Microservices

PROF. ANDREAS HARTMANN – VERTEILTE ANWENDUNGEN

Definitionen

Microservice

An individual microservice is a service that is implemented with a single purpose, that is self-contained, and independent of other instances and services. A microservice is a primary architectural building block of a Microservices Architecture.

Microservices Architecture (MSA)

Microservices Architecture (MSA) is a style of architecture that defines and creates systems through the use of small independent and self-contained services that align closely with business activities.

Quelle: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

Überblick nach [Quelle]

„…einzelne Programme, die als eigene Prozesse laufen“

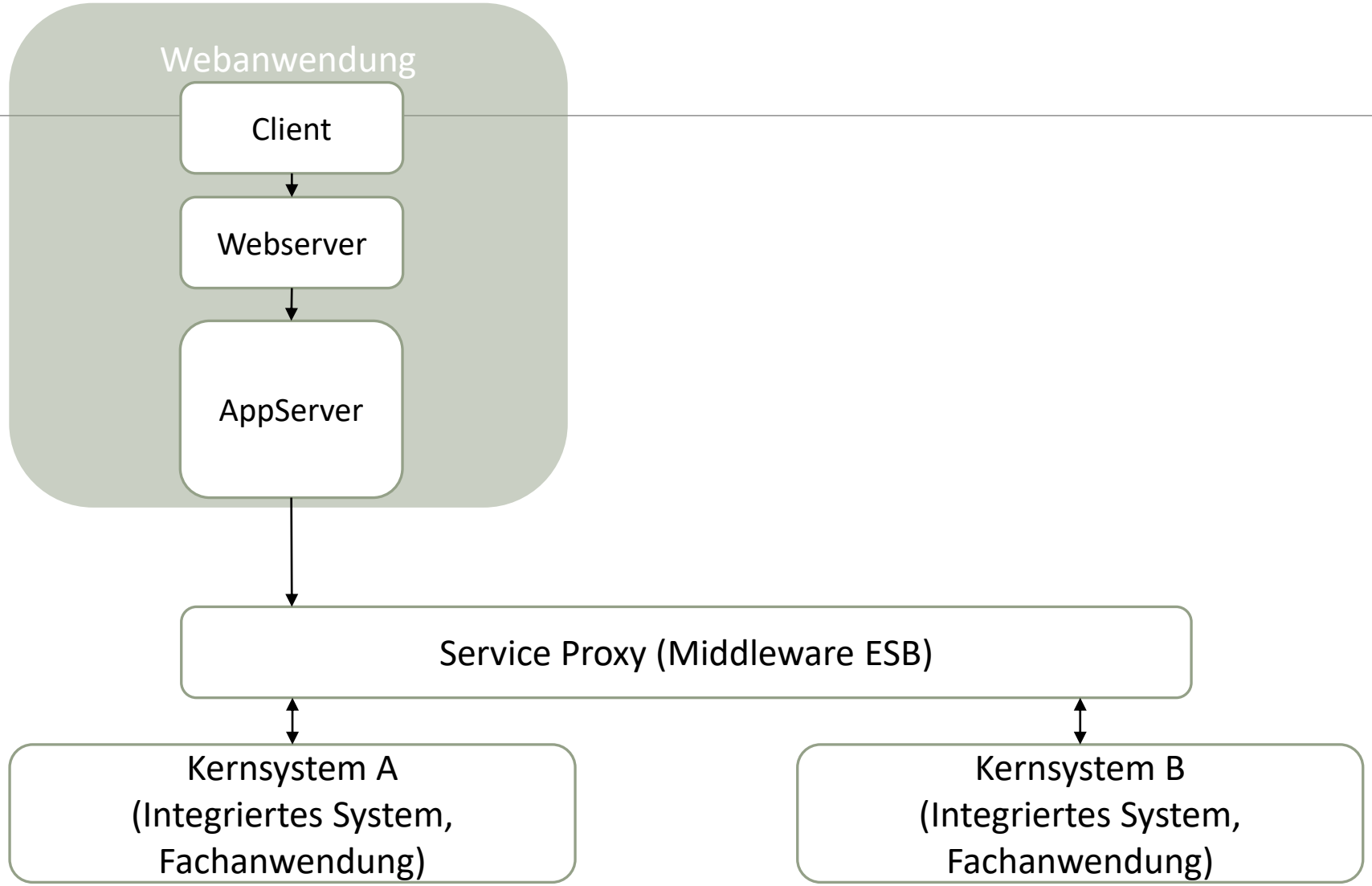
Entspricht Unix-Philosophie:

- Ein Programm soll nur eine Aufgabe erledigen, und das soll es gut machen.
- Programme sollen zusammen arbeiten können.
- Nutze eine universelle Schnittstelle. In UNIX sind das Textströme.

Trotz des Namens *Microservice* gibt es keine klare Definition betreffend Größe.

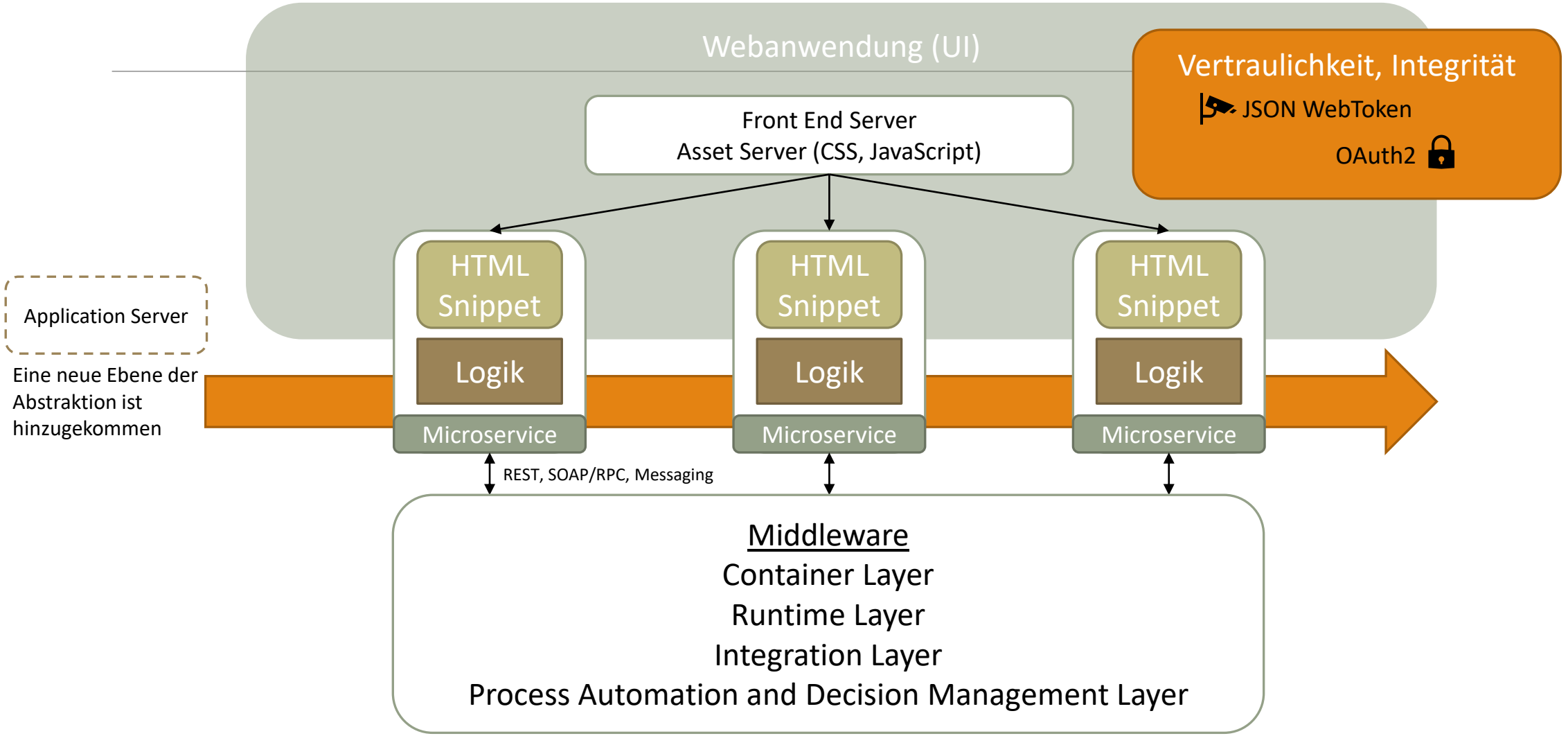
Quelle: E.Wolff. Das Microservices-Praxisbuch: Grundlagen, Konzepte und Rezepte. dpunkt.verlag GmbH, 2018

Microservices



Microservices

Unix: „Do one thing and do it well.“



Erste Hinweise zu den Eigenschaften

Modularisierungskonzept um große (komplexe) Softwaresysteme aufzuteilen und beherrschbar zu machen

Hat Auswirkungen auf Organisation und Software-Entwicklungsprozesse (vgl. DevOps)

Einzelne Microservices können unabhängig voneinander entwickelt und installiert werden

Es ist keine einheitliche Technologie vorgeschrieben (JavaScript, Python, …)

Daten können im Service gehalten werden oder einer gemeinsamen Datenbank

Microservices sind eigenständige Prozesse und kommunizieren über das Netzwerk (z.B. REST)

Der entgegenstehende Architekturstil ist der *Monolith*

Ziele des Architekturstils

Dazu später noch mehr!

- Starke Modularisierung
- Leichte Ersetzbarkeit
- Nachhaltige Software-Entwicklung
- Einfache Erweiterung von Legacy-Systemen
- Bessere Time-To-Market
- Unabhängige Skalierung
- Höhere Technologiefreiheiten
- Bessere Unterstützung von Continuous Delivery

Herausforderungen

Versteckte Beziehungen	Architektur besteht letztlich aus den (bekannten und unbekannten) Schnittstellen zwischen den Services
Schwieriges Refactoring	Funktionalitäten später schwer änderbar (als Architektur)
Hoher Aufwand in der Vorbereitung und Organisation	Fachliche Architektur muss erarbeitet und fortgeschrieben werden
Komplexer Betrieb	Hoher Automatisierungsgrad notwendig, Infrastruktur anspruchsvoll
Komplexität verteilter Systeme	Verteilte Systeme bringen Komplexität bzgl. Kommunikation und Nachrichtenaustausch

Ursprünge

Amazon 2006

- Für Amazon Webseite und Cloud-Plattform
- Die *Anwendung* wird in verschiedene Services aufgeteilt, da Monolith nicht mehr realisierbar
- Jeder Service liefert einen Teil der Webseite:
  - Suche,
  - Empfehlungen, …
- In der Nutzerschnittstelle (UI) wieder zusammengeführt, d.h. der Nutzer merkt nichts
- Je Service ist ein Team zuständig (Entwicklung & Betrieb!): *„you build it – you run it“*
- Gemeinsame Plattform auf Basis von virtuellen Maschinen: Cloud Plattform

Fun Fact: AWS ist entstanden, da Amazon unbenutzte Kapazitäten der eigenen Plattform vermietet hat.

Falle Komplexität

Ausfall der Kommunikation

- Verteiltes System muss nutzbar bleiben und auf Nutzeranfragen reagieren
- Technologien für Ausfallschutz notwendig (vgl. Kubernetes)
- ABER: auch fachliche Begutachtung notwendig (vgl. „…System steht gleich wieder zur Verfügung“)

Komplexe Infrastruktur

- Kann aus zahlreichen Schichten bestehen (Virtualisierung, Plattformen)
- Innerhalb einer Schicht/Technologie gibt es viele Freiheiten
- Abhängigkeiten sind jedoch komplex
- Mitarbeiter können oft nicht mehr alle Technologien überschauen -> ein Architekt wird benötigt
- Standardisierung immer noch ein wichtiges Thema!

Größe von Microservices

Teamgröße	Der Microservice muss von einem einzelnen Team beherrschbar sein. Ein Team kann aber an mehreren Microservices arbeiten.
Modularisierung	Der Microservice sollte nur so groß sein, dass ein Entwickler ihn verstehen und weiterentwickeln kann.
Ersetzbarkeit	Der Microservice muss leicht ersetzbar sein, was seine Größe beschränkt (vgl. Monolithen).
Infrastruktur	Der Microservice sollte mindestens so groß sein, dass sich die Infrastruktur dafür rentiert.
Kommunikation	Je mehr kleine Microservices existieren, um so größer wird die Komplexität der Kommunikation.
Konsistenz	Konsistenz und Transaktionen dürfen nicht mehrere Microservices umfassen.

Transaktionen - ACID

Atomizität	Transaktionen werden entweder ganz oder gar nicht ausgeführt.
	Bei einem Fehler werden alle Änderungen rückgängig gemacht.
Konsistenz (**C**onsistency)	Daten sind vor und nach der Änderung konsistent.
	Validierungen werden nicht verletzt.
Isolation	Transaktionen sind voneinander getrennt.
Dauerhaftigkeit (Durability)	Änderungen einer Transaktion werden gespeichert und stehen auch nach Abstürzen noch zur Verfügung.

Das Gesetz von Conway

CONWAY‘S LAW

MELVIN EDWARD CONWAY:

„Organisationen, die Systeme designen, können nur solche Designs entwerfen, welche die Kommunikationsstrukturen dieser Organisation abbilden.“

Microservices können daher zu einer neuen Organisationsform führen!

Quelle: https://en.wikipedia.org/wiki/Conway%27s_law

VARIATIONEN

RAYMOND: *„If you have four groups working on a compiler, you'll get a 4-pass compiler.”*

„If a group of N persons implements a COBOL compiler, there will be N−1 passes. Someone in the group has to be the manager.”

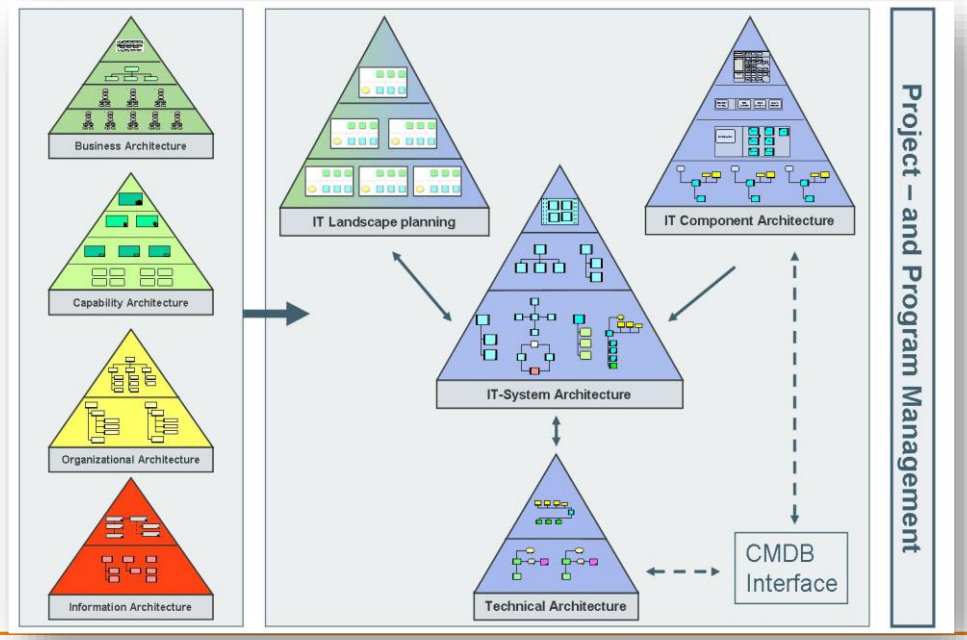
YOURDON AND CONSTANTINE: *„The structure of any system designed by an organization is isomorphic to the structure of the organization.”*

COBLIEN: *“If the parts of an organization (e.g., teams, departments, or subdivisions) do not closely reflect the essential parts of the product, or if the relationship between organizations do not reflect the relationships between product parts, then the project will be in trouble ... Therefore: Make sure the organization is compatible with the product architecture.”*

Fachliche Architektur

Wie sollen dann Microservices und Teams aufgeteilt werden?

Das führt zurück zur **Enterprise Architektur** (vgl. EAM) und einem Entwurfskonzept wie z.B. **Domain Driven Design** (DDD).



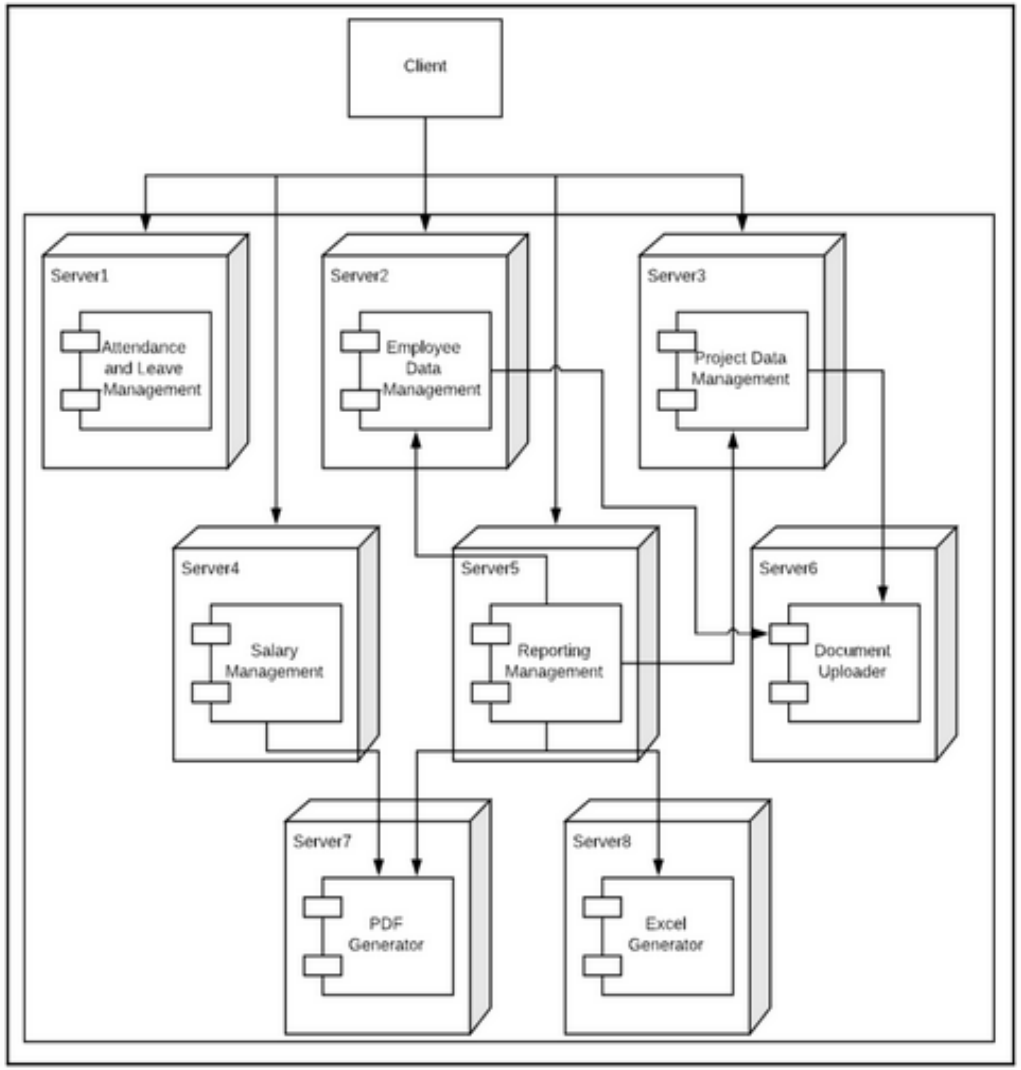
Quelle: ARIS IT-Architekt User Manual

Microservice Architektur (MSA)

QUELLE: TOG WHITE PAPER, DOCUMENT NO.:W169, PUBLISHED BY THE OPEN GROUP, JULY 2016.

Microservice-Architektur

Exemplarisches Beispiel einer Microservice-Architektur



Quelle: O. Mihályi, K. Singh, M. Caliskan. Java EE 8 Microservices. Packt Publishing, 2018

Definition *Architektur*

TOGAF 9.1 Standard

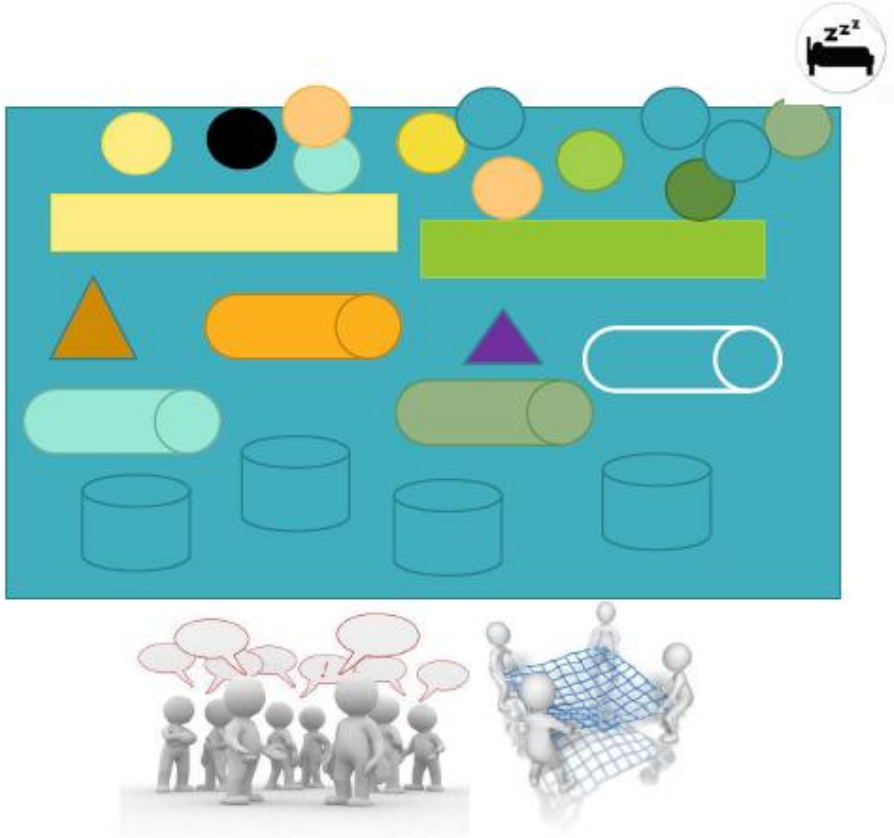
Architecture

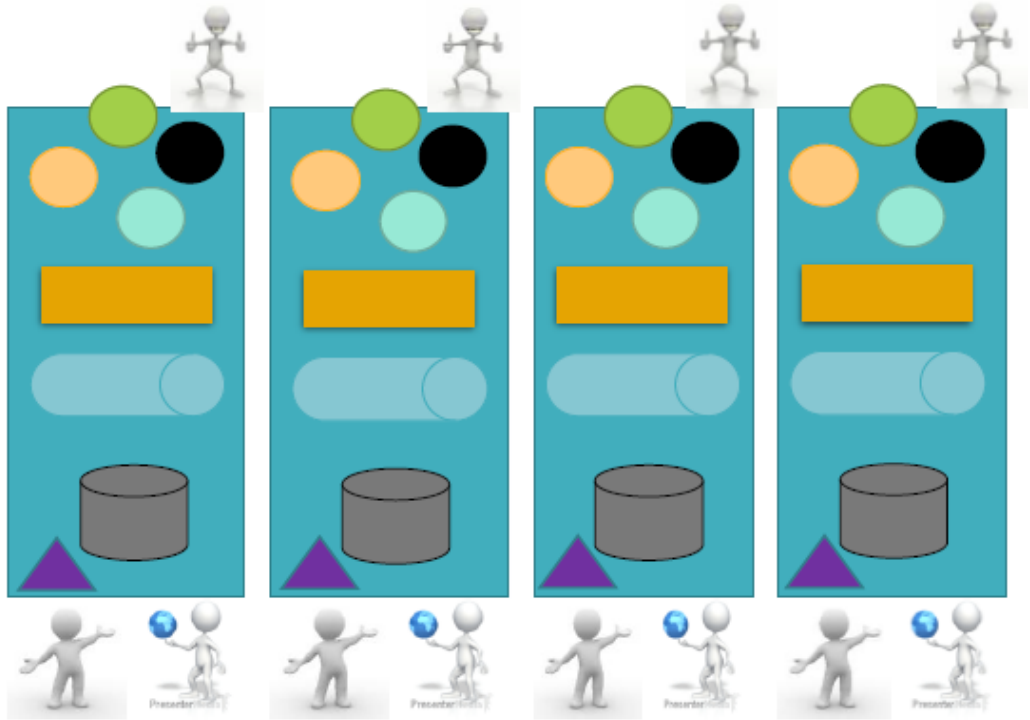
- *„the structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time”*

Architecture Style

- *“the combination of distinctive features in which architecture is performed or expressed”*

Microservice Architektur ist ein Architekturstil





Quelle: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

Eine MSA definiert und erzeugt Systeme durch kleine, unabhängige und „self-contained“ Services, die eng an die Geschäftsarchitektur und Geschäftsaktivitäten angelehnt sind.

Problembereich

Senken der Komplexität bzgl. Entwicklung, Betrieb und Management von Services

- Änderungen an Komponenten von vernetzten (abhängigen) Anwendungen können dazu führen, dass selbst für kleine Fixes ein großes Projekt folgt und die gesamte Lösung validiert werden muss.
- Im Fall von Monolithen muss die gesamte Software neu gebaut werden (build/compile).
- Das kann auch für SOA-basierten Anwendungen zutreffen.
- Die Modularität und saubere Architekturprinzipien tendieren dazu, über die Zeit abzunehmen und sich zu verschlechtern (z.B. hohe Kohäsion und lose Kopplung).
- Die Skalierung eines Monolithen oder von eng gekoppelten Komponenten erfordert immer eine Skalierung der gesamten Lösung – was teilweise unmöglich werden kann.

Simplifizierung zur Reduktion funktioneller Überladung

- Klassische Anwendungen neigen dazu, sich mit zusätzlichen Funktionen zu überladen.
- Das führt zu komplizierten Abhängigkeiten und einer gesamt steigenden Komplexität.

…Fortsetzung

MSA erlaubt schnellere Reaktionszeiten auf Änderungen des Marktes.

MSA nutzt die Vorteile moderner Paradigmen, wie CI/CD und DevOps.

MSA nutzt die Vorteile von Cloud-Konzepten oder die Lastverteilung zwischen On-Premise – Cloud.

MSA ist stark auf User Experience ausgerichtet.

Merkmale nach TOG

Softwarekomponenten werden auf unabhängige Services heruntergebrochen.

Die Services können unabhängig voneinander deployt werden.

Die Services sind auf atomare Geschäftsfähigkeiten (Capabilities) gemappt.

Die Services sind vollständig entkoppelt.

Durch die Verteilung und multiple parallele Instanzen der Services wird eine maximale Skalierbarkeit und Ausfallsicherheit erreicht.

Governance

In einer MSA sind die Teams dezentral aufgestellt bzw. verteilt.

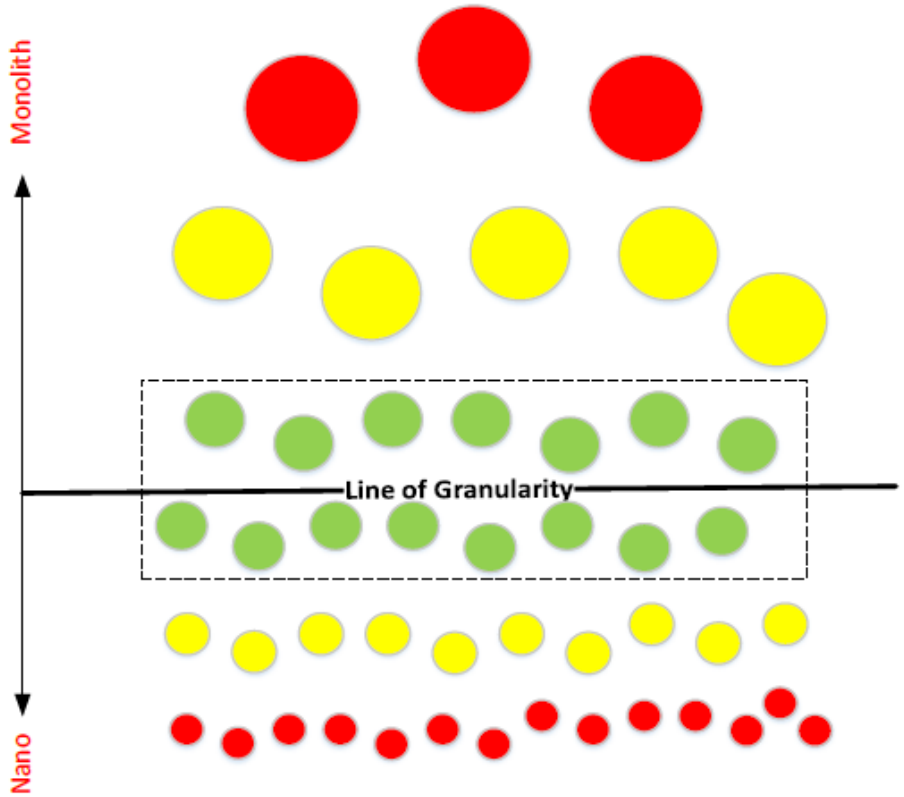
Daher können sich die Teams individuell auf die Technologie des Microservices einstellen.

Gleichzeitig müssen sich die Teams weniger (gar nicht) auf Abhängigkeiten zu anderen Services einstellen.

Das führt automatisch zu den Archetypen (vgl. IT Governance Weil & Ross):

- Feudal,
- Federal, oder
- *Anarchy*

Granularität



Quelle: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

Schlüsselprinzipien (1)

No.	Principle Name	Statement	Rationale	Implications
1	Independent Services	A microservice is independent of all other services.	Independence of services enables rapid service development and deployment, and permits scalability through instantiation of parallel, independent services. This characteristic also provides resilience; a microsservice is allowed to fail and its responsibilities are taken over by parallel instantiations (of the same microservice), which do not depend on other services. When a microservice fails, it does not bring down other services.	Both design and runtime independence of services are required. It is necessary for the business to determine whether providing scalability and resilience of the business function are paramount considerations. If so, MSA provides a means of achieving these characteristics.

Zitiert aus: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

Schlüsselprinzipien (2)

No.	Principle Name	Statement	Rationale	Implications
2	Single Responsibility	A microservice focuses on one task only and on doing it well. A microservice focuses on delivering a small specific business capability	This develops ideas including the principle of Single responsibility, Open-closed, Liskov substitution, Interface segregation, and Dependency inversion (SOLID), which is a tenet of Object-Oriented Design (OOD) and the core business domain and bounded context of DDD. Microservices are aligned to atomic business functions, which can be modified and deployed independently. To achieve this it is critical that each microservice caters to a single functional responsibility.	This requires business function decomposition into atomic functional services and data exchanges.

Zitiert aus: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

Schlüsselprinzipien (3)

No.	Principle Name	Statement	Rationale	Implications
3	Self-Containment	A microservice is a self-contained, independent deployable unit.	In order for a microservice to be independent, it needs to include all necessary building blocks for its operation, or there will be dependencies to external systems and services.	This has architecture, design, implementation, and deployment implications.

Zitiert aus: TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.

MSA versus SOA

www.opengroup.org/soa/source-book/soa/soa.htm ein Service ist:

- Eine logische Repräsentation einer wiederholbaren Geschäftsaktivität mit einem definierten Ergebnis
- In sich geschlossen (self-contained)
- Kann sich aus anderen Services zusammensetzen
- Ist eine „black box“ aus Sicht des Servicenutzers

SOA:

- Basiert auf dem Design von Services, die Geschäftsaktivitäten und schließlich Geschäftsprozesse reflektieren
- Service Repräsentation verwendet Geschäftsbeschreibungen um den Kontext bereitzustellen
- Services werden orchestriert
- Setzt eindeutige (unique) Anforderungen an die Infrastruktur
- Implementierung hängt von der Umgebung ab

MSA versus SOA (Fortsetzung)

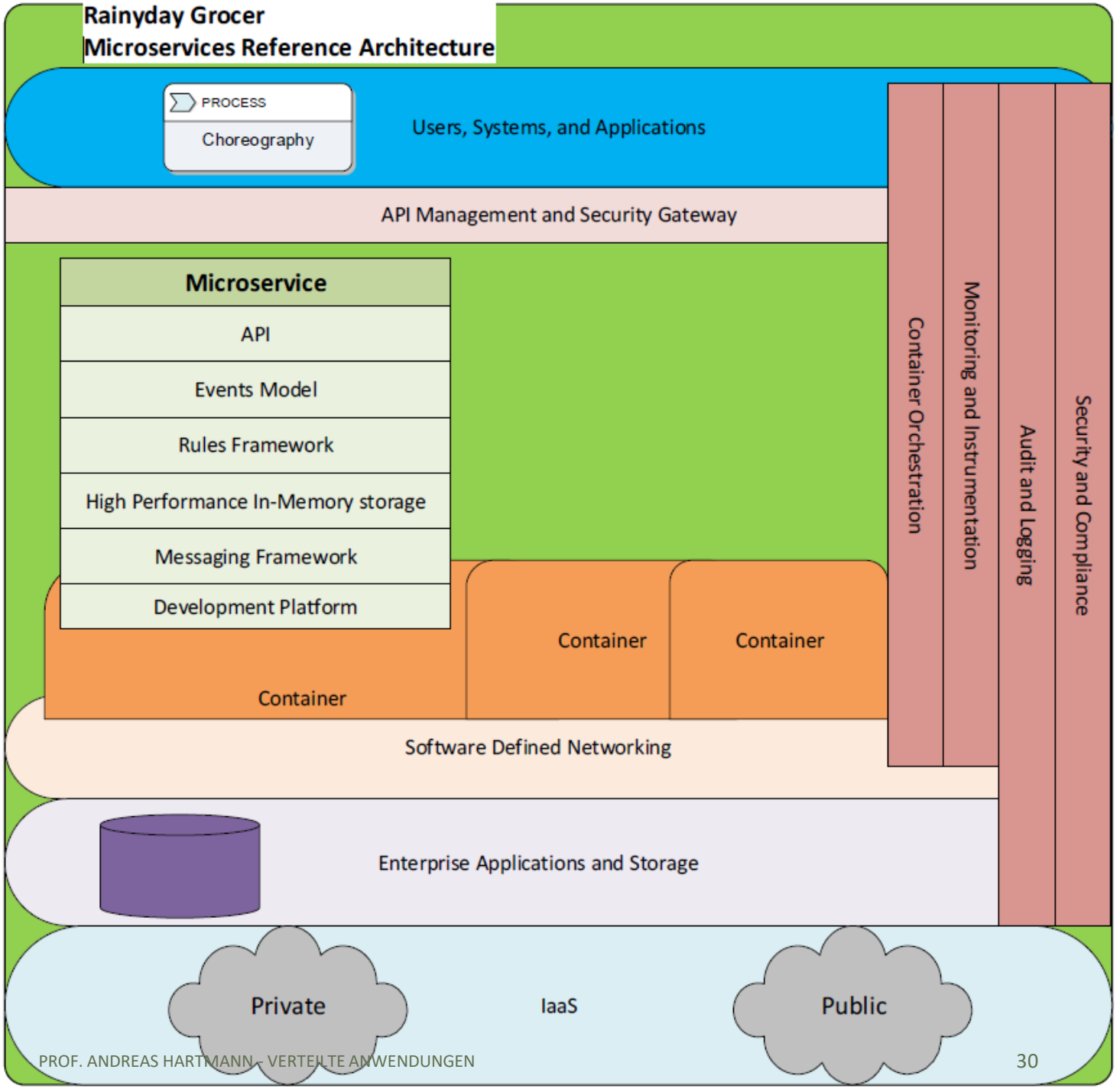
Daraus leitet sich ein wesentlicher Unterschied ab:

- MSA definiert unabhängige und in sich geschlossene Services, was impliziert, das Services nicht aus anderen Services zusammengesetzt sein können.

Daher kann MSA als eine spezielle (Unter-)Variante von SOA betrachtet werden.

Microservice Referenz-Architektur

Zitiert aus: *TOG White Paper, Document No.:W169, Published by The Open Group, July 2016.*



Zusammenfassung

Wir haben uns zwei Quellen für Microservices und MSA angesehen:

- Definition von Microservices
- Eigenschaften und Prinzipien
- Vorteile und Nachteile von Microservices
- Charakteristika einer MSA
- Ziele und Problembereiche einer MSA
- Vergleich zwischen SOA und MSA