Projekte. Beratung. Spezialisten.

IKS
Individuelle Softwarelösungen

Komplexität beherrschen

Micro, Nano, Mono? Microservices verständlich erklärt

IKS-Thementag

Christoph Schmidt-Casdorff

14.11.2017

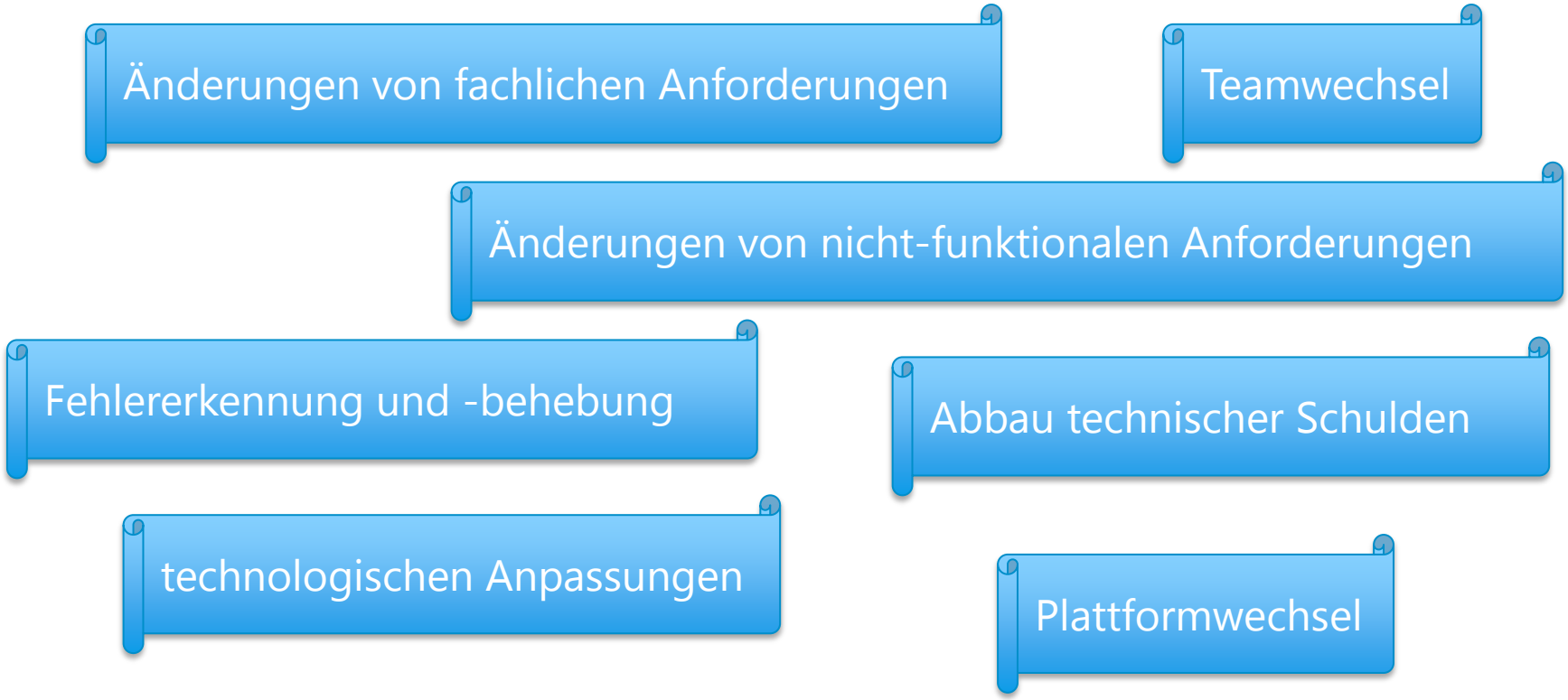
Agenda

- Softwaresysteme unter Veränderungen
- Was sind Microservices?
- Aspekte der Microservice-Architektur
- Zum Abschluss

Softwaresysteme unter Veränderungen

Auch Softwaresysteme altern

Treiber der Alterung von Softwaresystemen



Veränderungen über die Zeit lassen Softwaresysteme altern.

Das macht die Komplexität von Softwaresystemen aus.

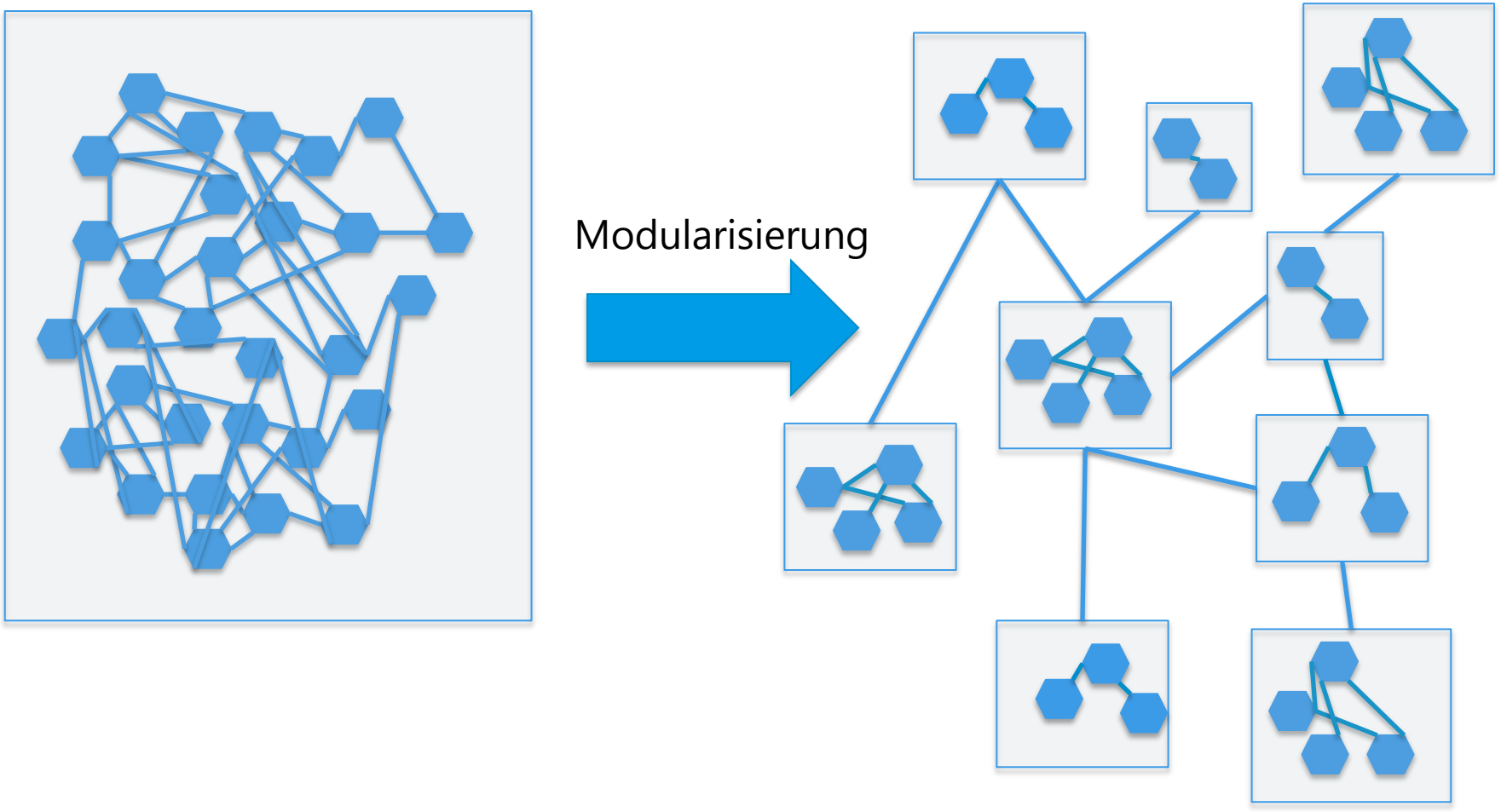
Mögliche Auswirkungen von Veränderungsprozessen

IKS Individuelle Softwarelösungen

- Es dauert immer länger, um Releases freizugeben
- Es ist fast nicht mehr möglich, neue Technologien zu integrieren
- Fehler im System nehmen von Release zu Release zu
- Fachliche Änderungen verstreuen sich über Ihre gesamte Anwendung
  - Oft ist gar nicht klar, welche Teile des Systems betroffen sind
- Kleine Änderungen haben große Auswirkungen
  - Z.B. aufwendige Abnahme des Gesamtsystems
- Ihre Datenbankstruktur ist unübersichtlich
  - Es ist nicht klar, welche Tabellen miteinander zu tun haben


Was sind Microservices?

Das sind Microservices!



Microservice(s)-Architektur

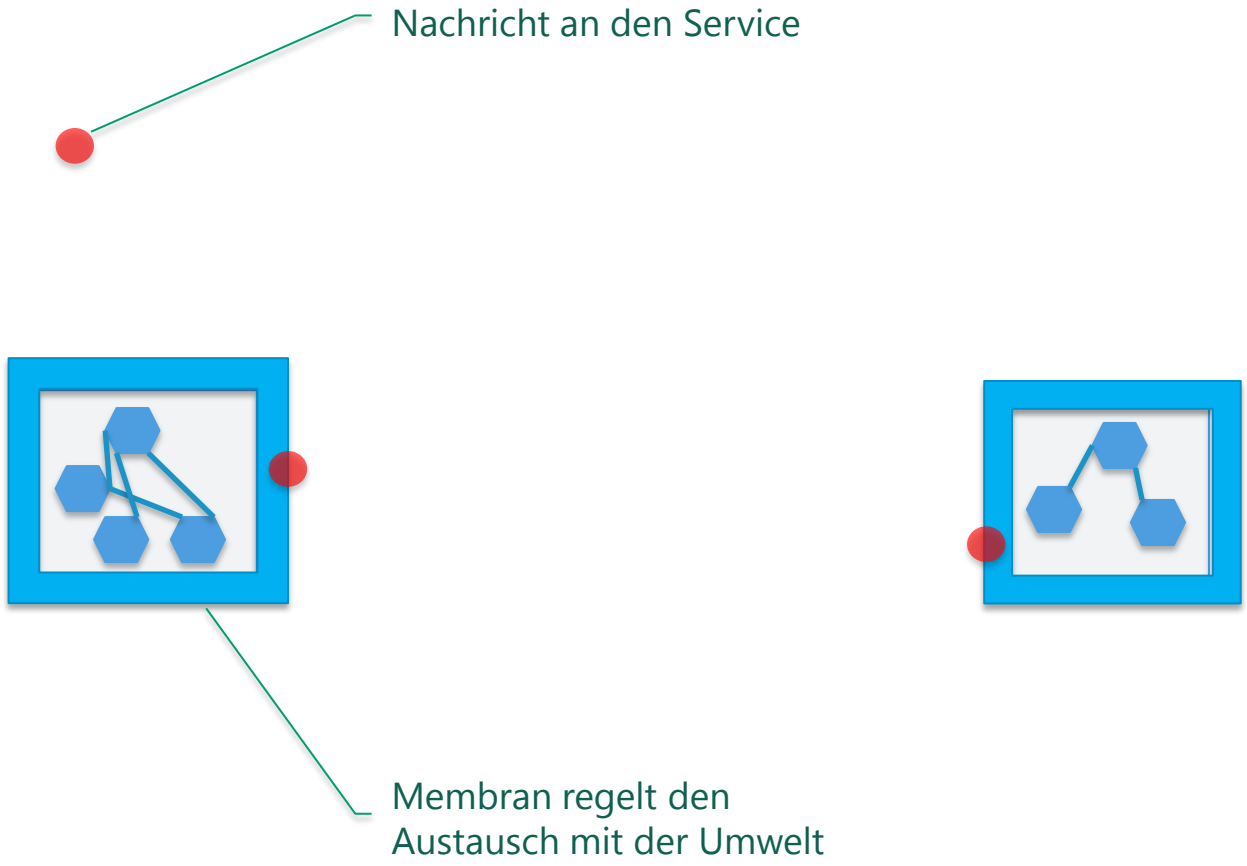
- beschreibt einen Architektur-Stil
  - Unabhängig von Technologien
  - D.h. Microservices = Microservice-Architektur
- unterstützt Evolution von Architektur in komplexen Systemen
  - Unterstützt die Änderung der Architektur (Eigendynamik) durch Modularisierung
  - Architektur ist so dynamisch, wie die Einflüsse auf das System
- Es gibt keine normierte Definition

Eigenschaften von Microservice-Architekturen[1)]

- Microservice-Architektur beschreibt ein System von lose gekoppelten Services, welche sich über leichtgewichtige Kommunikation verständigen.
  - Kommunikation nur zwischen Microservices
  - Das Innere der Microservices ist strikt von der Außenwelt isoliert
- Microservices sind unabhängig voneinander deploybar

1) Sehr häufig finden Sie eine Definition von Microservices über diese Eigenschaften [Fowler]

Membran des Microservice – Public Interfaces

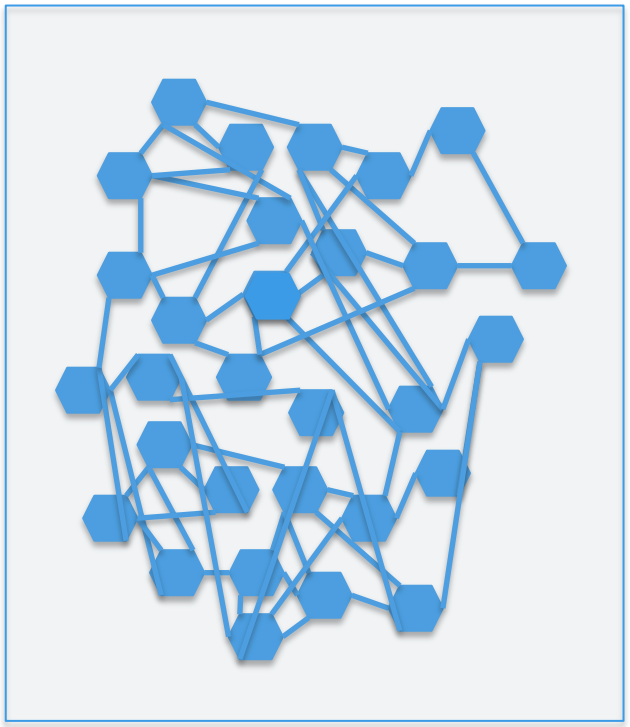


Membran des Microservice – Public Interfaces

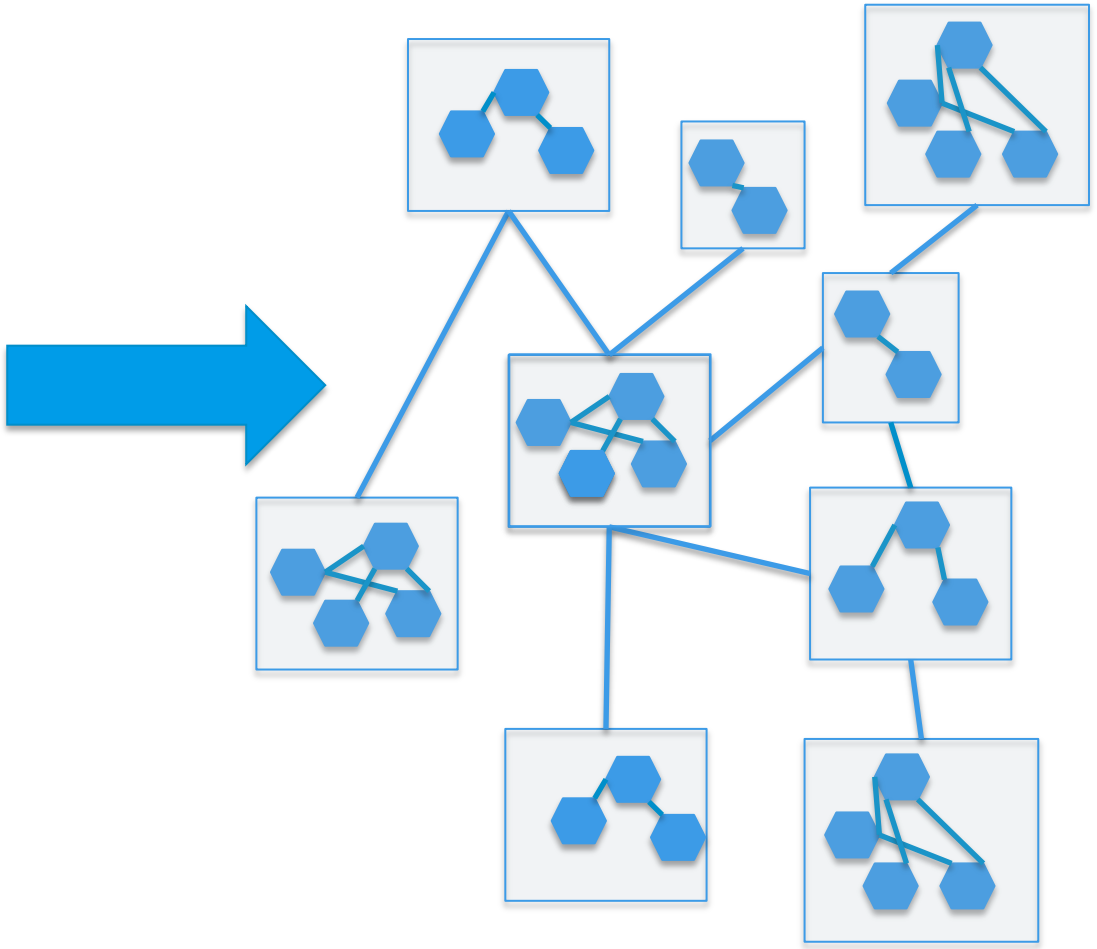
- Microservices bieten öffentliche Schnittstellen an
  - Kommunikation mit der Außenwelt nur über diese Schnittstellen
  - *Public Interfaces*
- Public Interfaces erlauben die kontrollierte Abschottung eines Services
  - Es wird nur sichtbar, was sichtbar sein soll
- Schnittstellen-Design muss auch evolutionär sein
  - Umgang mit Schnittstellenänderungen
  - Umgang mit Daten der Schnittstelle
- Fehler im Design der Schnittstelle sind teuer
- *REST over HTTP(S)* ist gängige Technologie für *Public Interfaces*
  - Es gibt noch weitere Technologien

Isoliertes Deployment gegen Dependency Hell

Monolith



Microservices



Aspekte der Microservice-Architektur

Es gibt nicht **die eine** Microservices-Architektur

Spannungsfelder der Microservices-Architektur

- Granularität – Größe von Microservices
- Kommunikation zwischen den Microservices
- Datenhaltung
- Steuerung der Zusammenarbeit der Microservices
- Technologische Autonomie der Microservices
- Deployment-Strategien

Granularität – Größe von Microservices

- Regel für die Größe eines Microservices
  - ~~100 – 1000 LOC findet man oft~~
  - ~~one/two Pizza Team~~
- Exakte Metrik für der Größe ist m.E. nicht zielführend
- Die Größe wird durch Verantwortlichkeit definiert

?

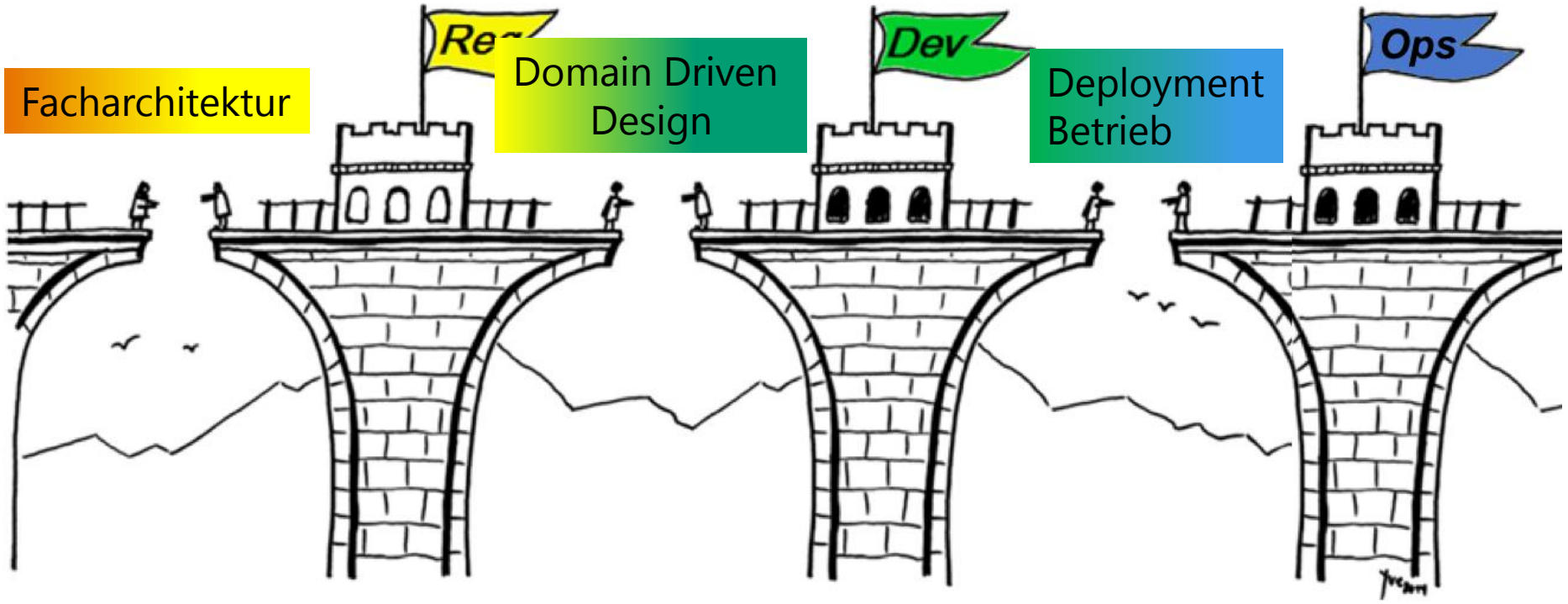
Wo kommen Services her?

- Services werden durch Geschäftsprozesse und (Fach-)Domänen bestimmt
  - Services werden durch das *Business* definiert
    - Stabilisiert das System
  - Microservices sind die technische Umsetzung von Geschäftsvorfällen
    - Grob, grob, grob ….
  - Facharchitektur muss die Services fachlich vorgegeben

- Zwischen Business und Entwicklung muss eine Brücke geschlagen sein
  - *Domain Driven Design (DDD)* liefert eine Methode, Microservices konsistent zu designen

- Microservices entstehen durch technische Adaption des DDD
  - Vereinfacht

Servicezentriert – Durchgängigkeit der Microservices



Services sind ein **durchgängiges** Konzept

- Stabil über alle Lebenszyklen eines Systems

Wo kommen Services her?

Falls Microservices **nicht** die **Fachlichkeiten** wiederspiegeln, ergeben sich die gleichen Probleme wie bei **Monolithen**.

Microservices **scheitern**, falls sie **nicht** durch eine **Facharchitektur** gestützt werden.

Services sind . . .

- kohärent
  - Ein Microservices ist für abgeschlossene, konsistente Menge an Funktionalität zuständig
  - Siehe auch *Single Responsibility Principle (SRP)*
- autonom
  - Die Erledigung seiner Aufgabe hängt nicht von anderen Services ab
  - Wir sehen später, welche Konsequenzen diese Forderungen haben können
- Dies sind Ziele, keine Gesetze
  - Können im Zweifel aufgeweicht werden
  - Aber nicht zu sehr ;-)

Granularität – Größe von Microservices

- Wird durch fachliche Verantwortlichkeit bestimmt
- Dennoch bleibt eine Bandbreite …

Große Microservices — Kleine Microservices

Lose Kopplung durch weniger Kommunikation
Starke innere Kopplung
Monolithische Tendenzen

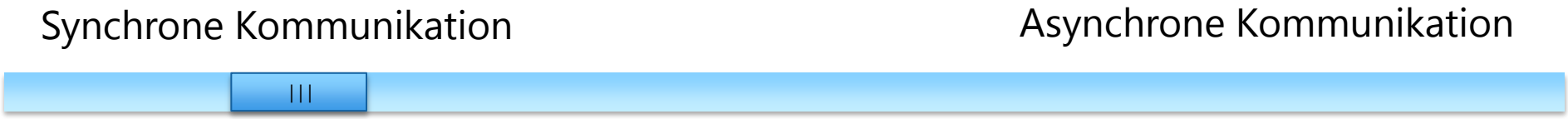
Enge Kopplung durch mehr Kommunikation
Überschaubar/verstehbar
Aufwendigeres Deployment

Kommunikation

- Microservices-Architektur führt zu verteilten Systemen
- => verteilte Kommunikation
- 7 irrige Annahmen über verteilte Systeme (7 fallacies of distributed Computing [wikipedia-2])
  - Netzwerk ist stabil und verlässlich
  - Es gibt keine Latenzzeiten
  - Netzwerk ist sicher und geschützt
  - . . .

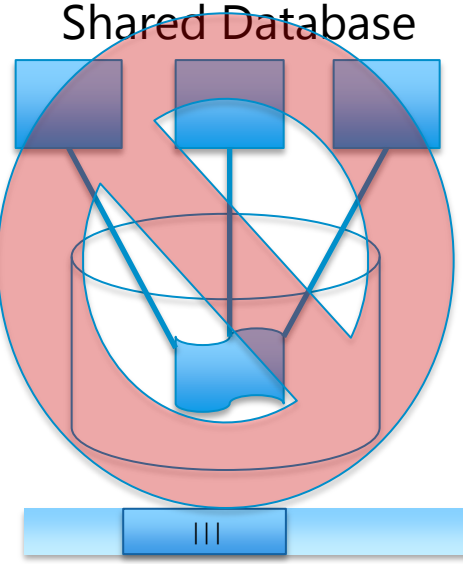
Kommunikation zwischen Microservices

- Synchrone Kommunikation
  - Synchrone Kommunikation wartet auf Antwort
    - Microservice hängt vom Antwortverhalten des Partnerservice ab
  - Erhöht die Kopplung der Microservices
  - Verringert Autonomie
  - Beispiel: REST
- Asynchrone Kommunikation
  - *Fire and Forget*
  - Beispiel: Messagequeues



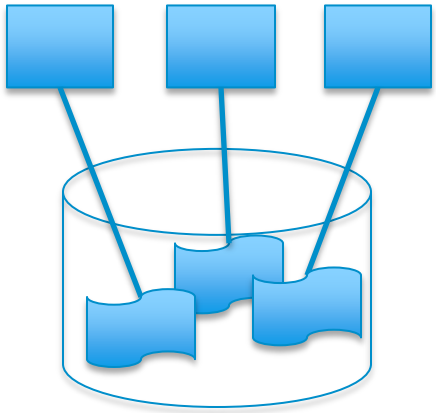
Datenhaltung

Shared Database



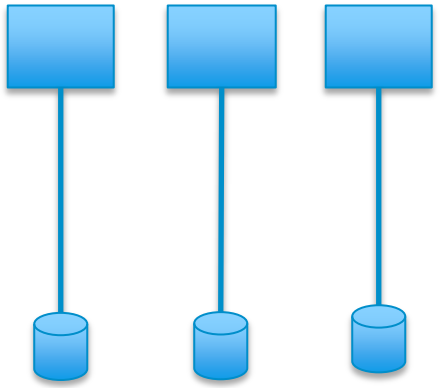
Enge und implizite Kopplung
Einfaches Programmiermodell
Anfällige Kohärenz

Shared Database System, Database per Service



Lose Kopplung
Komplexeres Programmiermodell
Stabile Kohärenz

Database System per Service



Lose Kopplung
Sehr gute Skalierbarkeit
Komplexes Betriebsmodell
Stabile Kohärenz

Steuerung der Zusammenarbeit

- Orchestrierung
  - Zentrale Instanz koordiniert die Zusammenarbeit der Services
  - Automatische Workflows (BPEL, BPMN)
  - Massive Kenntnis über Services <-> viel implizites Wissen der Services
- Choreographie
  - Interaktion unter Gleichgestellten
  - Keine zentrale Instanz
  - Koordination statt zentraler Kontrolle

Orchestrierung

Choreographie

Zentrale Kontrolle
Synchrone Kommunikation
Enge Kopplung

Koordination unter Gleichgestellten
Asynchrone Kommunikation möglich
Lose Kopplung

Technologische Autonomie

IKS
Individuelle Softwarelösungen

- Isoliertes Deployment verbessert die technologische Autonomie
  - Kommunikationsverfahren müssen immer zwischen Microservices abgestimmt werden

- Fragestellungen:
  - Wie viele Technologien kann meine Entwicklung vertragen/beherrschen?
  - Wie viele verschiedene Datenbanksysteme kann mein Betrieb beherrschen?
  - Wie viel Wert lege ich auf Abbau technischer Schulden pro Microservice?

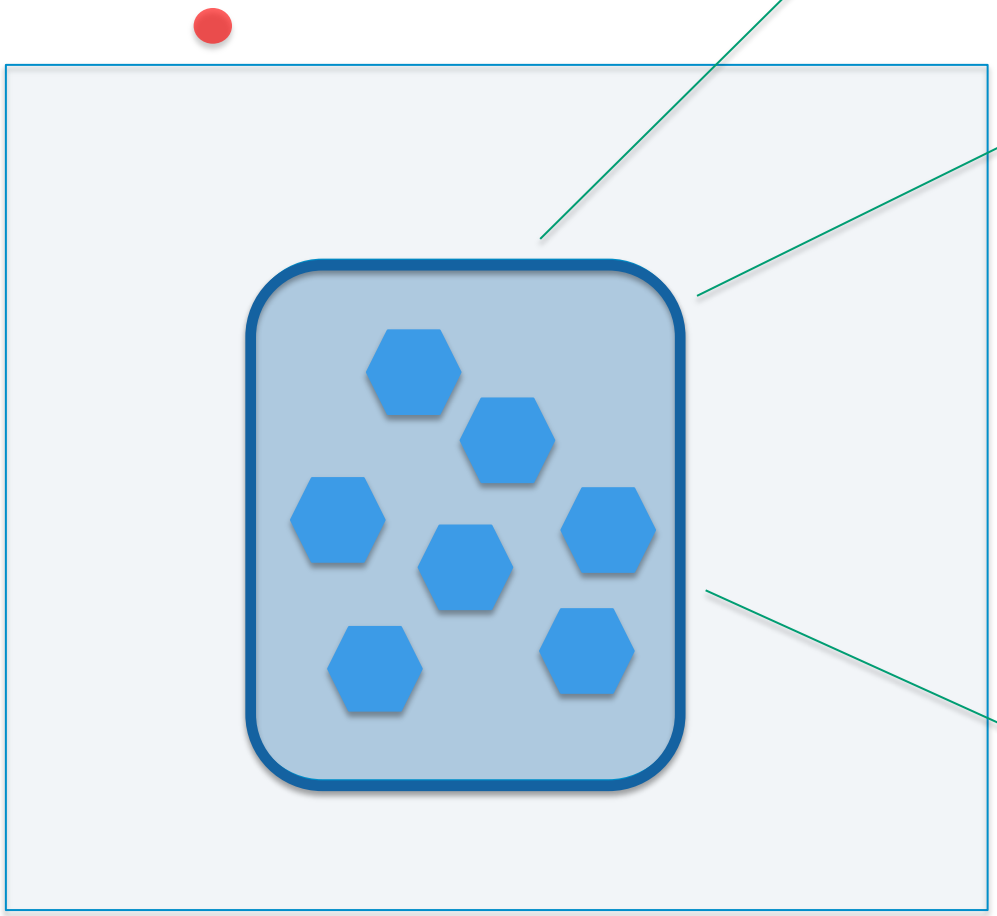
Enge technologische Rahmenbedingungen

Technologische Autonomie des Services

Enge technologische Kopplung
Technologische Koordination
Kleiner Zoo an Technologien

Lose technologische Kopplung
Wenig technologische Koordination
Umfangreicher Zoo an Technologien
Abbau technischer Schulden pro Service

Deployment-Strategien – Mehrere Service-Instanzen per Host

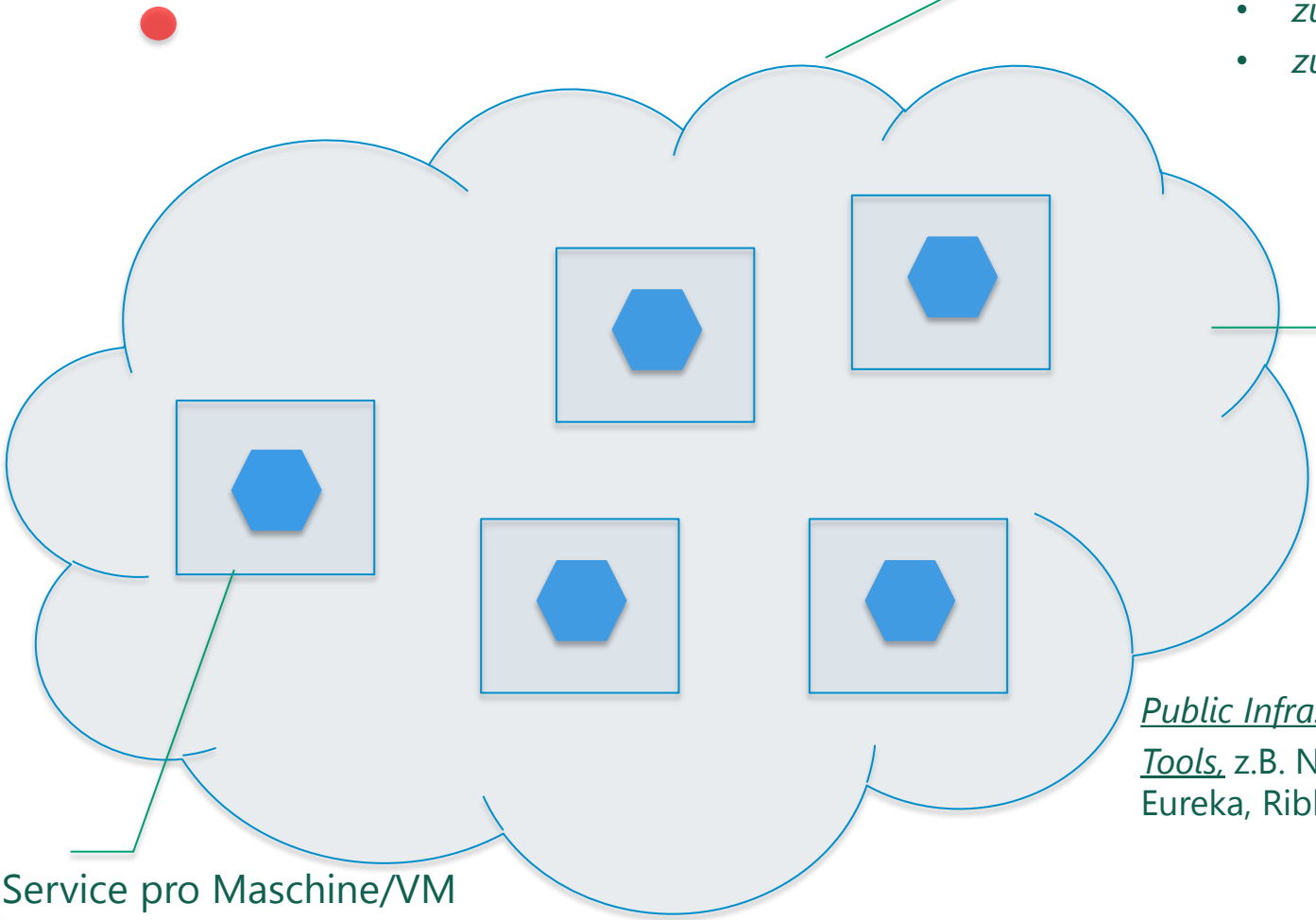


Managed Laufzeitumgebung für Services

z.B. ApplicationServer, OSGi, ESB

Ist in der Regel ein Prozess

Deployment-Strategien – Service Instanz pro Virtueller Maschine



Infrastruktur, um Services

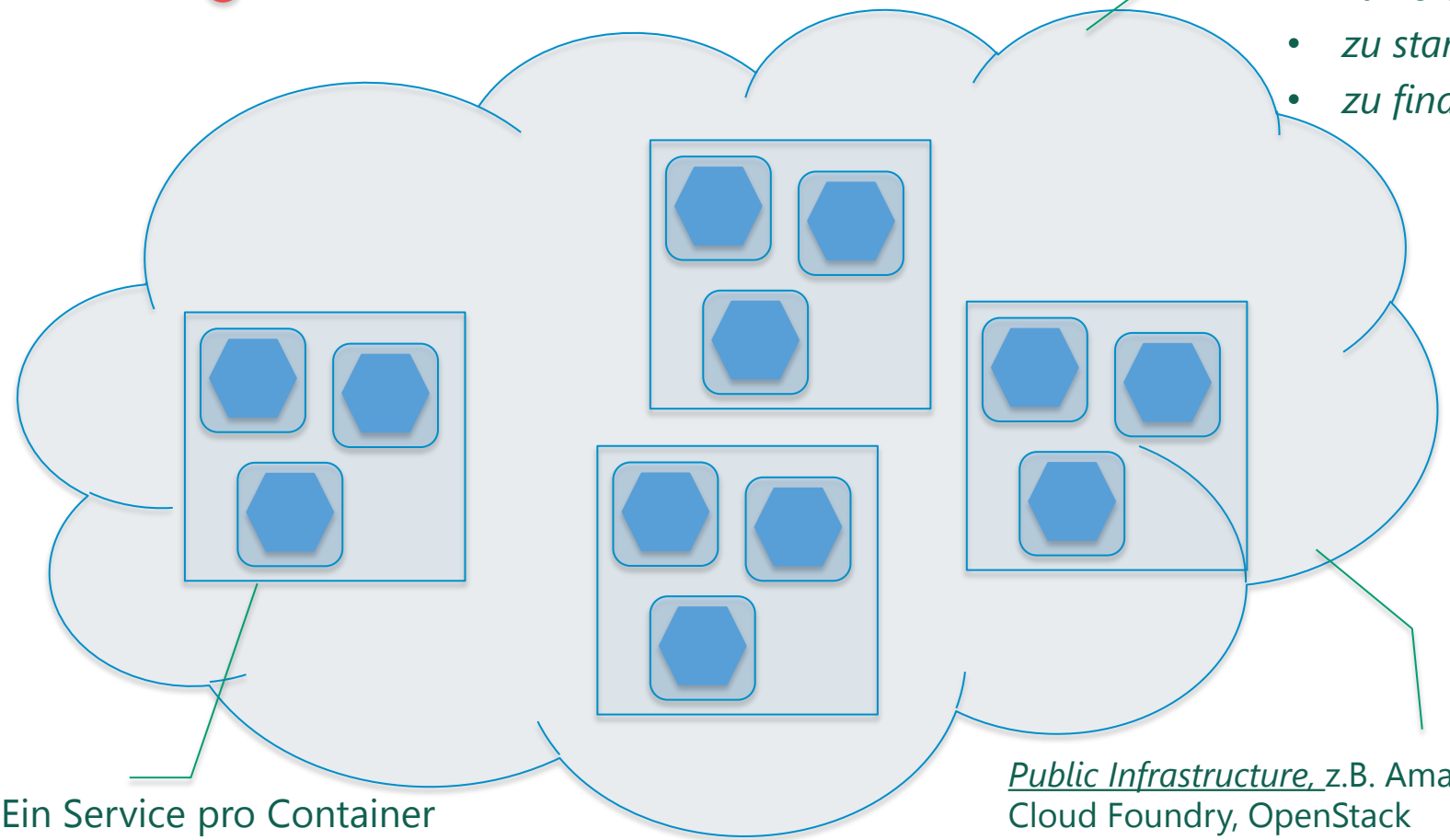
- *zu verteilen*
- *zu starten*
- *zu finden*

Public Infrastructure, z.B. Amazon EC2

Tools, z.B. Netflix Stack mit Eureka, Ribbon, Zuul,...

Ein Service pro Maschine/VM

Deployment-Strategien – Service Instanz pro Container



Infrastruktur, um Container

- *zu verteilen*
- *zu starten*
- *zu finden*

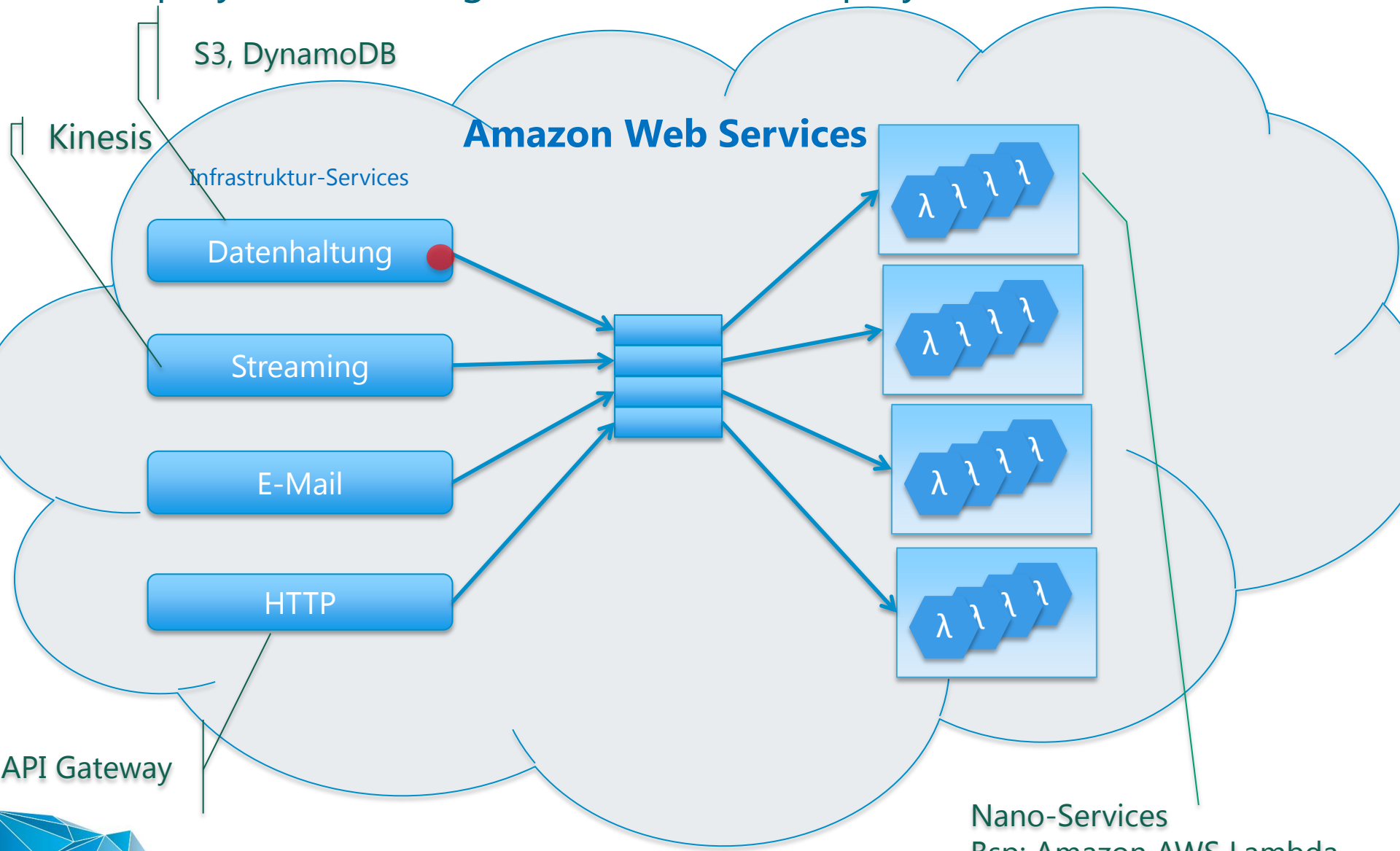
Ein Service pro Container

Mehrere Container pro Host/VM

Public Infrastructure, z.B. Amazon EC2, GAE Cloud Foundry, OpenStack

Tools, z.B. Docker Engine, Kubernetes, Apache Mesos, . . .

Deployment-Strategien – Serverless Deployment



Deployment-Szenarien

Mehrere Service-Instanzen
pro Host/Maschine (JEE)

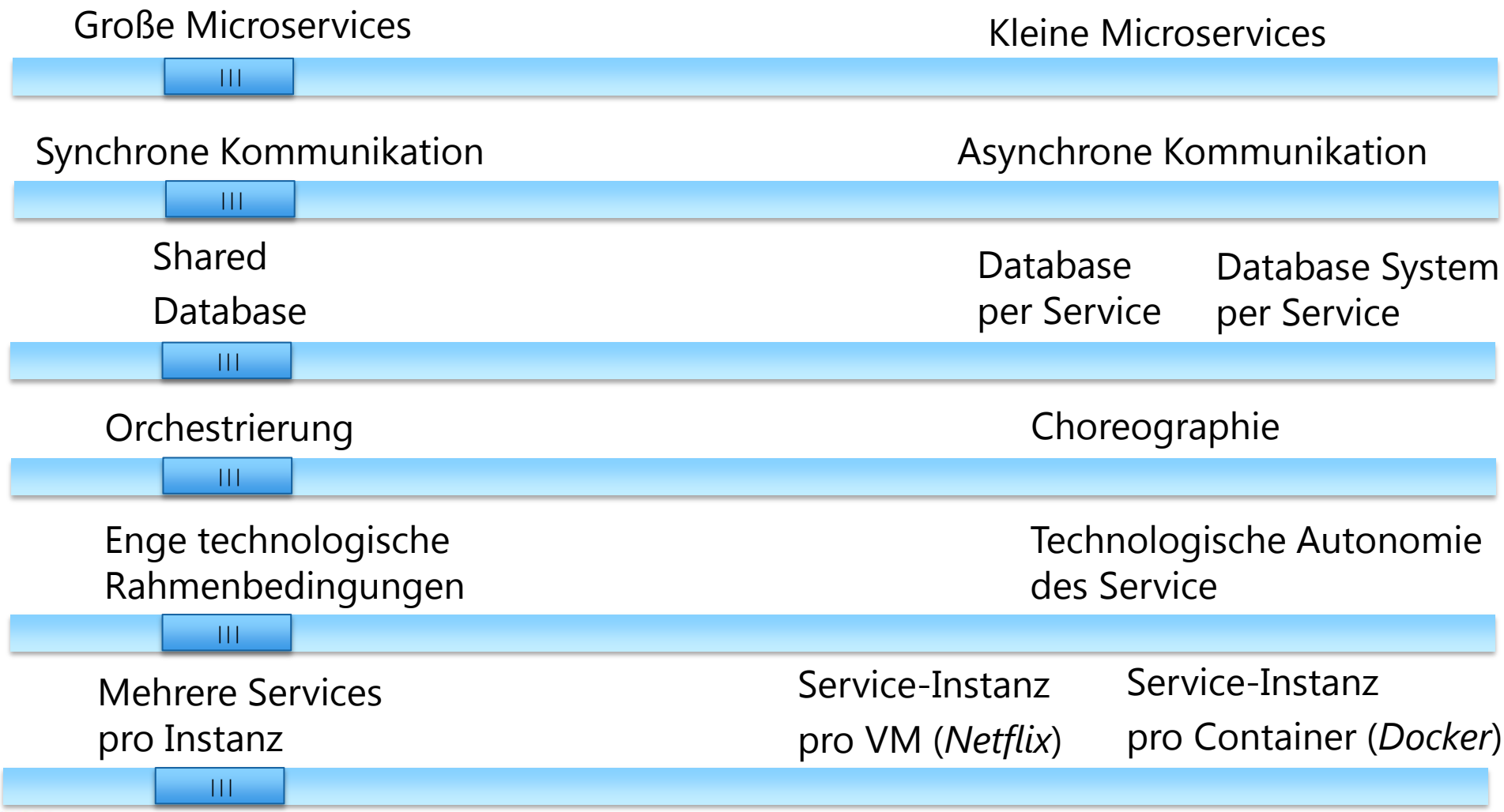
Service-Instanz
pro VM (*Netflix*)

Service-Instanz
pro Container (*Docker*)

Nicht für Skalierung optimierte Infrastruktur
Klassisches Deployment
Nicht isolierte Laufzeitumgebung
Nicht automatisierte Infrastruktur
Geht ohne DevOps

Für Skalierung optimierte Infrastruktur
Isolierte Laufzeitumgebung
Automatisierte Infrastruktur
Geht nicht ohne DevOps

Jede Domäne ist anders



Weitere Herausforderungen

- Logging und Monitoring
  - Je mehr Beteiligte, desto entscheidender ist Logging und Monitoring
  - Informationen aller Microservices müssen zentral zusammengeführt werden
    - Sammeln, speichern, suchen und aufbereiten
  - Jeder Microservice integriert sich selbst

Weitere Herausforderungen

- Logging und Monitoring

- Security
  - Autorisierung und Authentifizierung der Kommunikation
  - SSO + Loadbalancing
  - Delegationsverfahren wie OAuth2, ….

Weitere Herausforderungen

- Logging und Monitoring
- Security
- Testverfahren
  - Testverfahren auf Ebene der Microservices
  - Consumer-Driven-Tests

Weitere Herausforderungen

- Logging und Monitoring
- Security
- Testverfahren
- Fehlertoleranz und Resilienz – *design for failure*
  - Absicherung gegen Nicht-Erreichbarkeiten
  - Implementiert jeder Microservice für sich

Jede Domäne ist anders

- Es gibt nicht **die eine** Microservice-Architektur

- Ihre Microservices-Architektur muss
  - Ihre Probleme lösen
  - Ihren Anforderungen entsprechen
  - Ihre Gegebenheiten respektieren


Zum Abschluss

Microservices . . .

- sind ein Architekturstil
  - Unterstützen evolutionäre Architektur
- werden durch fachliche Geschäftsvorfälle bestimmt
- Architektur ist individuell
  - Muss an Anforderungen und Gegebenheiten Ihres Unternehmens angepasst werden
- müssen in einem System bewusst und konsequent angewendet werden
  - Systeme neigen zu impliziten Kopplungen
- sind an keine Technologie gebunden

Microservices . . .

- basieren auf einem übergreifenden, durchgängigen fachlichen Servicekonzept
  - Ohne dieses durchgängige Konzept sind Microservices nur sehr schwer zu beherrschen
- spielen ihre Stärken und Potential voll aus, wenn sie
  - durch agile Methoden unterstützt werden
  - durch automatisierte Infrastruktur unterstützt werden (*DevOps*)
  - durch passende Organisationsstrukturen unterstützt werden

Vielen Dank!
Fragen?

Referenzen

- [Fowler] http://martinfowler.com/articles/microservices.html
- [schlemm] A. Schlemm http://www.thur.de/philo/som/somkomplex.htm
- [schoeneberg] Komplexitätsmanagement in Unternehmen; Schoeneberg http://www.springer.com/de/book/9783658012830
- [Peinl] Überblick über Docker-Cluster-Technologien - Peinl 2016 (SIGS-DATACOM) http://www.sigs-datacom.de/uploads/tx_dmjournals/peinl_OTS_Microservices_Docker_16.pdf
- [SOA-Manifest] http://soa-manifest.de/
- [RAML] http://raml.org/
- [*wikipedia-1*] https://de.wikipedia.org/wiki/Komplexit%C3%A4t
- [wikipedia-2] https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing
- [wikipedia-3] https://de.wikipedia.org/wiki/Cynefin-Framework

Weiterführende Literatur

- https://blog.iks-gmbh.com/was-ist-eine-microservice-architektur/
- http://samnewman.io/blog/2015/04/07/microservices-for-greenfield/
- https://genehughson.wordpress.com/2015/05/18/microservices-sharpening-the-focus/
- https://genehughson.wordpress.com/2014/05/23/carving-it-up-microservices-monoliths-conways-law/
- http://simplearchitectures.blogspot.de/2012/09/snowman-architecture-part-one-overview.html
- https://genehughson.wordpress.com/2014/11/24/microservice-principles-and-enterprise-it-architecture/
- https://genehughson.wordpress.com/2014/06/04/more-on-microservices-boundaries-governance-reuse-complexity/
- https://www.tigerteam.dk/2014/micro-services-its-not-only-the-size-that-matters-its-also-how-you-use-them-part-1/ Teile 1-6
- http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p15.pdf

Bildreferenzen

- https://pixabay.com/de/fischernetze-fischernetz-fischerei-101992/
- https://pixabay.com/de/bienen-bienenstock-imkerei-honig-486872/
- https://pixabay.com/de/bienenwabe-honigwabe-honig-lecker-1564956/
- https://pixabay.com/de/blumenwiese-wiesenblumen-sommerwiese-1657016/
- https://pixabay.com/de/himmel-wolken-sonnenstrahlen-414198/
- https://pixabay.com/de/sonne-abendrot-morgenrot-209495/
- https://pixabay.com/de/adler-vogel-raubvogel-greifvogel-339125/

Impulsvorträge für Ihr Unternehmen

IKS
Individuelle Softwarelösungen

- Überblick über das gesamte Angebot an Impulsvorträgen unter: www.iks-gmbh.com/impulsvotraege

- Ihr Nutzen:
  - Unabhängiges, aktuelles Expertenwissen.
  - Individuell auf Ihr Publikum und Ihr Unternehmen zugeschnittene Vorträge.
  - Referenten mit langjähriger und branchenübergreifender Expertise in der IT-Beratung.
  - Praxisnahe Vorträge, die aus Projektarbeit entstanden sind, frei von Produktwerbung.
  - Ideale Ergänzung für Ihre Führungskräftetreffen, Abteilungsmeetings, Hausmessen, Innovation Days, Konferenzen, Open Spaces, Kick-off-Meetings oder Zukunftsworkshops.

IKS
Individuelle Softwarelösungen

WWW.IKS-GMBH.COM

Projekte. Beratung. Spezialisten.

IKS
Individuelle Softwarelösungen

IKS Gesellschaft für Informations-und Kommunikationssysteme mbH

T. +49 2103-5872-0 | www.iks-gmbh.com