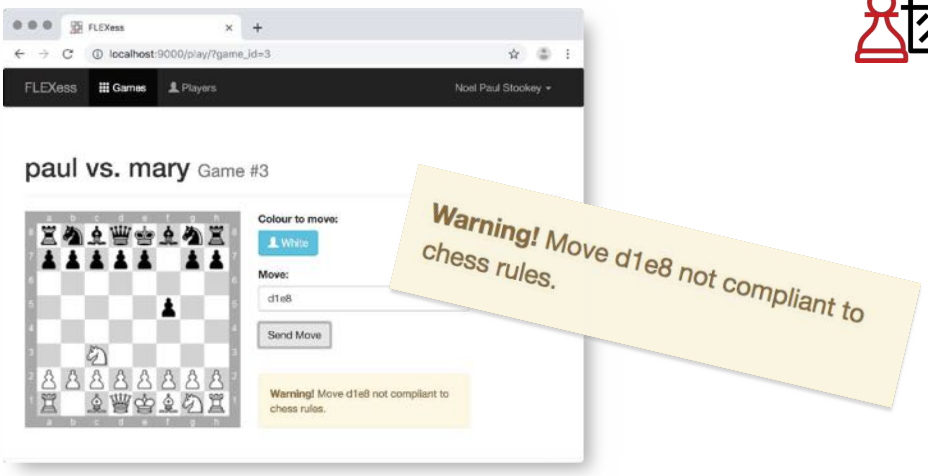
Beispiel: Eine Online-Schachplattfform





Nutzung der Spielregeln aus den laufenden Partien, um eingehende Züge zu prüfen.

Beispiel: Eine Online-Schachplattform



Die Frage im Detail (1)

direkt vs. indirekt

Direkt
Der Client „kennt“ den Service und spricht ihn direkt an.

Indirekt
Der Client kommuniziert über eine Middleware, die ihn vom Service entkoppelt. Client und Server „kennen“ sich nicht.

Die Frage im Detail (2)

synchron vs. asynchron

Synchron
Der Client wartet auf die Antwort des Service und blockiert.

Asynchron
Der Client wartet nicht auf die Antwort. Er erhält diese falls nötig später, ggf. über einen anderen Kanal.

Eine Option: direkt + synchron

Top-Herausforderung aus „direkt"
Der Client muss den (ggf. redundanten) Service auffinden.

(Typische Lösung: Service Registry)

Top-Herausforderung aus „synchron"
Wenn der Service nicht verfügbar ist oder fehlerhaft bzw. langsam antwortet, zieht das weitere Systemteile runter.

(Typische Lösung: Circuit Breaker, Bulk Heads ...)

Microservices Patterns

The patterns

How to apply the patterns

Application architecture patterns
- Monolithic architecture
- Microservice architecture

Decomposition
- Decompose by business capability
- Decompose by subdomain
- Self-contained Service new
- Service per team new

Data management
- Database per Service
- Shared database
- Saga
- API Composition
- CQRS
- Domain event
- Event sourcing

Transactional messaging
- Transactional outbox
- Transaction log tailing
- Polling publisher

Testing
- Service Component Test
- Consumer-driven contract test
- Consumer-side contract test

Deployment patterns
- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

Cross cutting concerns
- Microservice chassis
- Externalized configuration

Communication style
- Remote Procedure Invocation
- Messaging
- Domain-specific protocol

External API
- API gateway
- Backend for front-end

Service discovery
- Client-side discovery
- Server-side discovery
- Service registry
- Self registration
- 3rd party registration

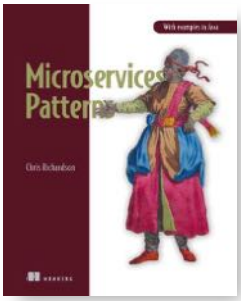
Reliability
- Circuit Breaker

Security
- Access Token

Observability
- Log aggregation
- Application metrics
- Audit logging
- Distributed tracing
- Exception tracking
- Health check API
- Log deployments and changes

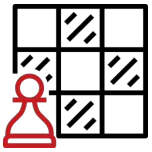
UI patterns
- Server-side page fragment composition
- Client-side UI composition

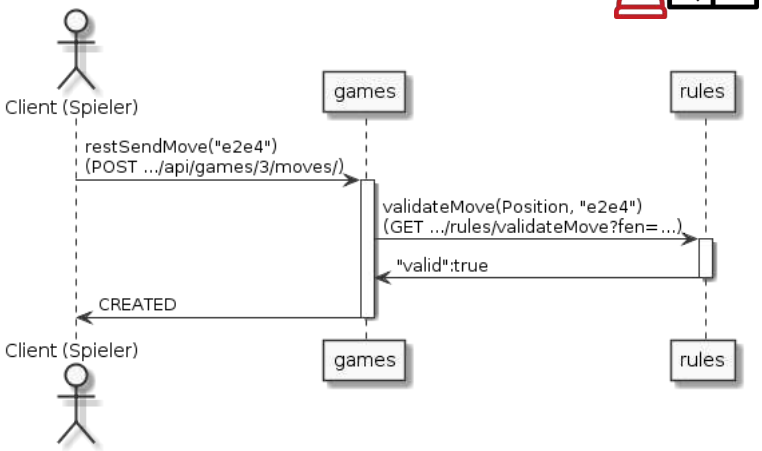
Chris Richardson
"Microservice Patterns"
Manning, Oktober 2018



➔ https://microservices.io/patterns/

Implementierung im Beispiel





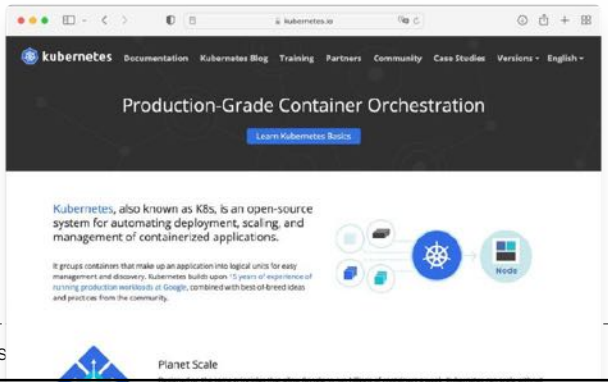
Agenda

4

1 Architekturstile und -prinzipien
2 Kleine Stilkunde der Softwarearchitektur
3 Einen Stil ausprägen
4 **Kubernetes und die Cloud**
5 Kein Fazit aber ein Fahrplan
6 Weitere Informationen

Was ist Kubernetes?

"Kubernetes (K8s) ist ein Open-Source-System zur Automatisierung der Verteilung, Skalierung und Verwaltung von containerisierten Anwendungen. Es gruppiert die Container, aus denen eine Anwendung besteht, in logische Einheiten zur einfachen Verwaltung und Auffindung …“



➔ https://kubernetes.io

K8s – Einige Fakten

- **Open Source**, Entwickelt in **Go**, verfügbar seit 2014
- **K8s** == K -ubernete- s (8 Buchstaben zwischen K und s)
- Initiiert von **Google**, das Erfahrung aus früheren, internen Projekten einfließen ließ („Borg“)
- Maintainer jetzt: **CNCF** (Cloud Native Computing Foundation)
- Sogenannte **Distributionen** “veredeln“ K8s, teilweise zu kommerziellen Produkten (Marketing: „K8s für Unternehmen“)

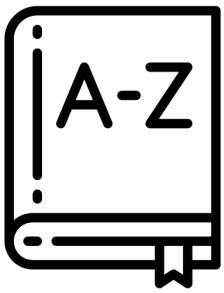
kubernetes

K8s – Wesentliche Aufgaben und Features

- Containerisierte Anwendungen **definieren**, **strukturieren** und **konfigurieren** (Stichwort *.yaml)
- Anwendungen **verteilen** und **aktualisieren** (Deployment, Updates)
- Zustand der Anwendung **überwachen** und bei Bedarf **heilen**
- **Lasten** über die Infrastruktur **verteilen** (Skalierung)
- **Kommunikation** zwischen Anwendungsteilen **leiten** und **absichern** (Traffic Routing)

Die 10 wichtigsten* Begriffe / Konzepte in K8s

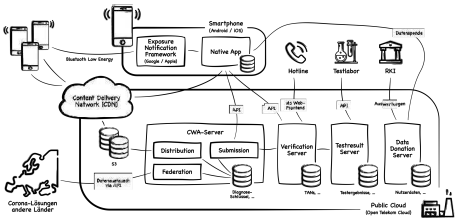
* meinungsbehaftet, aus Sicht eines Anwendungsentwicklers / einer -architektin



- Cluster
- Node
- Container
- Pod
- Volume
- Service
- Namespace
- Job
- Deployment (+ ReplicaSet)
- ConfigMap (+ Secret)

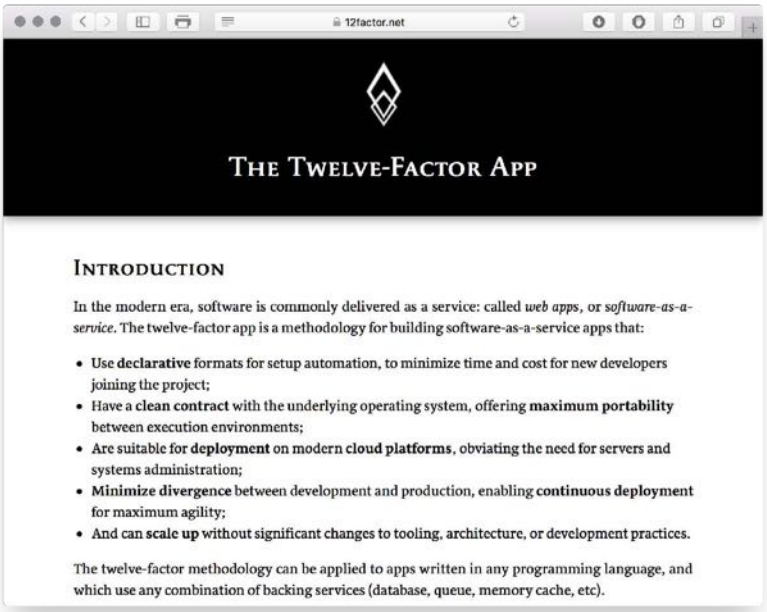
Zentrale K8s-Begriffe auf einem Bild

Wichtige Aufgabe: Begriffe in Beziehung setzen



Zerlegungsbegriffe Eures Architekturstils (z.B. Modul, Service, Teilsystem …)

Deployment-Begriffe Eurer Zielumgebung (hier z.B. Kubernetes)


12factor.net

THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

The Twelve-Factor App (2012)

”The twelve-factor app is a methodology for building software-as-a-service apps that:
…
- Are suitable for deployment on modern cloud platforms …”
- ...

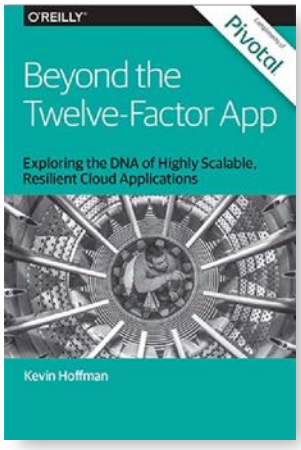
➔ 12factor.net

Ursprung:
Adam Wiggins, (Mit-)Gründer von Heroku.

“Heroku is a container-based cloud Platform as a Service (PaaS).”
https://www.heroku.com/about

heroku

“Beyond the Twelve-Factor App”



Kevin Hoffman
Beyond the Twelve-Factor App
Exploring the DNA of Highly Scalable, Resilient Cloud Applications

O’Reilly Media 2016
ISBN 978-1-491-94401-1

72 Seiten (PDF)

Wichtige Aspekte und Prinzipien

Die Graphik zeigt wichtige Themen und Facetten rund um die Anwendungsentwicklung in der Cloud und setzt sie in **Beziehung** zueinander.

Legende:

In lila Text: Unterstützende Methoden, Technologie-Beispiele.

TFA BTT

Nummer des Aspekts in den "Twelve Factor Apps" (TFA), bzw. "Beyond the TFA" (BTT).

Quelle: embarc Architektur-Spicker #5 „Cloudanwendungen“

Agenda

1 Architekturstile und -prinzipien
2 Kleine Stilkunde der Softwarearchitektur
3 Einen Stil ausprägen
4 Kubernetes und die Cloud
5 **Kein Fazit aber ein Fahrplan**
6 Weitere Informationen

5

Fahrplan in 6 Schritten

1 Architekturrelevante Anforderungen initial sammeln und verstehen

2 Architekturstil benennen und präzisieren

3 Zielsystem (technisch) skizzieren

4 Mapping zwischen Zerlegung und Technik herstellen

5 Makro- und Mikroarchitektur erkunden

6 Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

Nicht sklavisch beantworten. Die Fragen sollen Euch helfen, nicht behindern.

Zentrale Ergebnisse des Schrittes

Oftmals nur ein erster Wurf, den Ihr im weiteren Verlauf verfeinert, mitunter sogar widerlegt.

Fahrplan in 6 Schritten

1 **Architekturrelevante Anforderungen initial sammeln und verstehen**

2 Architekturstil benennen und präzisieren

3 Zielsystem (technisch) skizzieren

4 Mapping zwischen Zerlegung und Technik herstellen

5 Makro- und Mikroarchitektur erkunden

6 Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

- Was bauen wir?
- Was müssen wir dabei beachten?
- Was müssen wir besonders gut können?

Zentrale Ergebnisse des Schrittes

- Kontextabgrenzung
- Rahmenbedingungen
- Qualitätsziele

Fahrplan in 6 Schritten

1. Architekturrelevante Anforderungen initial sammeln und verstehen
2. **Architekturstil benennen und präzisieren**
3. Zielsystem (technisch) skizzieren
4. Mapping zwischen Zerlegung und Technik herstellen
5. Makro- und Mikroarchitektur erkunden
6. Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

- Welcher Stil unterstützt unsere Qualitätsziele?
- Wie heißen Elemente unserer Anwendung?
- Welche Regeln gelten für diese?

Zentrale Ergebnisse des Schrittes

- Prägender Stil
- Zerlegungsbegriffe und -prinzipien

Fahrplan in 6 Schritten

1. Architekturrelevante Anforderungen initial sammeln und verstehen
2. Architekturstil benennen und präzisieren
3. **Zielsystem (technisch) skizzieren**
4. Mapping zwischen Zerlegung und Technik herstellen
5. Makro- und Mikroarchitektur erkunden
6. Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

- Welche Plattform passt zu unseren Rahmenbedingungen und unserem Architekturstil?

Zentrale Ergebnisse des Schrittes

- Erstes Bild von Verteilung und Betrieb
- Grober Technologie-Stack

Fahrplan in 6 Schritten

1. Architekturrelevante Anforderungen initial sammeln und verstehen
2. Architekturstil benennen und präzisieren
3. Zielsystem (technisch) skizzieren
4. **Mapping zwischen Zerlegung und Technik herstellen**
5. Makro- und Mikroarchitektur erkunden
6. Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

- Wie passen unsere Zerlegungsbegriffe zur Nomenklatur des Zielsystems?

Zentrale Ergebnisse des Schrittes

- Zuordnung von Elementen zu Deployment-Begriffen

Fahrplan in 6 Schritten

1. Architekturrelevante Anforderungen initial sammeln und verstehen
2. Architekturstil benennen und präzisieren
3. Zielsystem (technisch) skizzieren
4. Mapping zwischen Zerlegung und Technik herstellen
5. **Makro- und Mikroarchitektur erkunden**
6. Zentrale Fragen der Makro-Architektur bearbeiten

Leitfragen für diesen Schritt

- Welche Aspekte sind für alle Elemente unserer Zerlegung gleich?
- Wo erlauben wir Freiheitsgrade?

Zentrale Ergebnisse des Schrittes

- Liste von Themen für die Makro-Architektur

Fahrplan in 6 Schritten

1. Architekturrelevante Anforderungen initial sammeln und verstehen
2. Architekturstil benennen und präzisieren
3. Zielsystem (technisch) skizzieren
4. Mapping zwischen Zerlegung und Technik herstellen
5. Makro- und Mikroarchitektur erkunden
6. **Zentrale Fragen der Makro-Architektur bearbeiten**

Leitfragen für diesen Schritt

- Wie beantworten wir die dringendsten Fragen zur Makro-Architektur?
- Welche Lösungen schlagen wir im Fall von Freiheitsgraden vor?

Zentrale Ergebnisse des Schrittes

- Übergreifende Konzepte
- Verfeinerter Technologie-Stack

Der Fahrplan auf einen Blick

Schritt	Leitfragen	Zentrale Ergebnisse
Architekturrelevante Anforderungen initial sammeln und verstehen	Was bauen wir? Was müssen wir dabei beachten? Was müssen wir besonders gut können?	Kontextabgrenzung Rahmenbedingungen Qualitätsziele
Architekturstil benennen und präzisieren	Welcher Stil unterstützt unsere Qualitätsziele? Wie heißen Elemente unserer Anwendung? Welche Regeln gelten für diese?	Prägender Stil Zerlegungsbegriffe und -prinzipien
Zielsystem (technisch) skizzieren	Welche Plattform passt zu unseren Rahmenbedingungen und unserem Architekturstil?	Erstes Bild von Verteilung und Betrieb Grober Technologie-Stack
Mapping zwischen Zerlegung und Technik herstellen	Wie passen unsere Zerlegungsbegriffe zur Nomenklatur des Zielsystems	Zuordnung von Elementen zu Deployment-Begriffen
Makro- und Mikroarchitektur erkunden	Welche Aspekte sind für alle Elemente unserer Zerlegung gleich? Wo erlauben wir Freiheitsgrade?	Liste von Themen für die Makro-Architektur
Zentrale Fragen der Makro-Architektur bearbeiten	Wie beantworten wir die dringendsten Fragen zur Makro-Architektur. Welche Lösungen schlagen wir im Fall von Freiheitsgraden vor?	Übergreifende Konzepte Verfeinerter Technologiestack

- Die Schritte sind dabei nicht sklavisch der Reihe nach abzuarbeiten.
- Das Ergebnis einer ersten Iteration ist eine erste Idee ("Candidate Architecture")

Ergebnis: Eure Architekturvision

Architekturvision

Fokus für den Projekterfolg kritische Architekturaspekte für die Entwicklung unumstößlich wichtiger Rahmen

Typische Inhalte

- ▶ **Systemkontext** (Abgrenzung)
- ▶ **Rahmenbedingungen**
- ▶ **Qualitätsanforderungen** (priorisiert)
- **Szenarien**
- **Risiken**

Was?

- ▶ **Technologien** (inkl. Plattform, Programmiersprache)
- ▶ **Architekturstil** (bestimmende Muster)
- ▶ **Architekturprinzipien**
- **Grobe Gliederung** (Systemteile auf oberster Ebene)
- **Domänenmodell** (Begrifflichkeiten der Fachlichkeit)

Wie?

(▶ . . . besonders wichtig)

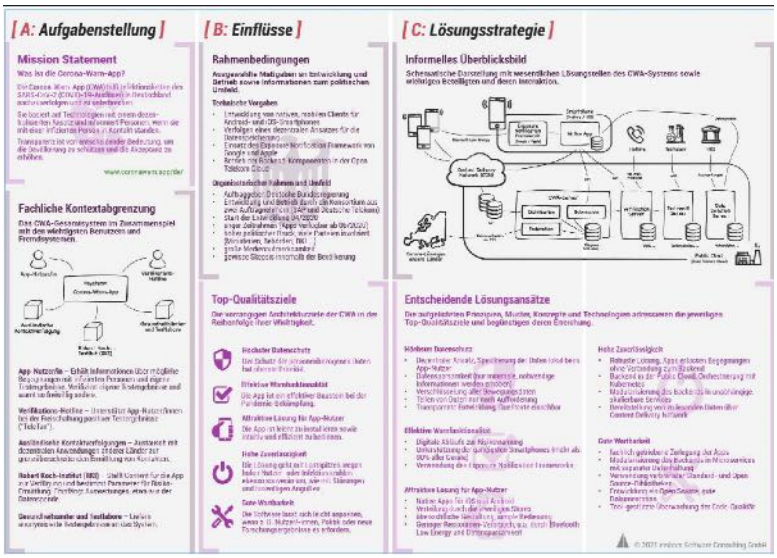
Als Idee, nicht als irreversible Entscheidung!

Agenda

1 Architekturstile und -prinzipien
2 Kleine Stilkunde der Softwarearchitektur
3 Einen Stil ausprägen
4 Kubernetes und die Cloud
5 Kein Fazit aber ein Fahrplan
6 **Weitere Informationen**

6

Überblicksflyer CWA, 1. Seite (innen)



Gibt es auch gedruckt!

Wir schicken Euch gerne ein gedrucktes Exemplar dieses Überblicks als Flyer im Treppenfalz zu! Einfach eine E-Mail senden an

info@embarc.de

mit Betreff „CWA-Flyer“ und Ihrer Postadresse im Text, dann geht das los …

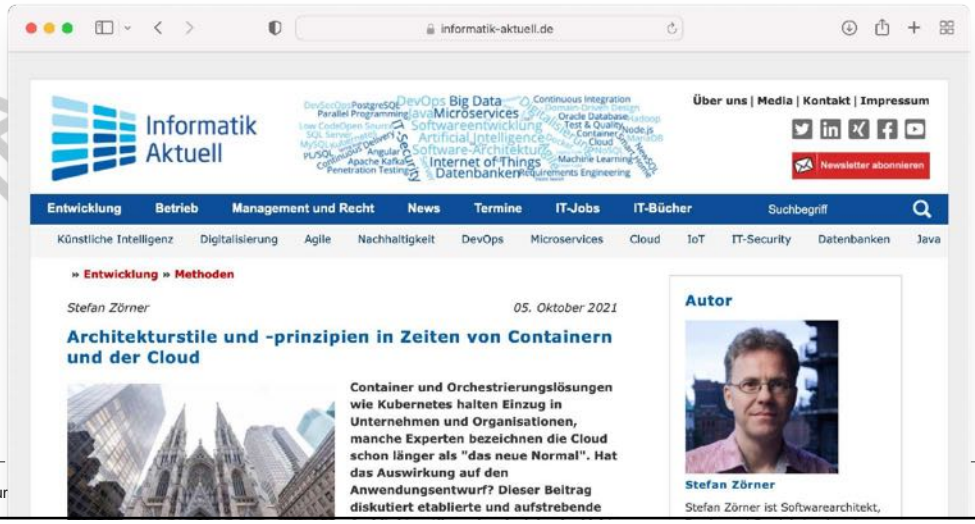
Als PDF auf unserer Seite …

➔ embarc.de/architektur-ueberblicke/

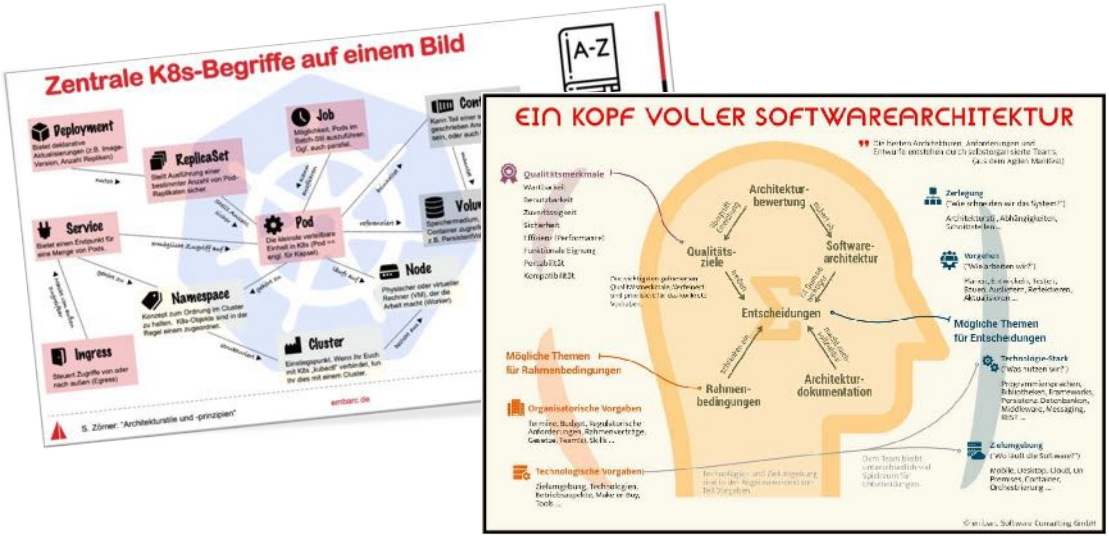
Architekturstile und –prinzipien …

Passender Artikel von mir zum Thema auf Informatik Aktuell

→ https://www.informatik-aktuell.de/entwicklung/methoden/architekturstile-und-prinzipien-in-zeiten-von-containern-und-der-cloud.html



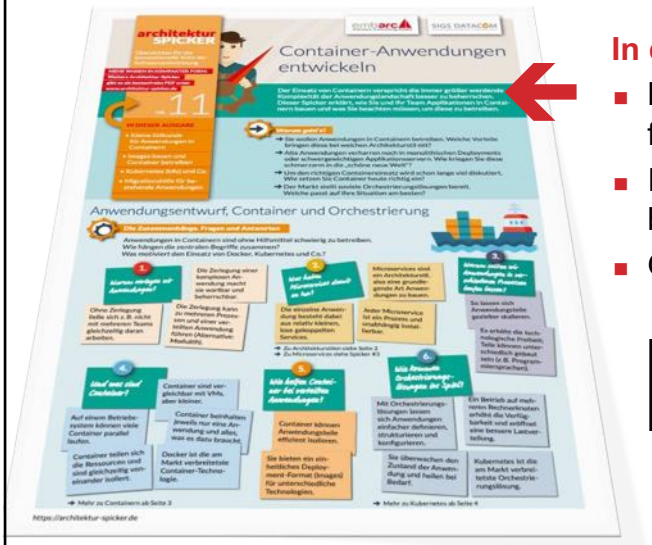
Concept Maps …



Blog-Beitrag zu Concept-Maps

➔ embarc.de/kopf-voller-softwarearchitektur-begriffsbild/

Spicker #11: „Container-Anwendungen entwicklen“



In dieser Ausgabe:

- Kleine Stilkunde für Anwendungen in Containern
- Images bauen und Container betreiben
- Orchestrierung: Kubernetes und Co.

PDF, 6 Seiten
Kostenloser Download.

➔ architektur-spicker.de

Vielen Dank.

Ich freue mich auf Eure Fragen!

sz@embarc.de
@StefanZoerner
➔ xing.to/szr

DOWNLOAD FOLIEN:
https://www.embarc.de/download/

embarc
Software Consulting GmbH

„Wir glauben weniger an ausgefeilte interne Strategien, als an die Möglichkeiten die entstehen wenn sich leidenschaftliche Leute zusammenschließen."

(Stefan Toth & Stefan Zörner, CEOs embarc)

Software-Architekt:innen gesucht

(Du entscheidest, wo Du arbeitest - flexibel in Deutschland)

Die Rolle der Softwarearchitekt:in ist in Unternehmen häufig unscharf definiert, so formulieren wir unsere Suchkriterien auch eher frei und freuen uns, wenn Du Dich hier wieder findest:

- Du hast ein Studium im Bereich Informatik abgeschlossen oder Du bist Quereinsteiger:in
- Du beschäftigst Dich gern mit Softwarearchitektur/-Systemen im Zusammenspiel mit aktuellen Trends (Mikroservices, Agilität, Machine Learning, ...) und
- Du hast mehrjährige Erfahrung in IT-Projekten.
- Außerdem brennst Du dafür, Kunden dabei zu helfen, sich selbst zu helfen und
- Du hast Lust darauf, Dich auch intern bei uns einzubringen?

Klingt interessant?
Sprich uns einfach direkt an oder melde Dich per Mail an **jobs@embarc.de**