Cloud Native Architecture

FWP Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

Was bringt uns die Cloud?

MOTIVATION

Cloud als gemeinsamer Erfolgsfaktor

Diese ***Cloud Native*** Firmen zeigen die gleichen Charakteristiken:

- Hohe Innovationsgeschwindigkeit
- Permanent verfügbare Dienste
- Web Scale IT (Gartner)
- Optimale User Experience auf mobilen Endgeräten

Think Big! - In der Cloud ist alles unbegrenzt

Unbegrenztes Datenvolumen

- Big Data
- Hohe Verfügbarkeit durch weltweite Replikation
- Hohe Sicherheit durch redundante Abspeicherung

Unbegrenzte Rechenpower

- Rechenpower mit automatischer Lastanpassung
- Verarbeitung beliebig großer Datenmengen (auch in Echtzeit)
- Nur die genutzte Rechenleistung wird verrechnet

Unbegrenzte Bandbreite

- Gerüstet für Connected Services mit Millionen von Endgeräten
- Bereit für das Internet der Dinge (IoT)
- Dynamische Anpassung der Bandbreite in Selbstbedienung

Act Fast ! & Be Safe !

Agile Infrastruktur

- Provisionierung beliebiger Infrastruktur in Selbstbedienung
- Kurzfristiger Auf- und Abbau möglich
- Hohe Automatisierung (Infrastructure as Code)
- Abrechnung im Stundentakt

Eingebaute Sicherheit

- Hohe Ausfallsicherheit durch Verteilung auf mehrere Standorte
- Abschottung durch virtuelle private Netzwerke
- Permanente Sichtbarkeit des aktuellen Systemzustandes
- Robustheit durch Redundanz
- Private Direktverbindungen

Aber in welche Cloud soll ich gehen?

Hybrid Cloud / Multi Cloud

- Höchste Flexibilität
- Auswahl des besten Providers für die spezifische Anforderung (Bandbreite, Kosten, Office 365)
- Portabilität durch Cross-Cloud-Plattformen wie **Cloud Foundry** oder **OpenShift** möglich

On Premise / Private Cloud

- Höchste Sicherheit
- Höchste Kontrolle
- Beschränkte Skalierbarkeit und Bandbreite
- Eingeschränkte Verteilung auf Standorte (nur wo eigene Datacenter vorhanden)

Off Premise / Public Cloud

- Ausreichende Sicherheit (AWS DE, Azure DE)
- Ausreichende Kontrolle
- Unbeschränkte Skalierbarkeit und Bandbreite
- Weltweite Verteilung auf Standorte
- Anbindung an Firmennetze möglich (Express Route, Direct Connect)

Was für eine Architektur brauchen unsere Applikationen für die Cloud?

CLOUD NATIVE ARCHITECTURE FÜR CLOUD NATIVE APPLIKATIONEN

Cloud Native Landscape

v2.1

Cloud Native Computing Foundation Landscape

Greyed logos are not open source

App Definition & Development

Database & Data Warehouse

Vitess (CNCF Incubating)

Streaming

NATS (CNCF Incubating)

Source Code Management

Application Definition

Continuous Integration / Continuous Delivery (CI/CD)

Orchestration & Management

Scheduling & Orchestration

kubernetes (CNCF Graduated)

Coordination & Service Discovery

CoreDNS (CNCF Incubating)

Service Management

GRPC (CNCF Incubating)

envoy (CNCF Incubating)

LINKERD (CNCF Sandbox)

Runtime

Cloud-Native Storage

ROOK (CNCF Sandbox)

Container Runtime

containerd (CNCF Incubating)

rkt (CNCF Incubating)

Cloud-Native Network

CNI (CNCF Incubating)

Provisioning

Host Management / Tooling

Infrastructure Automation

Container Registries

Secure Images

Notary (CNCF Incubating)

TUF (CNCF Incubating)

Key Management

Cloud

Public

Private

Platforms

Certified Kubernetes - Distribution

Certified Kubernetes - Platform

Non-Certified Kubernetes

PaaS/Container Service

Observability & Analysis

Monitoring

Prometheus (CNCF Incubating)

Logging

fluentd (CNCF Incubating)

Tracing

OPENTRACING (CNCF Incubating)

JAEGER (CNCF Incubating)

Serverless

See the separate serverless landscape

Special

Kubernetes Certified Service Provider

github.com/cncf/landscape

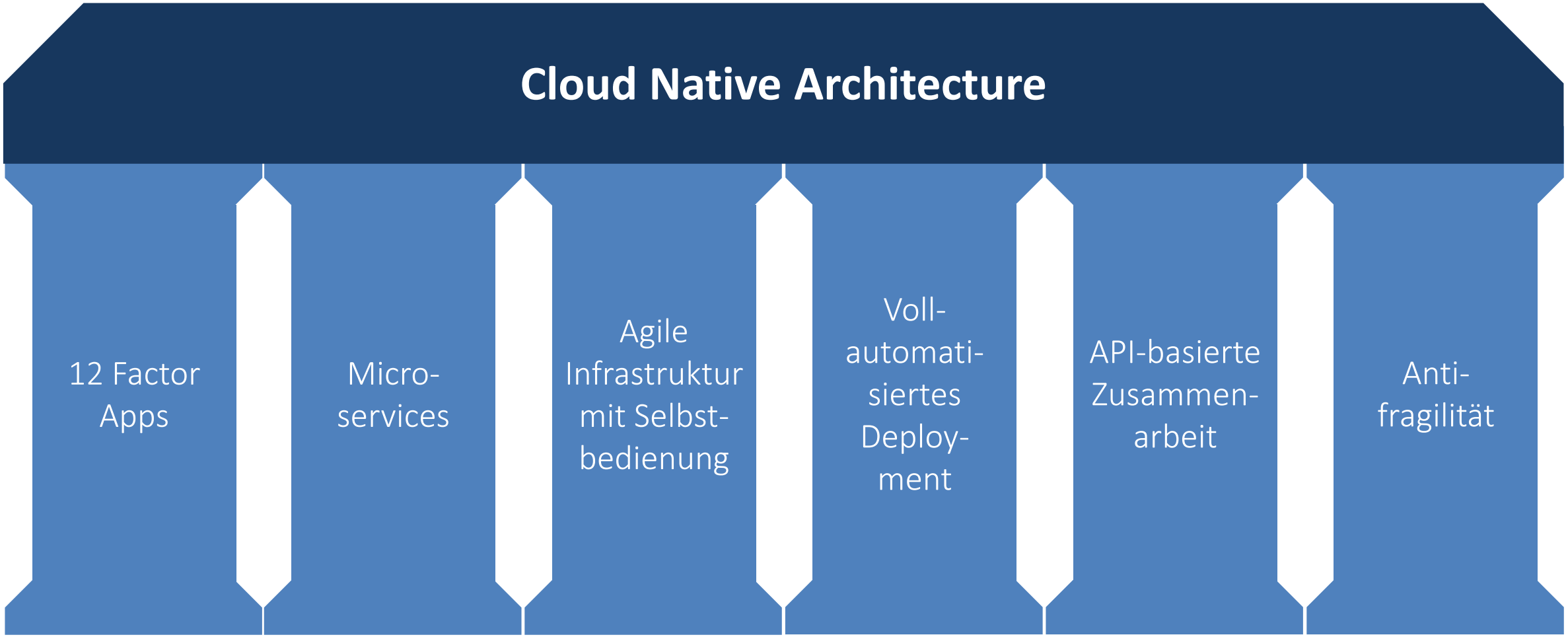
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

CLOUD NATIVE Landscape

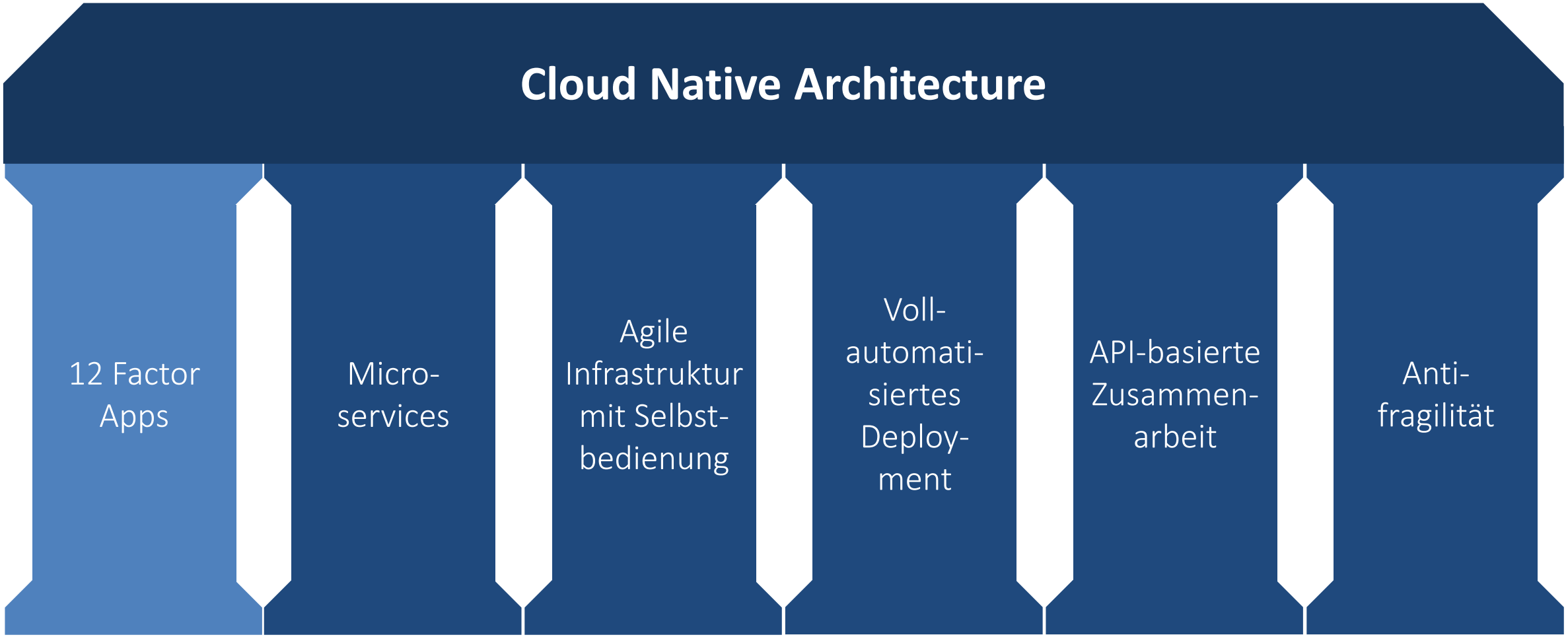
CLOUD NATIVE COMPUTING FOUNDATION

Redpoint Amplify PARTNERS

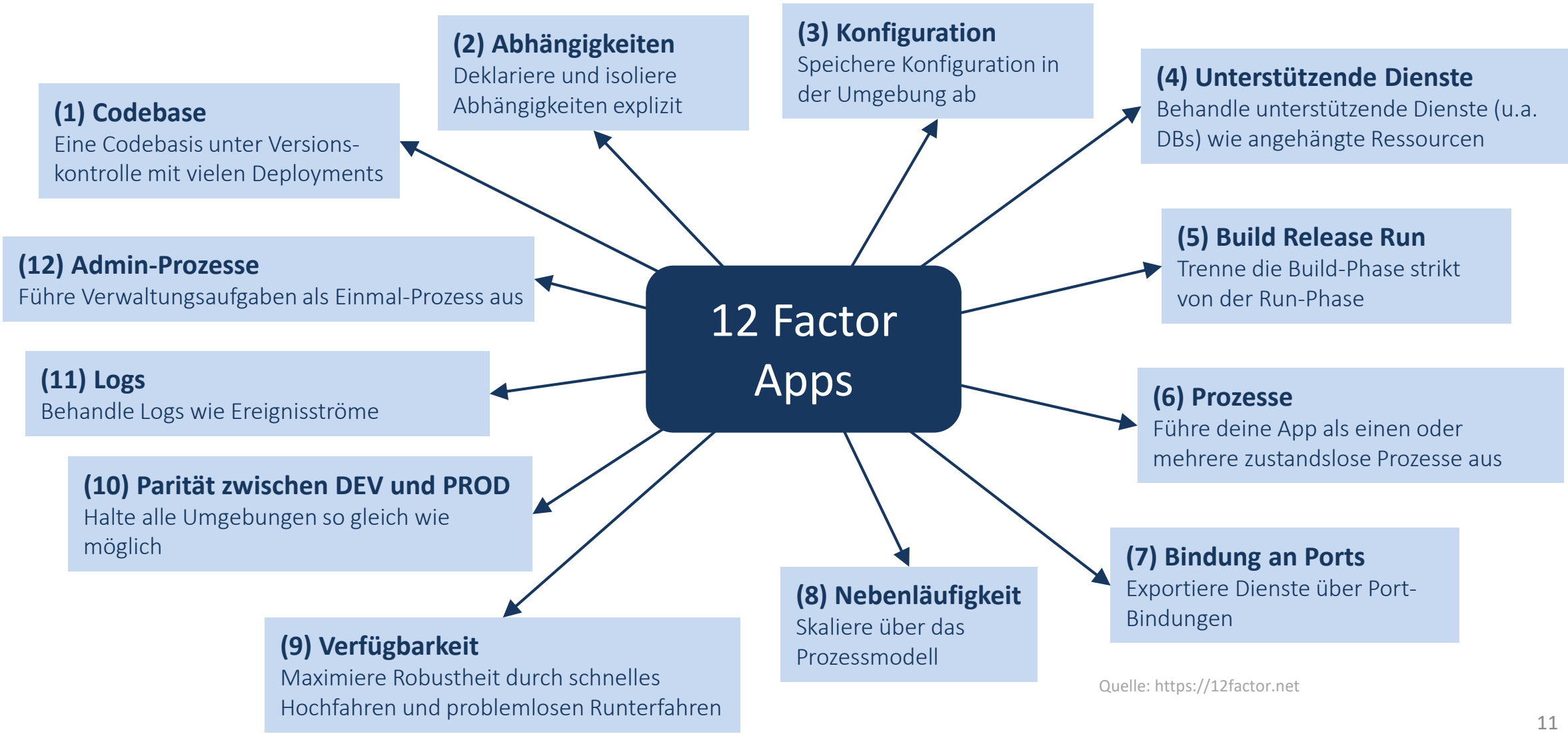
Bausteine einer Cloud Native Architecture



Bausteine einer Cloud Native Architecture

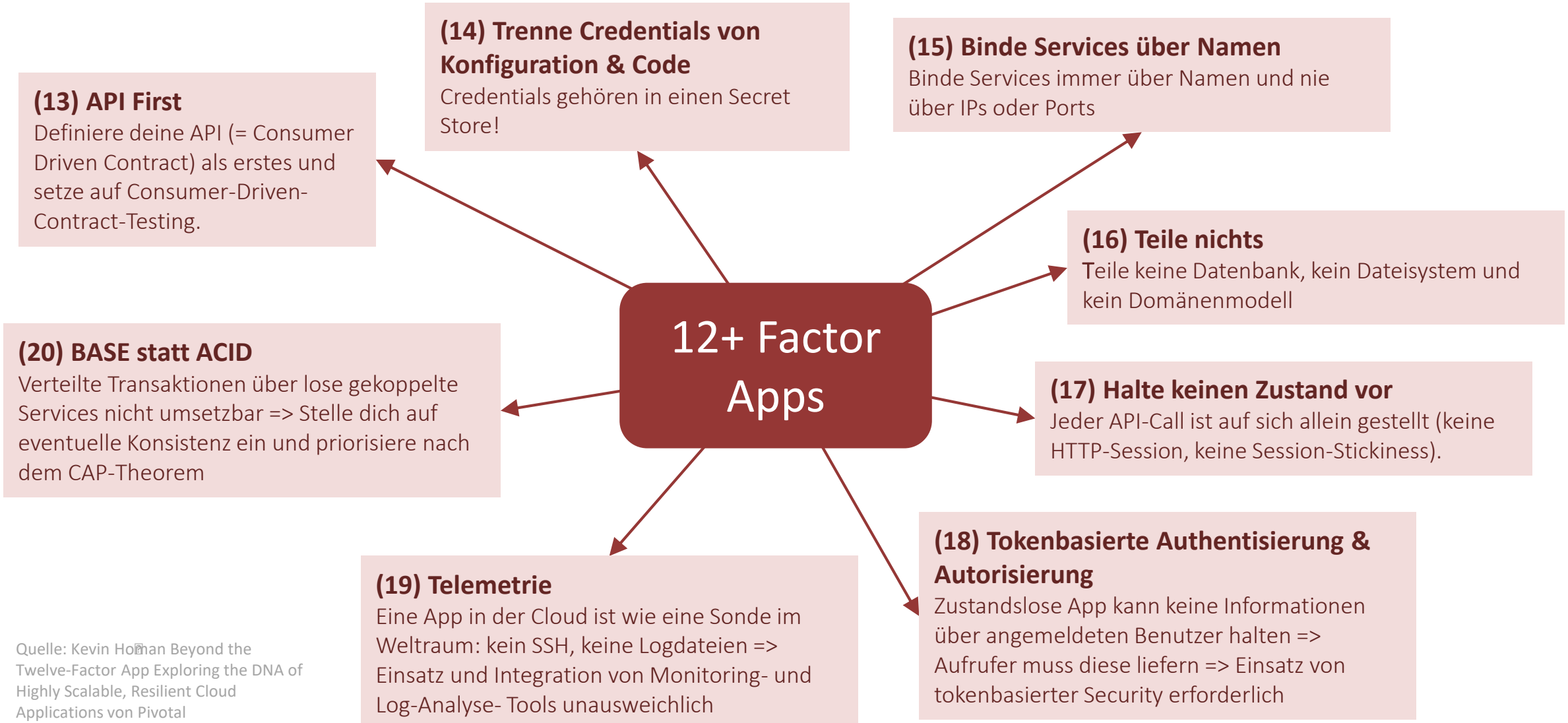


12 (Erfolgs-)Faktoren für Apps in der Cloud



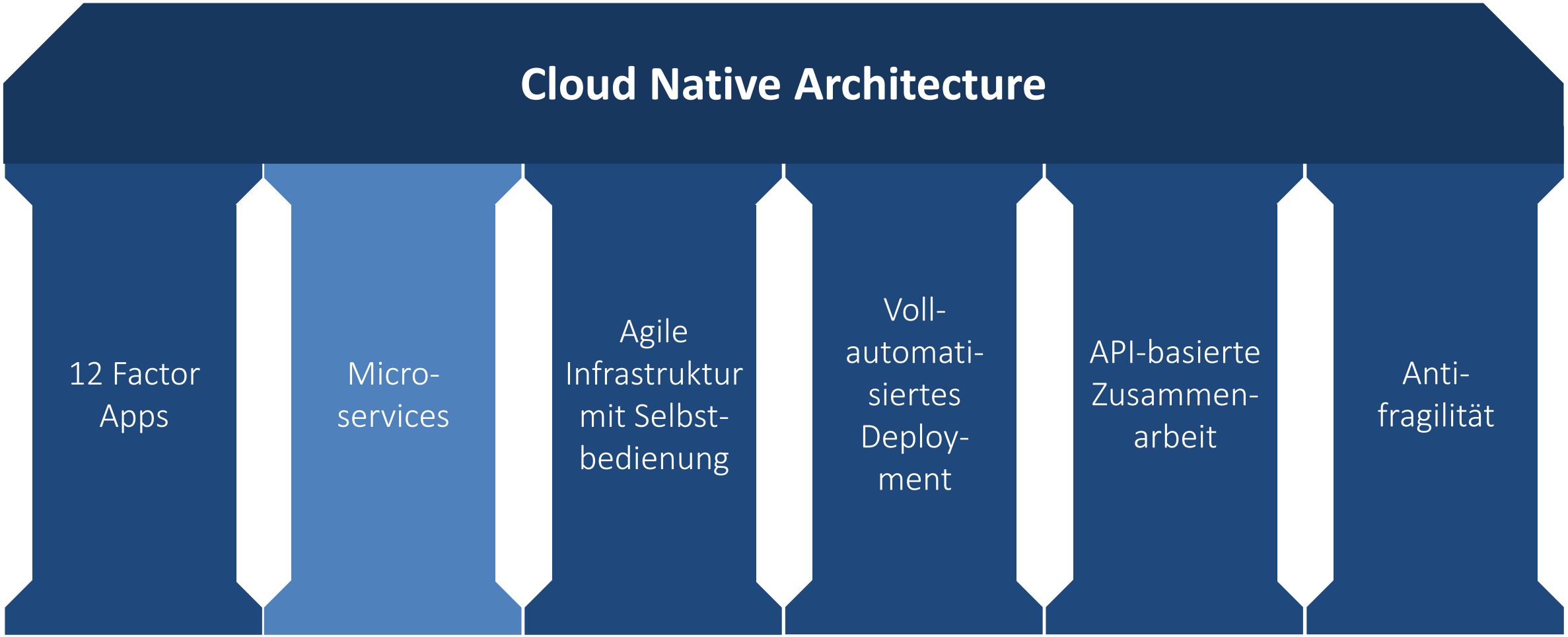
Quelle: https://12factor.net

… und was sonst noch wichtig ist



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

Bausteine einer Cloud Native Architecture



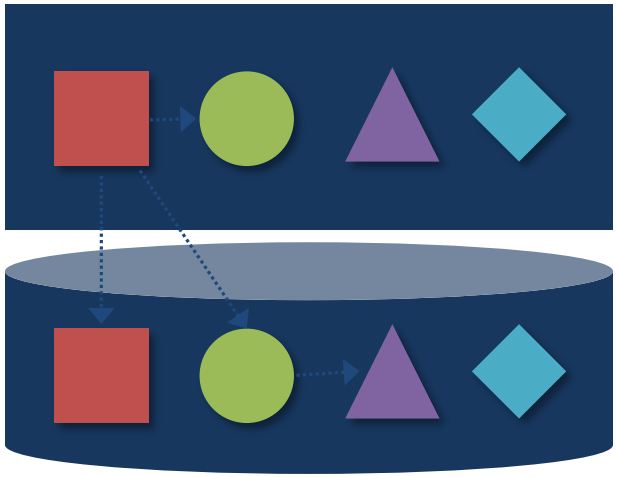
Microservices als Programmiermodell

Microservices sind ein Architekturmuster der Informationstechnik, bei dem komplexe Anwendungssoftware aus unabhängigen Prozessen komponiert wird, die untereinander mit sprachunabhängigen Programmierschnittstellen kommunizieren. Die Dienste sind weitgehend entkoppelt und erledigen eine kleine Aufgabe. So ermöglichen sie einen modularen Aufbau von Anwendungssoftware.

Quelle: Wikipedia; https://de.wikipedia.org/wiki/Microservices

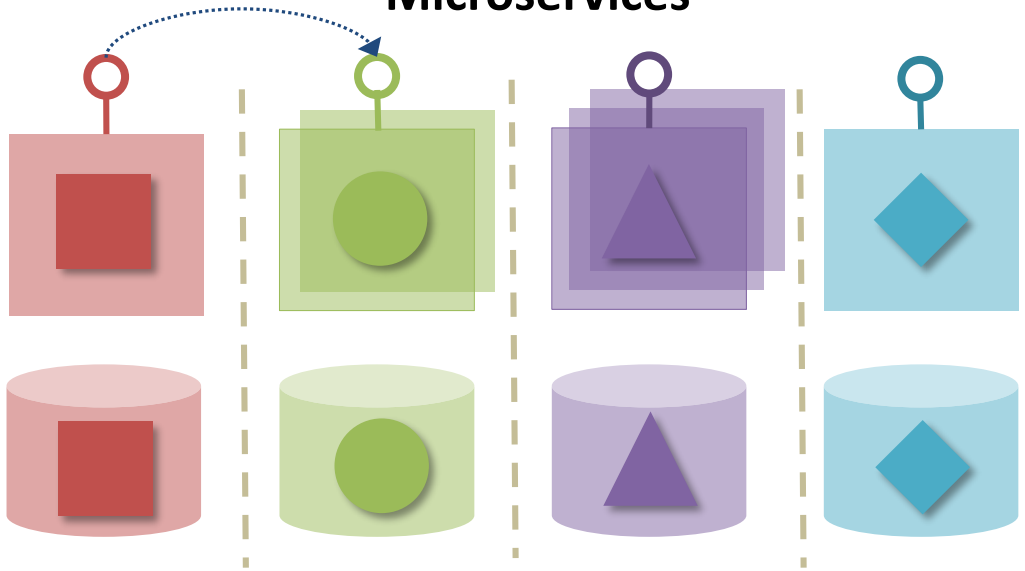
Teile und Herrsche!

Monolith



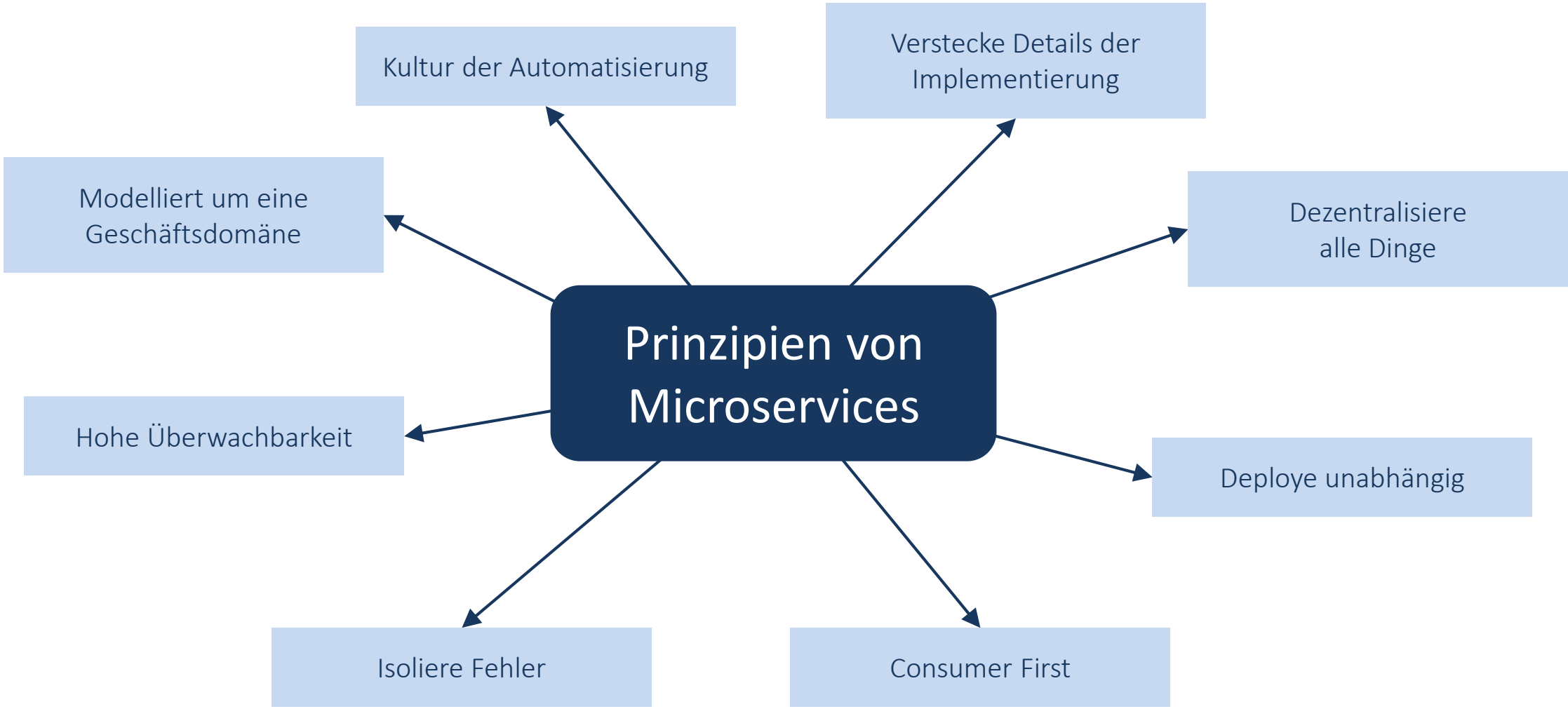
- Eine Plattform
- Eine Datenbank
- Ein Prozess, eine Deployment Unit, ein Release
- Hohe Einheitlichkeit
- Hohe Abhängigkeit
- Großes Team mit klaren Rollen

Microservices



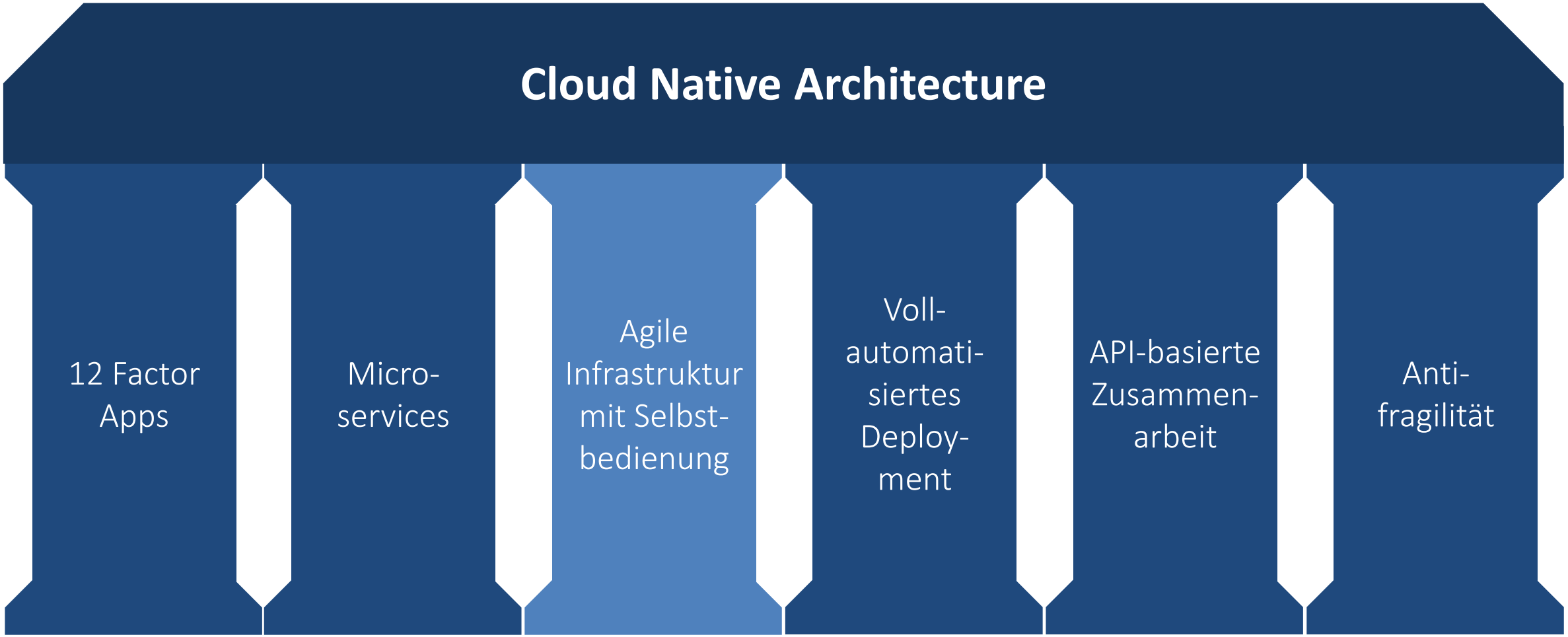
- Unterschiedliche Plattformen
- Polyglotte Persistenz
- Viele Prozesse, viele Deployment Units, viele Releases
- Hohe Flexibilität
- Keine Abhängigkeit
- Kleine cross-funktionale Teams

Microservices verlangen strikte Prinzipien



Quelle: Sam New, Building Microservices

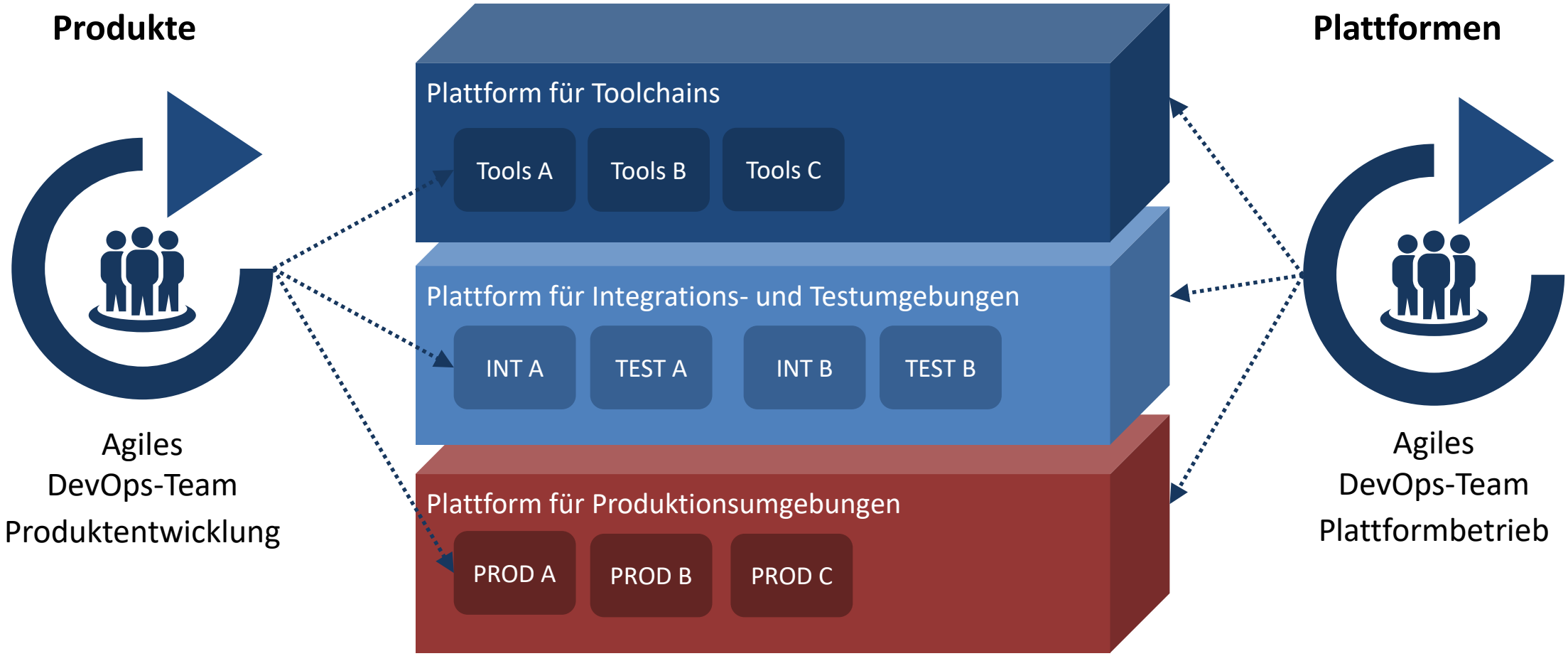
Bausteine einer Cloud Native Architecture



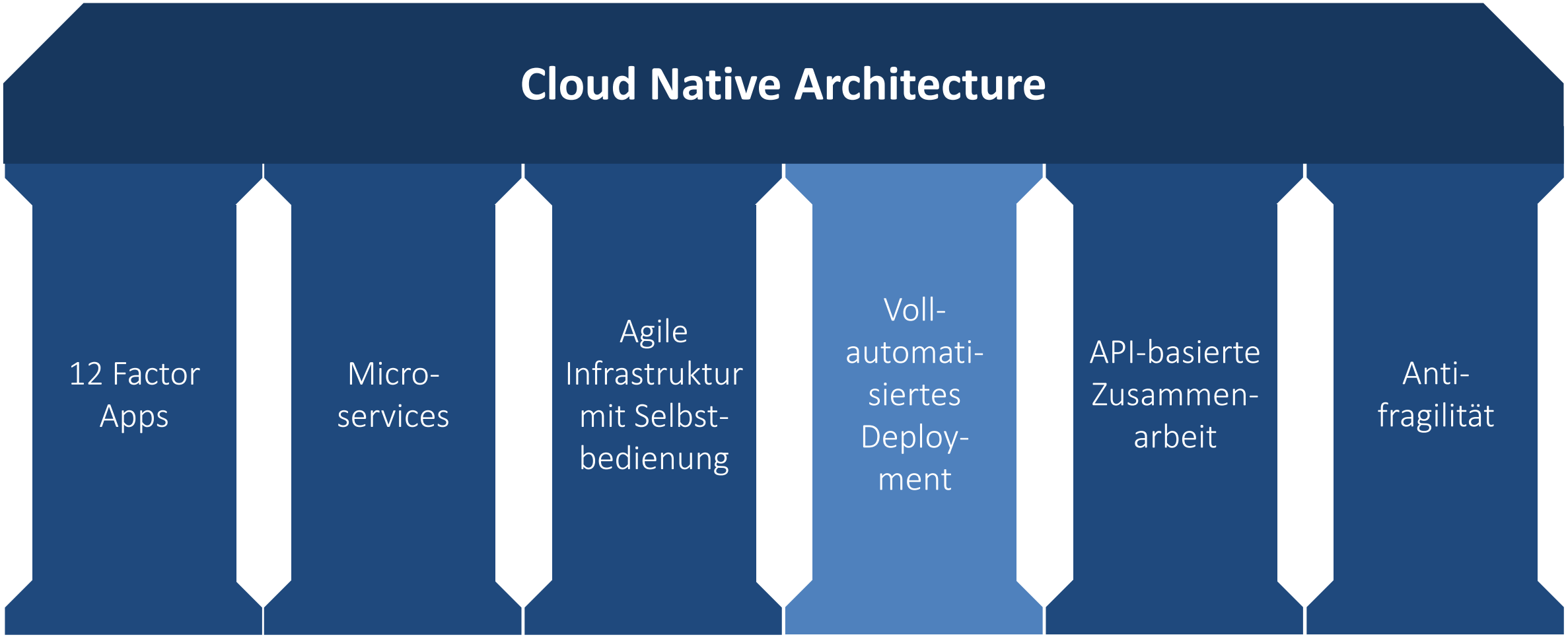
Merkmale agiler Infrastruktur

- Benötigte Infrastruktur und Services kann sich jeder selber provisionieren (Kiosk)
- Provisionierung von Infrastruktur dauert Minuten statt Monate
- Unterstützt Automatisierung (Infrastructure as Code)
- Abgerechnet werden nur die tatsächlich genutzten Ressourcen
- Status der provisionierten Ressourcen für jedermann einsehbar
- Verantwortung für provisionierte Ressourcen wandert ins Entwicklungsteam (DevOps)

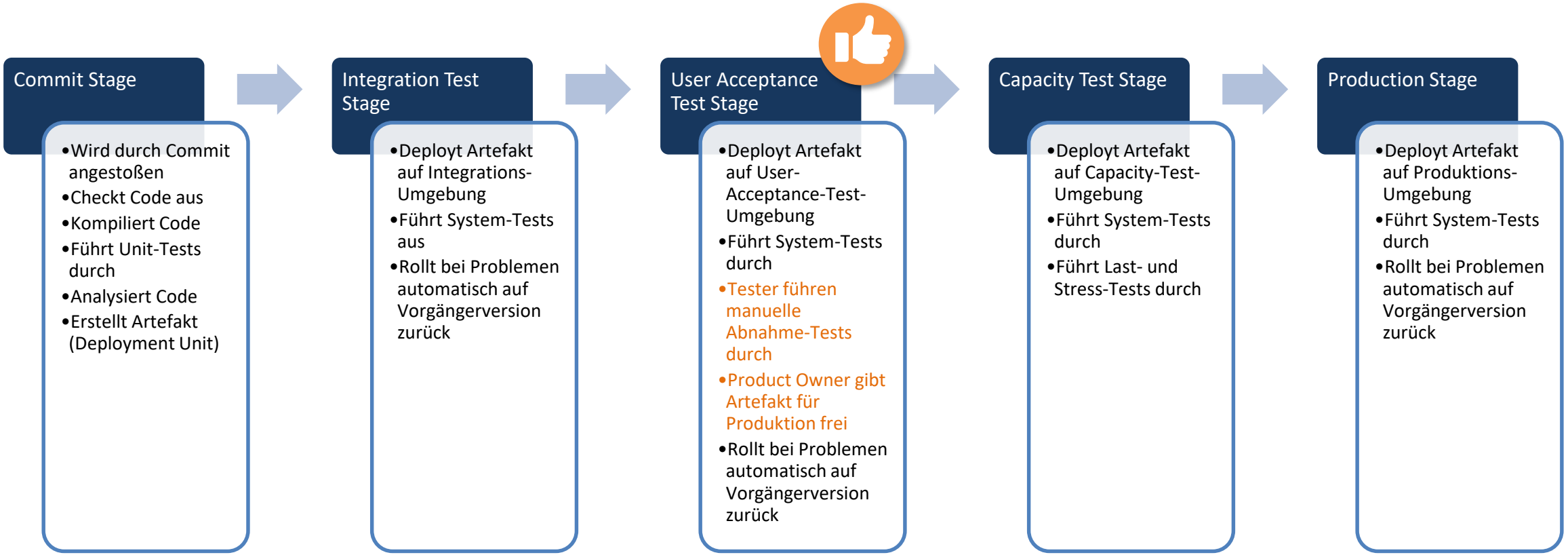
DevOps auf zwei Ebenen



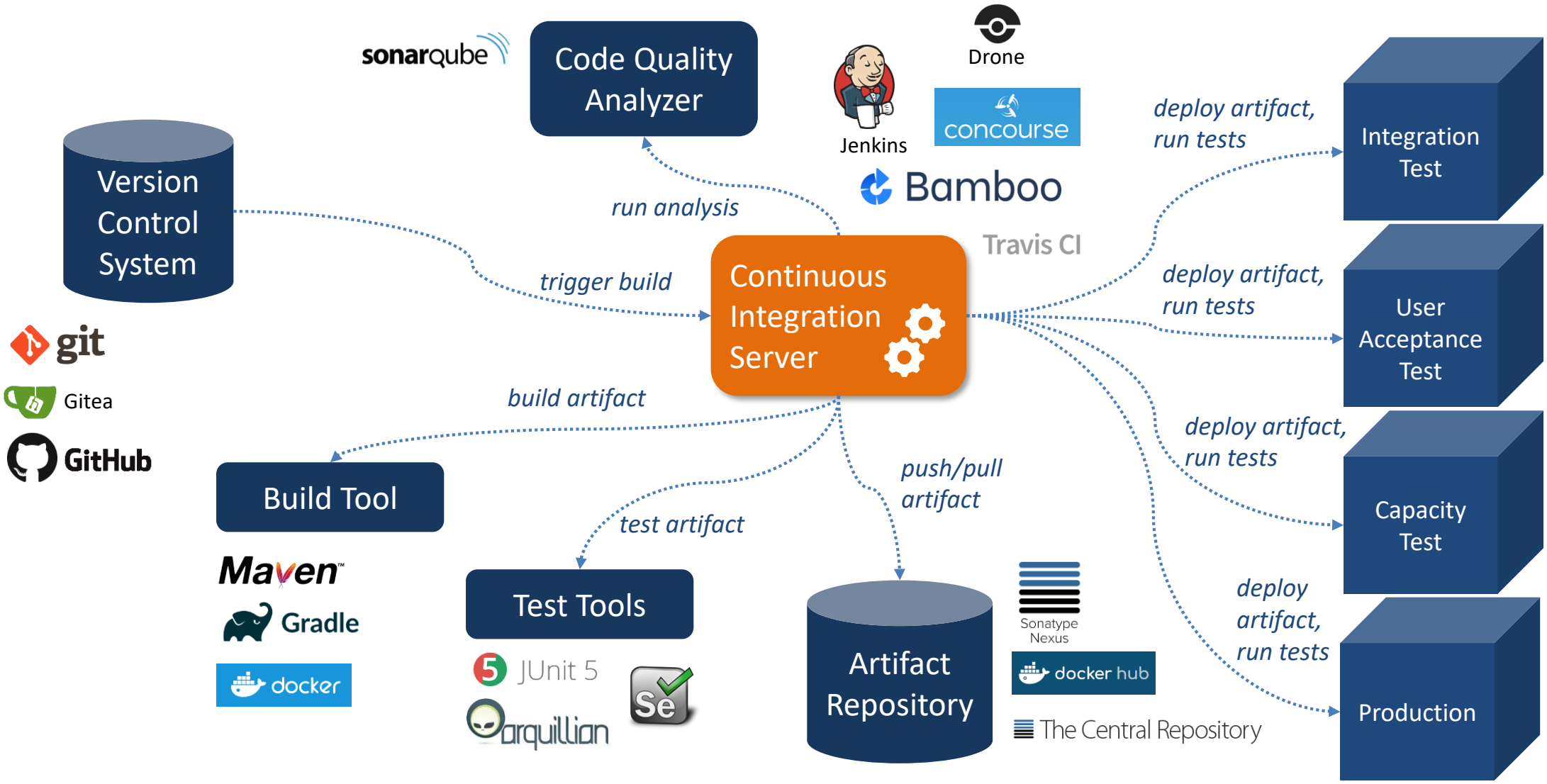
Bausteine einer Cloud Native Architecture



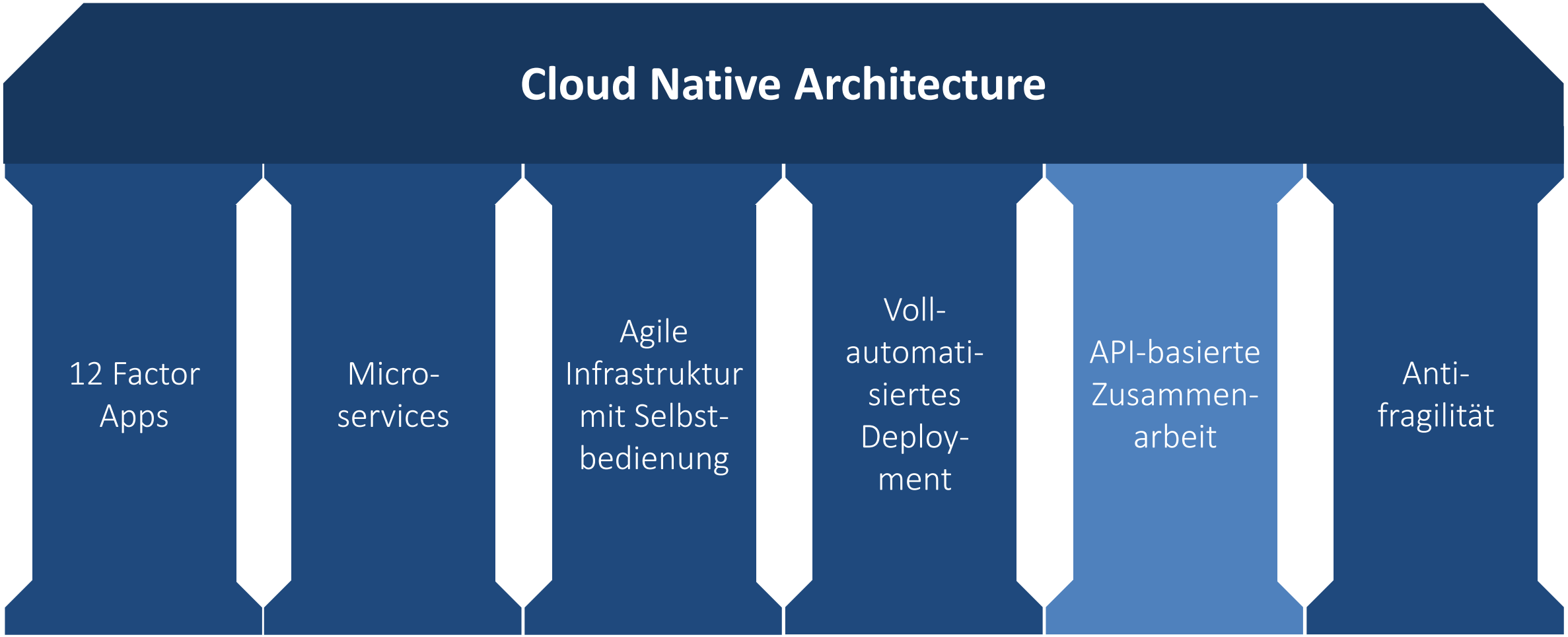
Vollautomatische Deployment Pipeline



Tools für Continuous Deployment

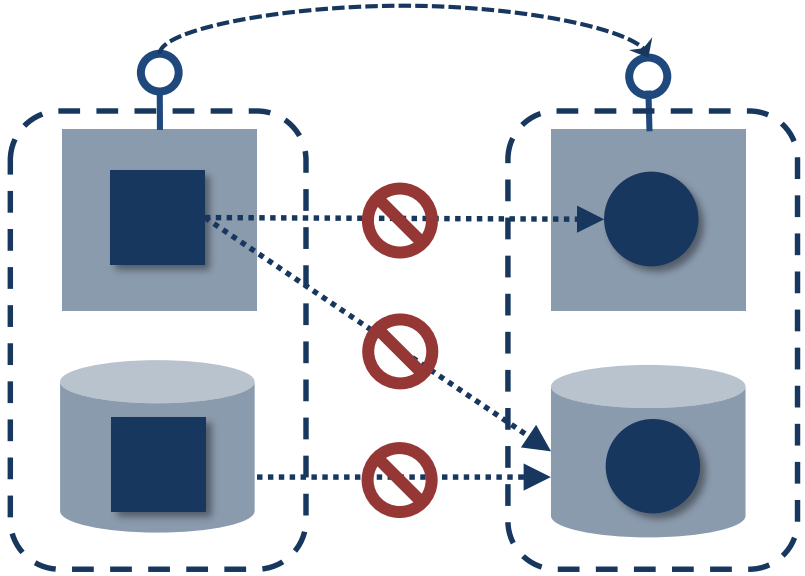


Bausteine einer Cloud Native Architecture

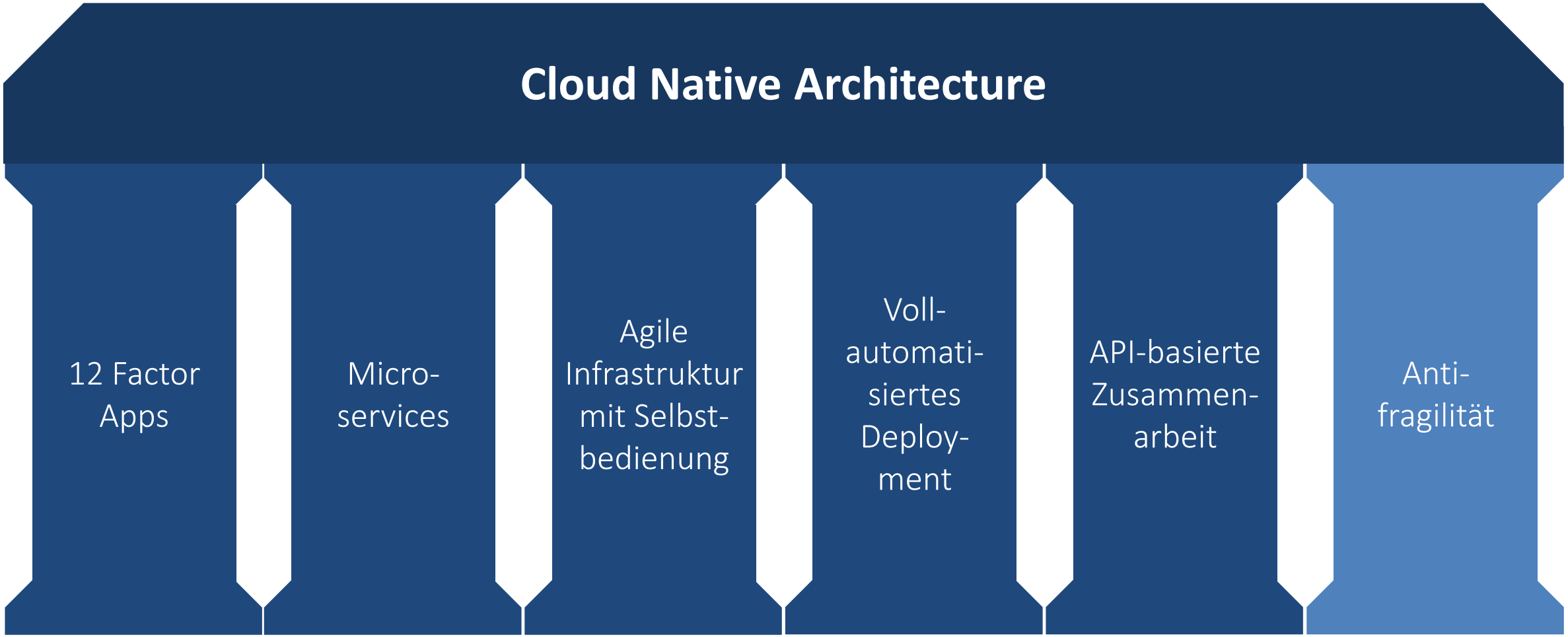


Kommunikation ausschließlich über APIs

- Motivation: Verbergen von Implementierungsdetails gegenüber dem Nutzer
- Außer der API wird nichts mehr geteilt
- Synchrone und asynchrone Kommunikation möglich
- Zahlreiche Herausforderungen
  - Kompatibilität, Granularität, Austausch großer Datenmengen, Orchestrierung oder Choreographie, HATEOAS
- Consumer Driven Testing erforderlich



Bausteine einer Cloud Native Architecture



Antifragilität - mehr als Resilienz/Robustheit

Antifragilität ist mehr als Resilienz oder Robustheit. Das Resiliente, das Widerstandsfähige widersteht Schocks und bleibt sich gleich; das Antifragile wird besser […]
Das Antifragile steht Zufälligkeit und Ungewissheit positiv gegenüber, und das beinhaltet auch – was entscheidend ist – die Vorliebe für eine bestimmte Art von Irrtümern. Antifragilität hat die einzigartige Eigenschaft, uns in die Lage zu versetzen, mit dem Unbekannten umzugehen, etwas anzupacken – und zwar erfolgreich –, ohne es zu verstehen.

Quelle: Nassim Nicholas Taleb, Antifragilität, S.21/22

Umgang mit Fehlern überlebenswichtig

- In massiv verteilten Systemen lautet die Frage nicht, *ob* Fehler auftreten werden sondern *wann* sie auftreten werden!
- Motivation: Antifragile Systeme werden durch Ausfälle besser
- Extrem erfolgreiches Beispiel für Antifragilität durch Chaos Engineering: Netflix mit seiner Simian Army
- Bestimmte Patterns helfen bei der Umsetzung von Antifragilität

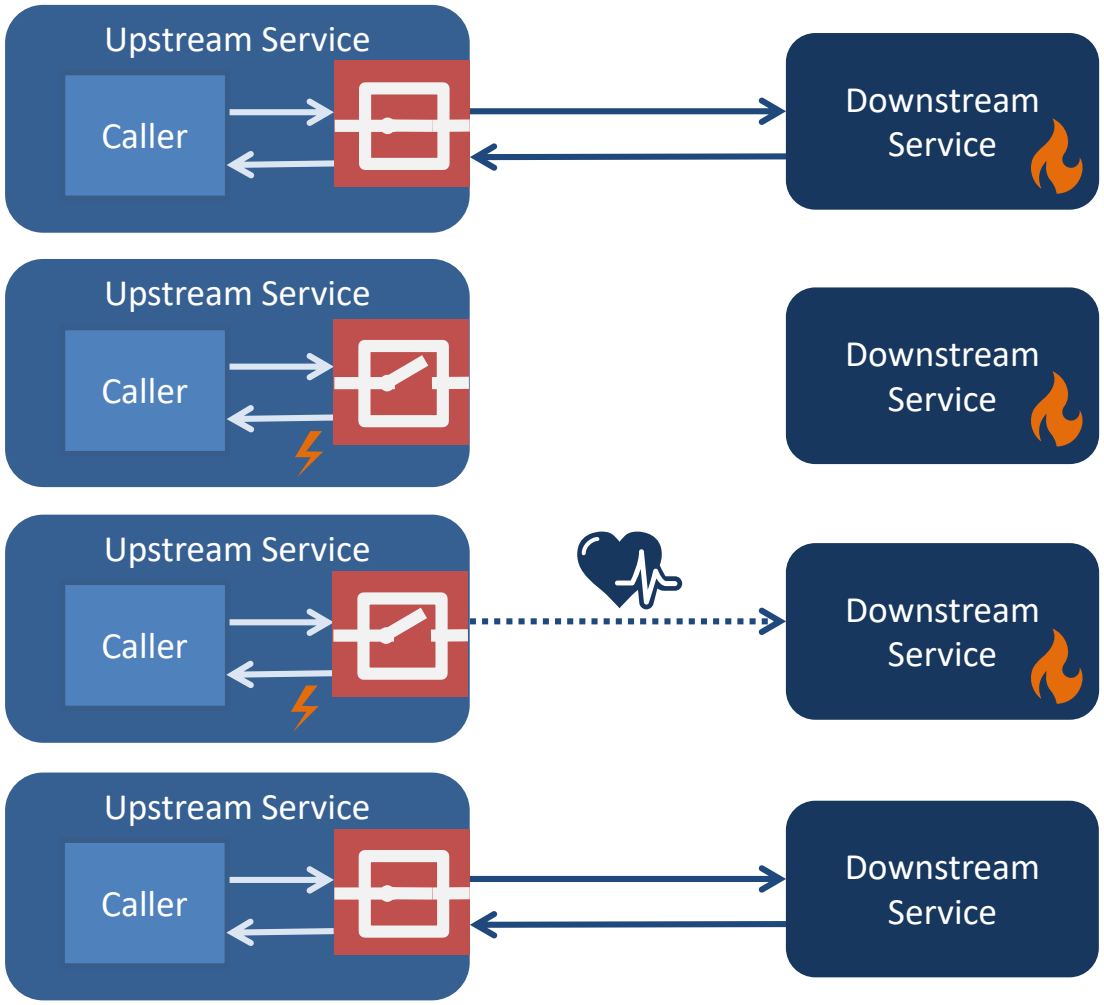
Timeouts: Fail Fast when Getting Slow!

> We discovered the hard way that systems that just act slow are much harder to deal with than systems that just fail fast.

- Sind leicht zu übersehen aber essentiell wichtig
- Aufrufe aus eigenem Prozess heraus mit Timeouts versehen
- Eintretende Timeouts müssen am besten aktiv überwacht jedoch mindestens passiv protokolliert werden
- Finden des richtigen Timeout-Wertes ist harte Arbeit!

Quelle: Sam Newman, Building Microservices, S. 210.

Circuit Breakers - Lieber die Lichter schnell ausgehen lassen bevor die Hütte brennt



(1) Aufrufe schlagen fehl oder laufen in Timeout

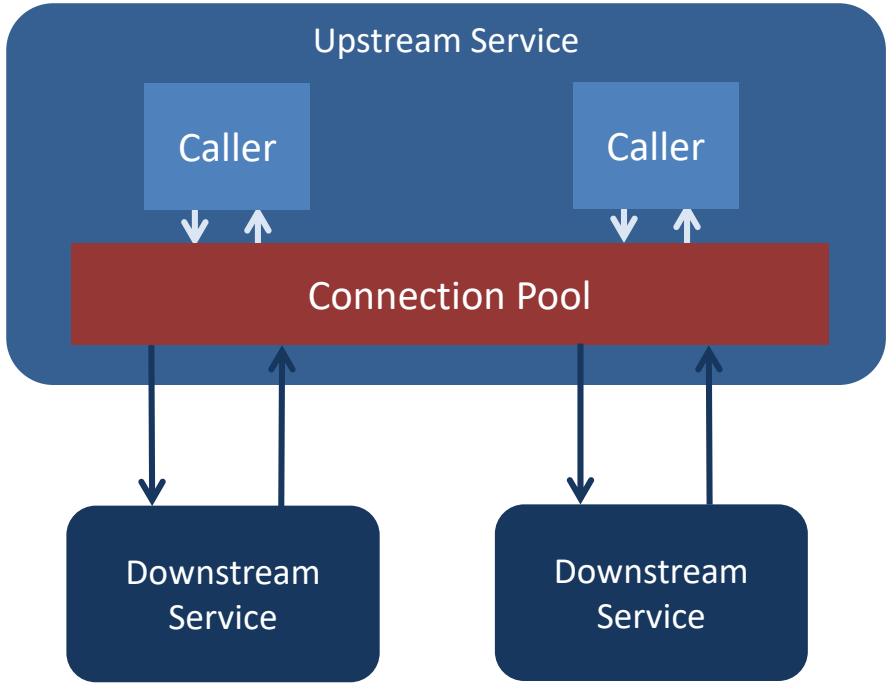
(2) Die Verbindung wird unterbrochen bei Erreichen eines Schwellwertes und Aufrufe schlagen schnell fehl

(3) Verfügbarkeit wird periodisch überprüft und Aufrufe schlagen schnell fehl

(4) Verbindung wird wieder hergestellt, wenn Schwellwert für positive Gesundheitschecks erreicht. Aufrufe werden wieder durchgereicht.

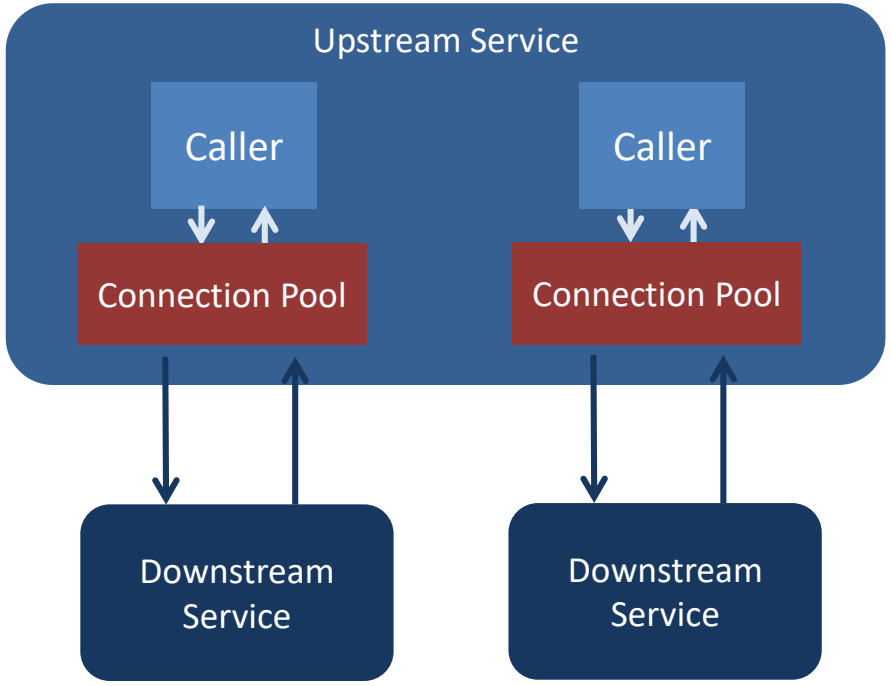
Quelle: Michael Nygard

Bulkheads oder die Kunst beim einem Fehler nicht gleich unterzugehen



Gemeinsamer Connection Pool / Thread Pool

- Ausfall eines Downstream Services führt zum Verhungern des Pools
- Alle Nutzer des gemeinsamen Pools betroffen



Getrennte Connection Pools / Thread Pools

- Ausfall eines Downstream Services führt nur zum Verhungern des zugeordneten Pools
- Nur Nutzer des dedizierten Pools betroffen

Fragen?

ANHANG

Quellen

- Josh Long, Kenny Bastani:
  Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud and Cloud Foundry
  O´Reilly UK Ltd, 1. Auflage, 22. August 2017; ISBN 978-1449374648
- Sam Newman: ***Building Microservices: Designing Fine Grained Systems***
  O´Reilly & Associates, 1. Auflage, 25. September 2015; ISBN 978-1491950357
- Michael T. Nygard: ***Release It! Design and Deploy Production-Ready Software***
  The Pragmatic Bookshelf, 1. Auflage, 12. Juli 2017, ISBN 978-0978739218
- Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones, Ali Basiri
  Chaos Engineering: Building Confidence in System Behavior Through Experiments
  O´Reilly Media, Inc; 1. Auflage August 2017; eBook
- Eric Evans: ***Domain Driven Design: Tackling Complexity in the Heart of Software***
  Addison Wesley 2004; ISBN 0-321-12521-5
- Nassim Nicholas Taleb: ***Antifragile: Things that Gain from Disorder***
  Penguin, 6. Juni 2013; ISBN 978-0141038223

Kontakt

Michael Theis
Lehrbeauftragter Hochschule München

email michael.theis@hm.edu
mobile + 49 170 5403805
web http://www.tschutschu.de/Lehrauftrag.html