

Modul CNT

Architekturstyles für Container (Microservices)

September 2021

Marcel Bernet

Dieses Werk ist lizenziert unter einer
[Creative Commons Namensnennung - Nicht-kommerziell -
Weitergabe unter gleichen Bedingungen 3.0 Schweiz Lizenz](https://creativecommons.org/licenses/by-nc-sa/3.0/de/) /




Lernziele

- ★ Sie haben einen ersten Überblick über die neue Welt der Microservices und Container.
- ★ Sie haben Microservices in einer Docker und Kubernetes Umgebung zum laufen gebracht.

Zeitlicher Ablauf

- ★ Container Life-Cycle
 - Die Sicht des Entwicklers (Dev)
 - Die Sicht des Operatings (Ops)
- ★ Architekturmuster Microservices
 - Konzepte für Microservices
 - Applikation vs Microservices vs Container
 - Unser Unternehmen und Microservices
 - Wie bestehende Applikationen einbinden?
 - Perspektive der CEOs/Entscheider
- ★ Übung
- ★ Reflexion
- ★ Lernzielkontrolle
- ★ Bonus: Microservices und Teambildung, Domain Driven Design

Die Sicht der Entwickler (Dev)



ASP.NET Core-Webanwendung

Projektvorlagen zum Erstellen von ASP.NET Core-Anwendungen für Windows, Linux und macOS mithilfe von .NET Core oder .NET Framework. Erstellen Sie Web-Apps mit Razor Pages, MVC und Blazor oder Single-Page-Anwendungen (SPA) mit Angular, React oder React + Redux. Erstellen Sie auch Web-APIs, gRPC-Dienste und Workerdienste.

C#


Windows

Linux

macOS

Web

1



Webanwendung (Model-View-Controller)


Eine Projektvorlage zum Erstellen einer ASP.NET Core-Anwendung mit Beispielen für ASP.NET Core-MVC-Ansichten und -Controller. Diese Vorlage kann auch für RESTful HTTP-Dienste verwendet werden.

2

Erweitert


☒ Für HTTPS konfigurieren

☒ Docker-Unterstützung aktivieren




Razor-Klassenbibliothek

Eine Projektvorlage zum Erstellen einer Razor-Klassenbibliothek.



Angular

Eine Projektvorlage zum Erstellen einer ASP.NET Core-Anwendung mit Angular.



React.js

[Zusätzliche Projektvorlagen abrufen](#)

Docker Desktop installieren

Zum Debuggen in einem Container muss Docker Desktop installiert sein, und die Windows-Features für Container und Hyper-V müssen aktiviert werden. Dieser Vorgang erfordert erhöhte Zugriffsrechte und nimmt einige Minuten in Anspruch. Führen Sie nach Abschluss der Installation einen Neustart durch.

3

Möchten Sie Docker Desktop installieren und diese Features aktivieren?

☐ Diese Meldung nicht mehr anzeigen

Ja






Nein

4

Webanwendung (Ausgabe)

localhost:5001

Apps Gmail Maps



Application uses

- Sample pages using ASP.NET Core MVC
- Theming using [Bootstrap](#)

About Docker Desktop

Discover Docker Enterprise Edition

Settings

Check for Updates

Diagnose and Fix

Switch to Windows

Docker Hub

Documentation

Kitematic

Sign in / Create

Repositories

Kubernetes

Restart...

Quit Docker Desktop

Veröffentlichungsziel auswählen

App Service

App Service Linux

Containerregistrierung

Azure Virtual Machines

IIS, FTP usw.

Ordner

Containerregistrierung

- Registrierung zum Speichern
- ☐ Neue Azure Container Registry
- ☐ Vorhandene Azure Container Registry
- ☒ Docker-Hub
- ☐ Benutzerdefiniert

5

Desktop

DEU

Was ist passiert auf Entwicklerseite (Dev)?

- 2

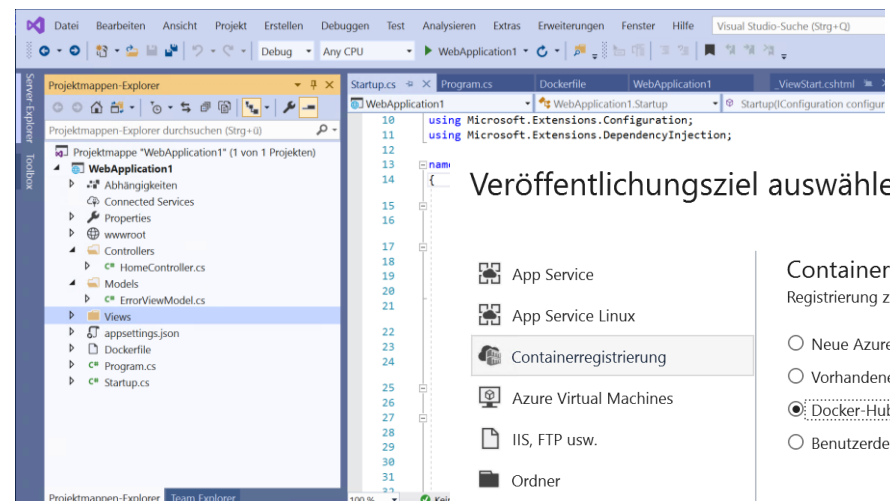
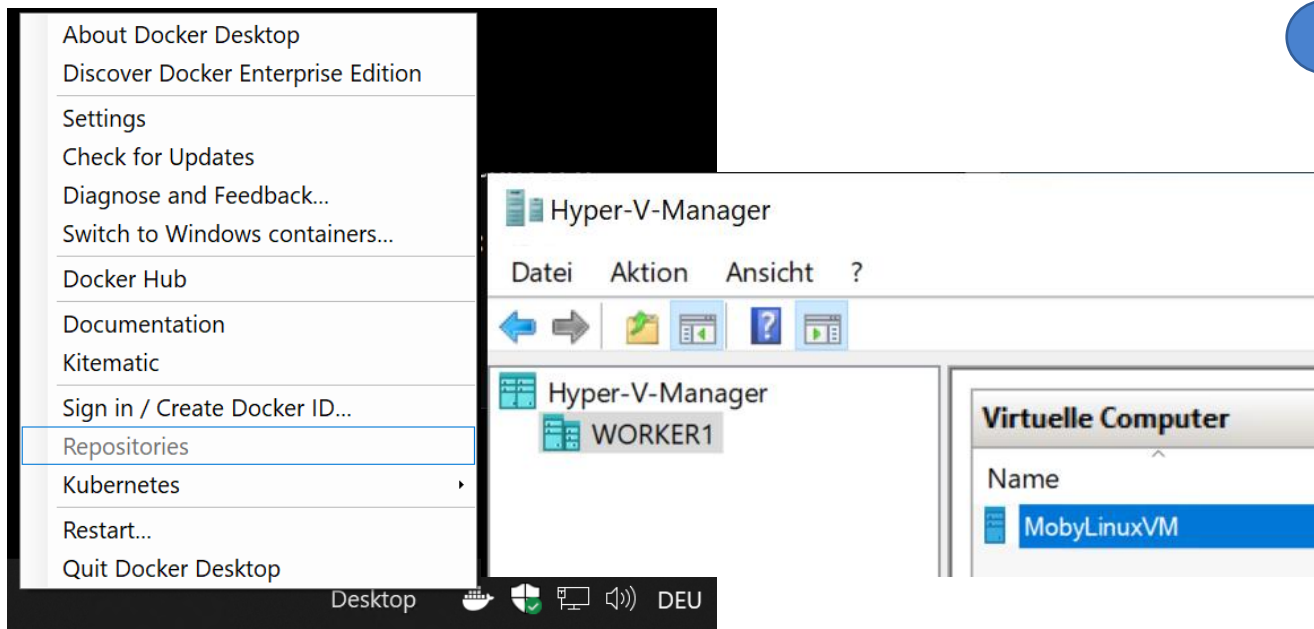
3

- ★ Docker Desktop (Applikation und Service) wurde installiert (= Container Umgebung)
 - ★ Hyper-V (Virtualisierung) wurde aktiviert
 - ★ Eine Linux VM wurde erstellt
- 1

- ★ Eine C# Webanwendung wurde erstellt
 - ★ Kompiliert in einem Container und mit allen Abhängigkeiten, **als Container Image verpackt**
- 4

- ★ Das Container Image wurde in einem **Container gestartet**
- 5

- ★ Das Container Image wurde in einer Registry (z.B. hub.docker.com) veröffentlicht.



Die Sicht des Operatings (Ops)

6

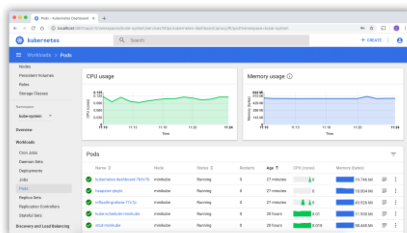
★ Starten und Orchestrieren (Kubernetes) von Containern mittels

★ **Docker**

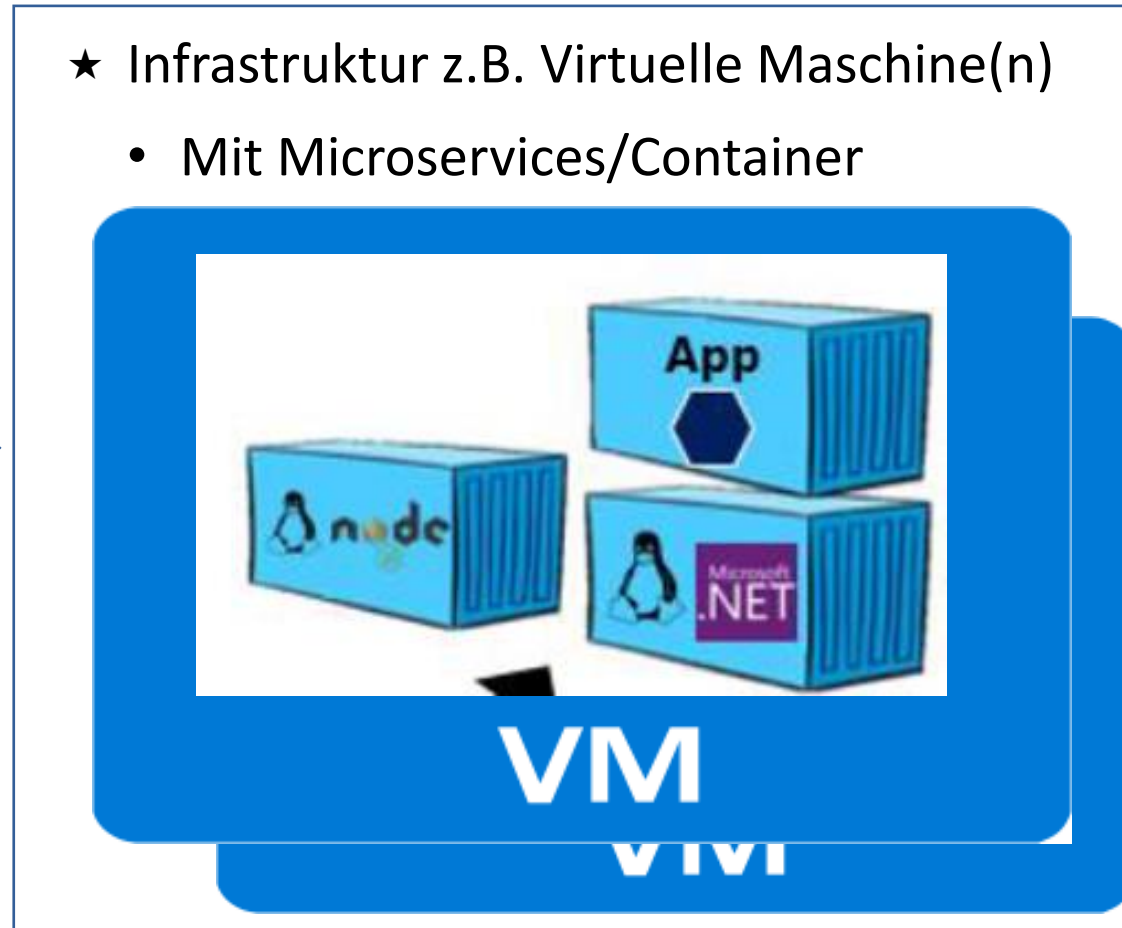
- docker run
- docker-compose

★ **Kubernetes**

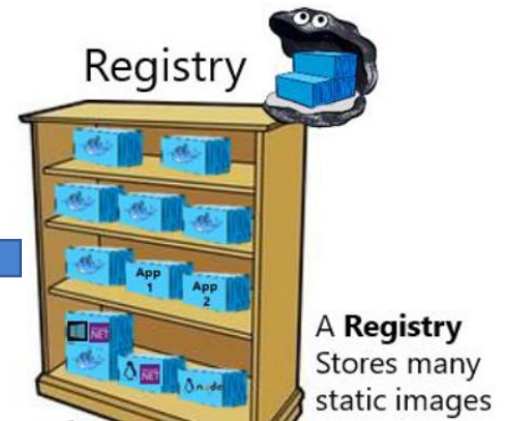
- (kubectl run ...)
- kubectl apply -f YAML
- Dashboard



- ★ Infrastruktur z.B. Virtuelle Maschine(n)
 - Mit Microservices/Container



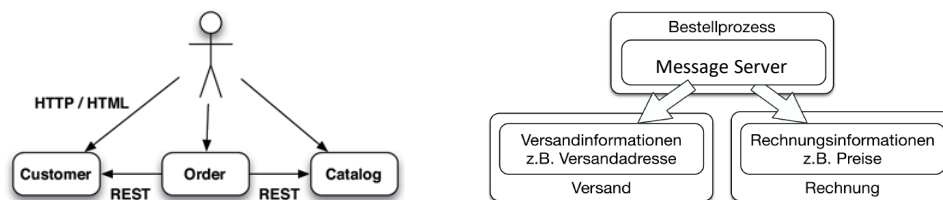
Es entfällt das Installieren von Applikationen!



z.B. <https://hub.docker.com>

Was hat der Entwickler gemacht?

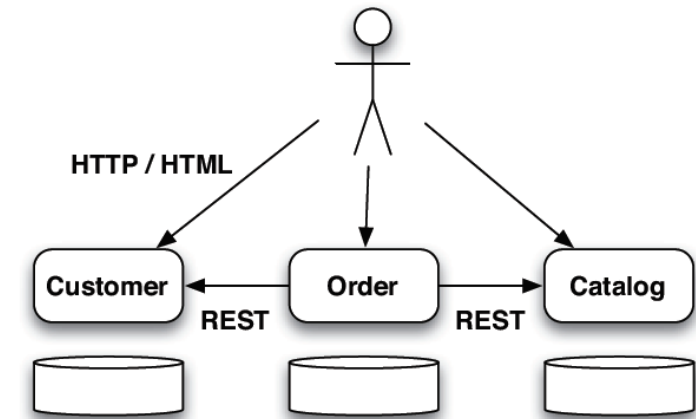
- ★ Der Entwickler hat das Architekturmuster **Microservices** verwendet, um eine Webanwendung zu erstellen.
- ★ Was sind Microservices?
 - Microservices – ein Ansatz zur Modularisierung von Software.
 - **Das Neue:** Microservices nutzen als Module einzelne Programme, die als eigene Prozesse laufen. Der Ansatz basiert auf der UNIX-Philosophie. Sie lässt sich auf drei Aspekte reduzieren:
 - ★ Ein Programm soll nur eine Aufgabe erledigen, und das soll es gut machen.
 - ★ Programme sollen zusammenarbeiten können.
 - ★ Nutze eine universelle Schnittstelle.
 - In UNIX sind das Textströme: `cat adressen.txt | sort | uniq`
 - Bei Microservices das Internet (TCP/IP, REST oder Messages)



Konzepte für Microservices (nicht abschliessend)

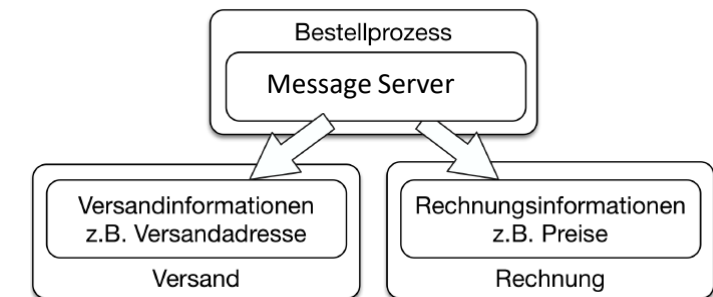
★ Sychrone Microservices (REST)

- Ein Microservice ist synchron, wenn er bei der Bearbeitung von Requests selber einen Request an andere Microservices stellt und auf das Ergebnis wartet.



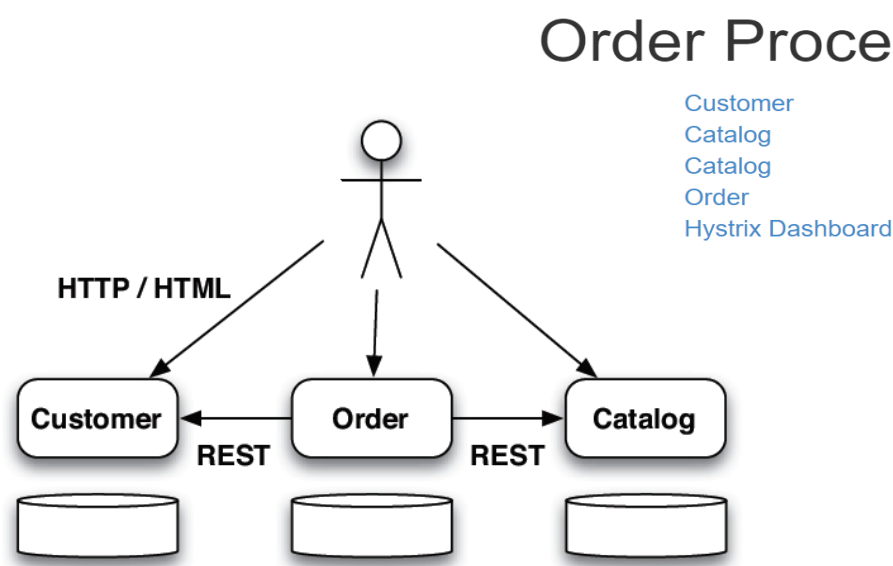
★ Asynchrone Microservices (Messages)

- Der Microservice schickt einem anderen Microservice (Event Bus, Message Server) einen Request, wartet aber nicht auf eine Antwort.



Synchrone Microservices: Applikation

- ★ Die Applikation besteht aus drei Microservices: *Order*, *Customer* und *Catalog*.
- ★ Order kommuniziert mittels Catalog und Customer via REST. Ausserdem bietet jeder Microservice einige HTML-Seiten an.
- ★ Zusätzlich ist ein Apache-Webserver installiert, der dem Benutzer mit einer Webseite einen einfachen Einstieg in das System ermöglicht.



Customer : View all

id	Name	Firstname	
1	Wolff	Eberhard	delete
3			
17			

Item : View all

id	Name	Price	
2	iPod	42.0	delete
4			

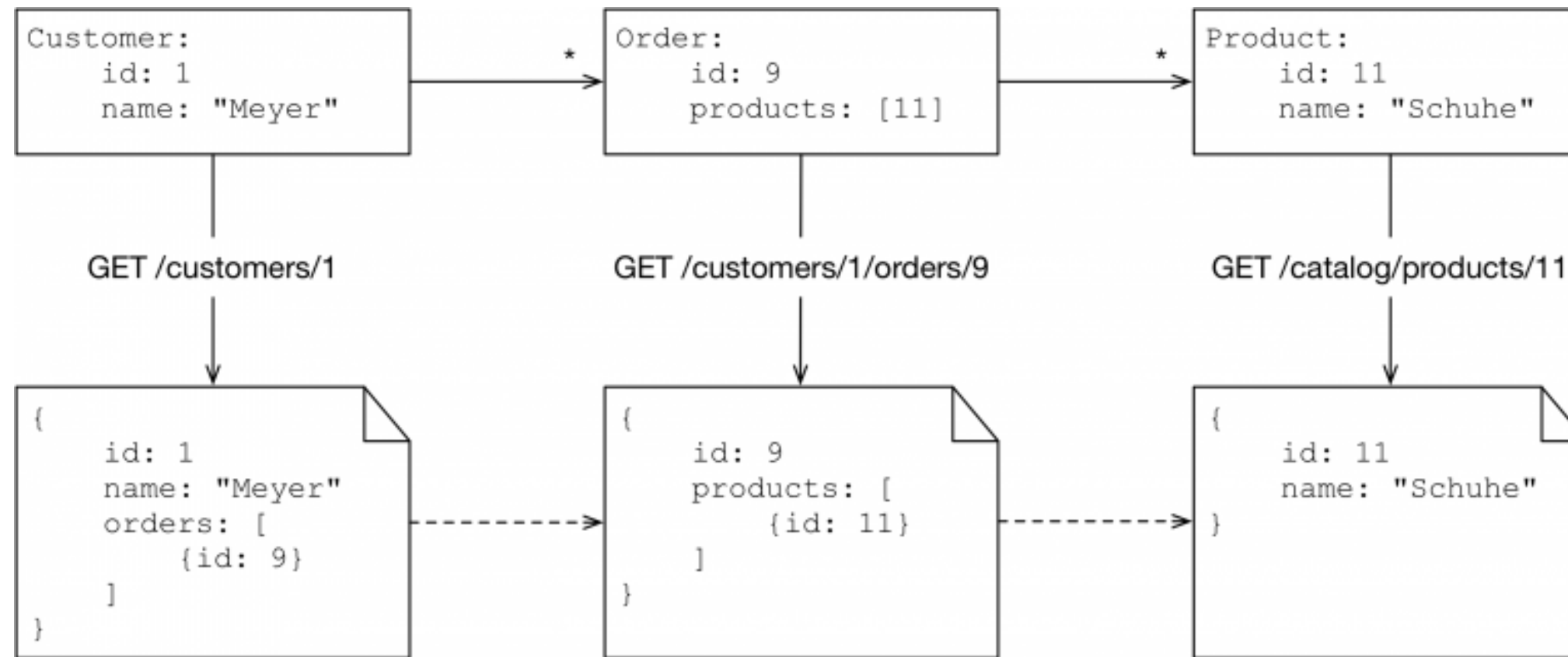
Order : View all

ID	Customer	Total Price	
18	VName Name	1.0	delete
20	VName Name	1.0	delete
22	VName Name	1.0	delete

Synchrone Microservices: REST (1)

- ★ Representational State Transfer (abgekürzt REST, seltener auch ReST) bezeichnet ein Programmierparadigma für verteilte Systeme.
- ★ REST ist eine Abstraktion der Struktur und des Verhaltens des World Wide Web (HTTP GET, PUT, POST, DELETE).
- ★ REST hat das Ziel, einen Architekturstil zu schaffen, der die Anforderungen des modernen Web besser darstellt. Dabei unterscheidet sich REST vor allem in der Forderung nach einer einheitlichen **Kommunikation Schnittstelle** von anderen Architekturstilen

Synchrone Microservices: REST (2)



- ★ Das Anfragen, Anlegen, Verändern und Löschen von Ressourcen erfolgt über die HTTP-Methoden GET, POST, PUT und DELETE. So liefert `GET /customers/1/orders/9` die Bestellung mit der ID 9, während `DELETE /customers/1/orders/9` diese Bestellung löscht.

Synchrone Microservices: Definition

- ★ Ein Microservice ist synchron, wenn er bei der Bearbeitung von Requests selbst einen Request an andere Microservices stellt und auf das Ergebnis wartet.

Synchrone Microservices: Vor- / Nachteile

- ★ Synchrone Microservices sind *einfach zu verstehen*. Statt eines lokalen Methoden-Aufrufs wird eine Funktionalität in einem anderen Microservice aufgerufen. Das entspricht dem, was Programmierer gewohnt sind.
- ★ Es kann eine bessere *Konsistenz* erreicht werden. Wenn bei jedem Aufruf die neuesten Informationen aus den anderen Services geholt werden, dann sind die Daten aktuell und entsprechen den Informationen der anderen Microservices, wenn nicht in letzter Sekunde noch eine Änderung eingetreten ist.
- ★ **Dafür ist *Resilience* aufwändiger:** Wenn der aufgerufene Microservice gerade nicht zur Verfügung steht, muss der Aufrufer mit dem Ausfall so umgehen, dass er nicht ebenfalls ausfällt. Dazu kann der Aufrufer Daten aus einem Cache nutzen oder, wie bei Kubernetes, mehrere Instanzen des abhängigen Microservices starten.

Übung: Synchrone Microservices (->demo ->...REST)

Quelle: Buch Microservices Rezepte

Das Beispiel besteht aus drei Microservices: **Order**, **Customer** und **Catalog**.

Order nutzt **Catalog** und **Customer** mit der REST-Schnittstelle. Ausserdem bietet jeder Microservice einige HTML-Seiten an.

Zusätzlich ist im Beispiel ein Apache-Webserver installiert, der dem Benutzer mit einer Webseite einen einfachen Einstieg in das System ermöglicht.

Ebenso steht ein [Hystrix Dashboard](#) als eigener Kubernetes-Pod zur Verfügung. Das Beispiel nutzt die Java-Library Hystrix22, um Resilience zu erreichen. Diese Bibliothek führt Aufrufe in einem anderen Thread Pool aus und implementiert unter anderem einen Timeout für die Aufrufe.

```
In [ ]: ! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/apache.yaml
! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/catalog.yaml
! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/customer.yaml
! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/hystrix.yaml
! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/order.yaml
! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/ewolff/ms-kubernetes/postgres.yaml
! kubectl config view -o=jsonpath='{.clusters[0].cluster.server}' | sed -e 's/https:/http/' -e 's/6443/${kubectl get service a
```

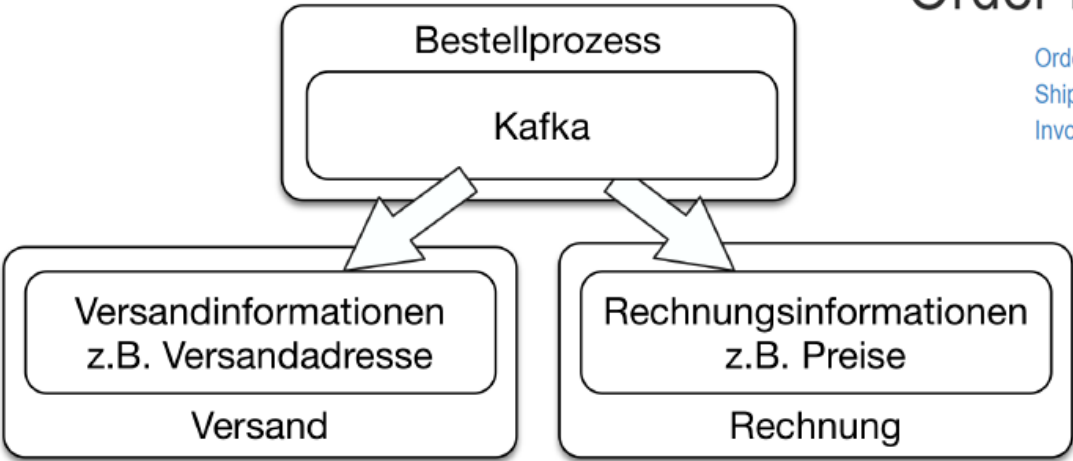
★ Fragen:

- Wie viele Prozesse laufen in den einzelnen Containern? (In der Weave Scope Umgebung auf Container Icon klicken).
- Warum haben die Microservices ein eigenes Schema in der Datenbank. Warum wird dieses nicht geteilt mit den anderen Microservices?
- Wie findet die Kommunikation statt?

https://gitlab.com/ch-tbz-hf/Stud/cnt/-/tree/main/2_Unterrichtsressourcen/J#microservice-umgebung

Asynchrone Microservices: Beschreibung

- ★ Die Applikation besteht aus einem Microservice **order**, der eine Bestellung über die Weboberfläche entgegennimmt.
- ★ Die Bestellung schickt **order** dann als Message über Kafka (Message Server) an den Microservice für den Versand **shipping** und den Microservice für die Erstellung der Rechnung **invoicing**.
- ★ Die Bestellung wird als JSON (-Message) übertragen. So können der Rechnungs-Microservice und der Versand-Microservice aus der JSON Datenstruktur jeweils die Daten auslesen, die für den jeweiligen Microservice relevant sind.



Order Processing

Order : View all

ID	Customer	Total Price	
7	Eberhard Wolff	84.0	delete

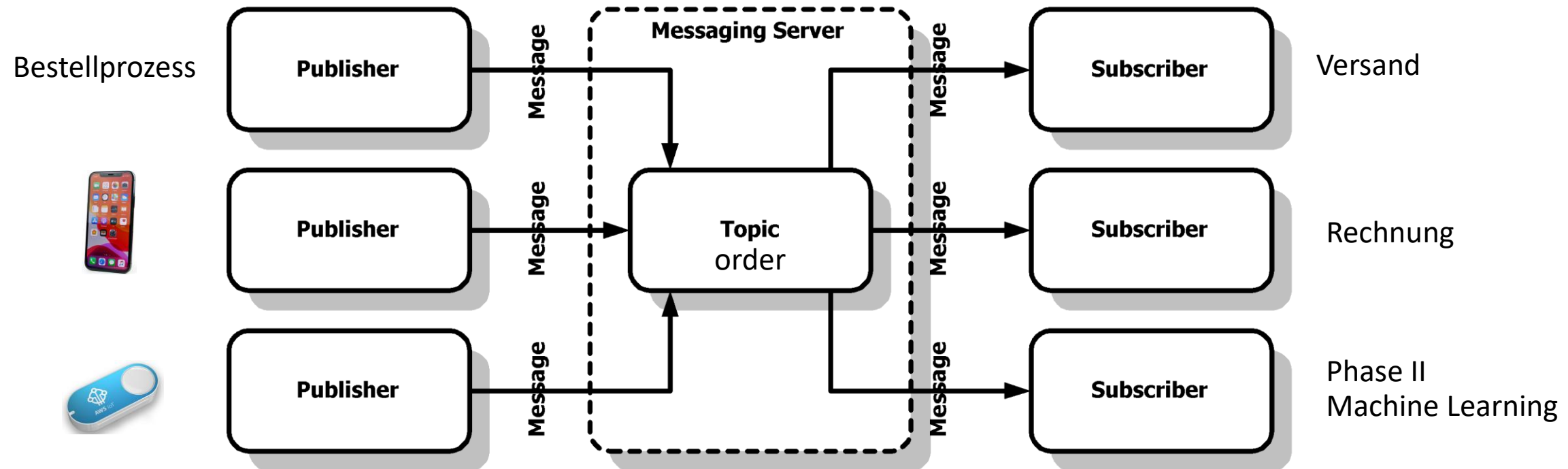
Shipments : View all

ID	Customer
7	
9	

Invoicing : View all

ID	Customer	Total Amount
7	Eberhard Wolff	84.0
9	Eberhard Wolff	84.0

Asynchrone Microservices: Publish/Subscribe Muster



- ★ Bei Asynchronen Microservices, kann das Publish/Subscribe Muster verwendet werden.
- ★ Es ersetzt die Punkt-zu-Punkt-Verbindungen durch einen zentralen Server (Messaging Server), zu dem sich Datenproduzenten (Publisher) und -nutzer (Subscriber) gleichermassen verbinden können
- ★ Dadurch bleiben die Abonnenten (Subscriber) unabhängig vom Sender (Publisher).
- ★ Die Abstimmung erfolgt mittels Topics. Topics können wie Verzeichnisse betrachtet werden, wo die Messages als einzelne Dateien abgestellt werden.

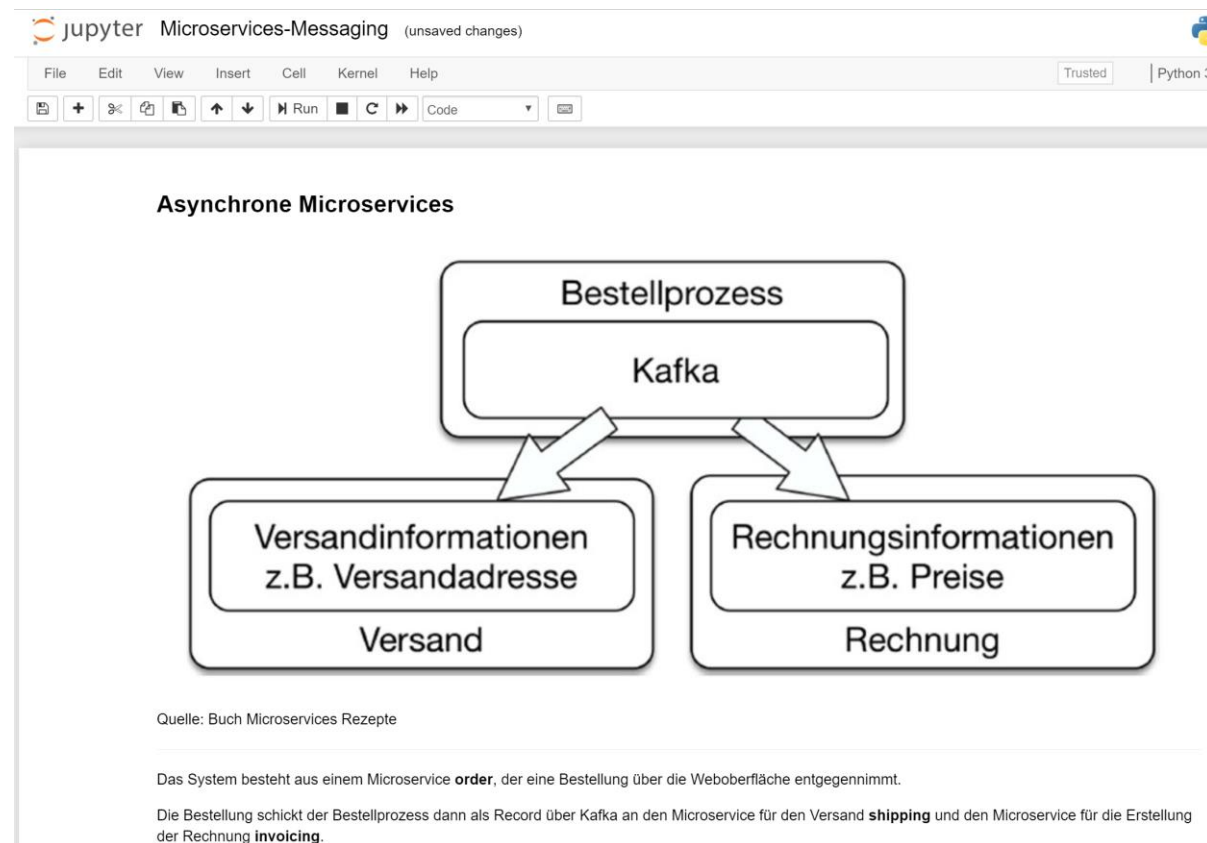
Asynchrone Microservices: Vorteile

- ★ Beim Ausfall eines Kommunikationspartners wird die Nachricht später übertragen, wenn der Kommunikationspartner wieder verfügbar ist.
- ★ Die Übertragung und auch die Bearbeitung einer Nachricht (Message) können fast immer *garantiert* werden.
- ★ Asynchrone Microservices können Messages als Events implementieren. Events bieten eine fachliche Entkopplung. Ein Event könnte beispielsweise “Bestellung eingegangen” sein. Jeder Microservice kann selbst entscheiden, wie er auf den Event reagiert.

Übung: Asynchrone Microservices (->demo ->...Messaging)

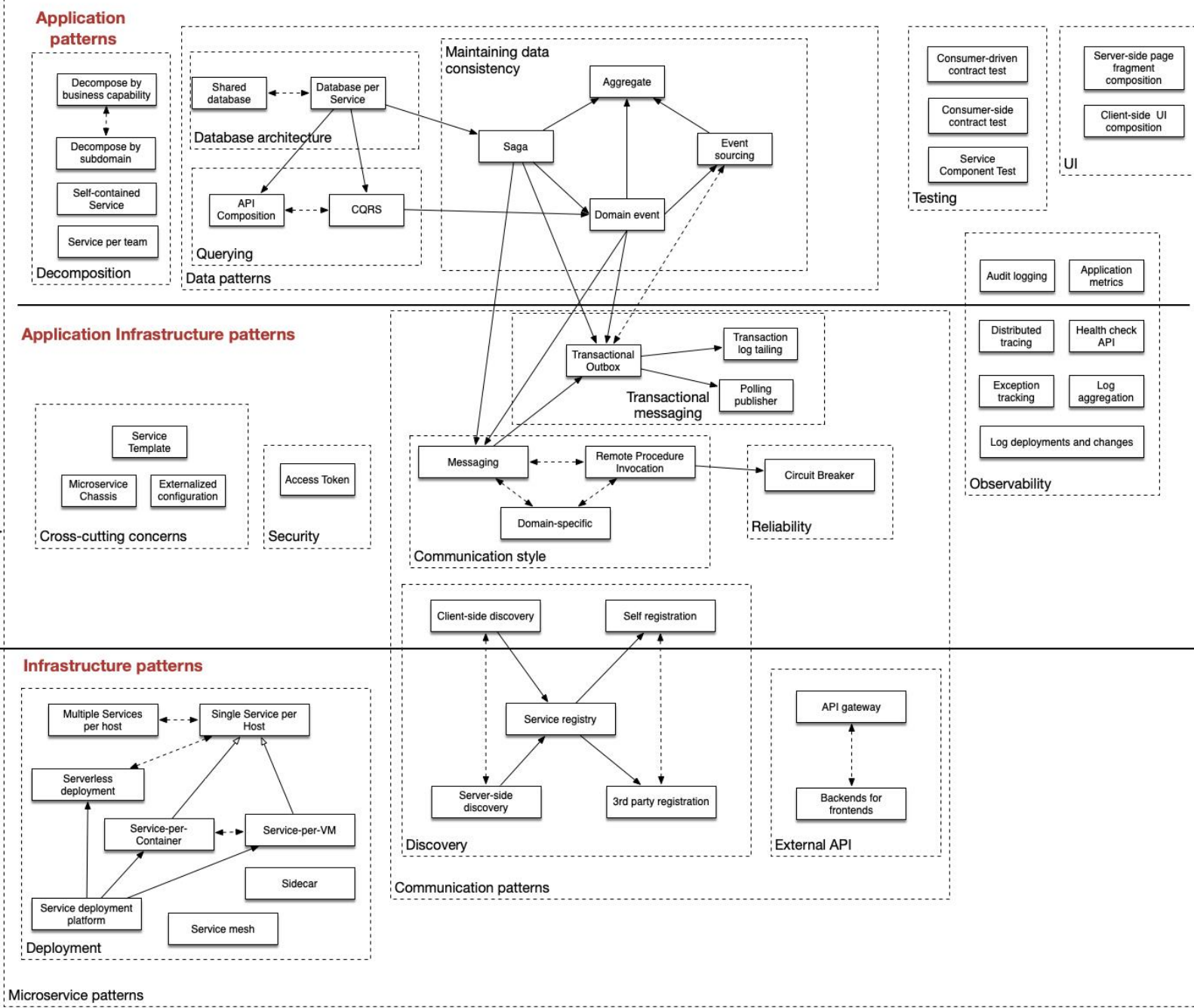
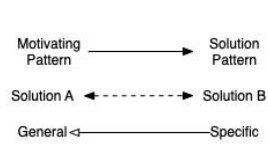
★ Fragen:

- Für welche Microservices spielt die Ausfallsicherheit keine grosse Rolle?



https://gitlab.com/ch-tbz-hf/Stud/cnt/-/tree/main/2_Unterrichtsressourcen/J#asynchrone-microservices

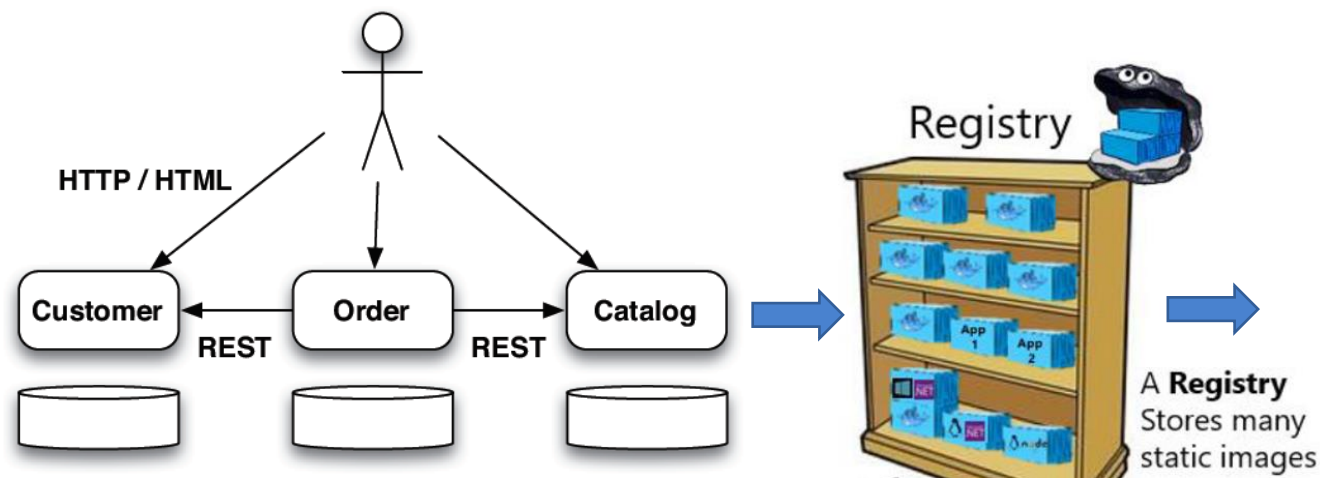
Patterns (Muster)



<http://microservices.io/patterns/microservices.html>
<https://eng.uber.com/microservice-architecture/>
 7 Microservices Best Practices for Developers

Applikation vs Microservices vs Container

- ★ Entwickler (Dev)
 - Microservices



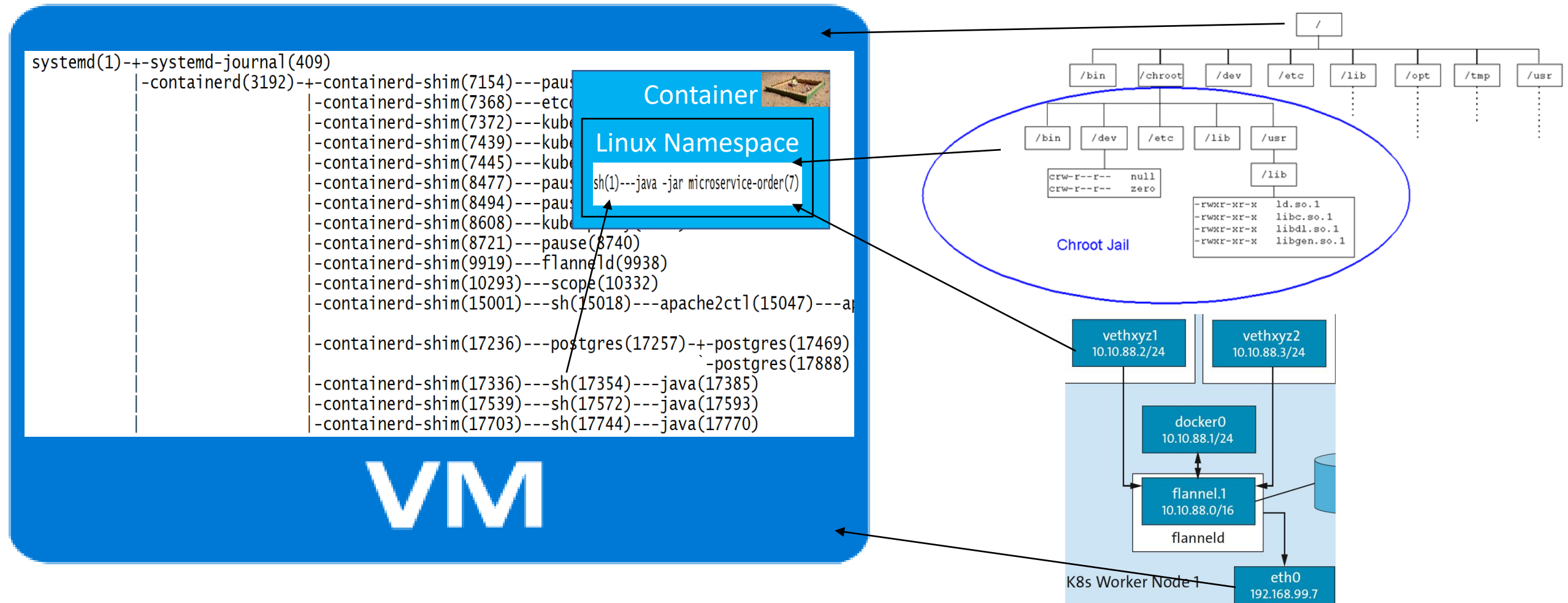
- ★ 1ne Applikation = 1:n Microservices
- ★ 1 Microservice = 1:n Container Image Versionen
- ★ 1 Microservice = 1:n Container Instanzen

- ★ Operating (Ops)
 - VMs als Behälter für Container



- ★ Container basieren auf Container Images
- ★ 1 Container Image = 1:n Container Instanzen
- ★ 1 Container Instanz = 1 Prozess

Container verwenden Linux Namespaces zur Isolation von Prozessen, Dateisystem, Netzwerk



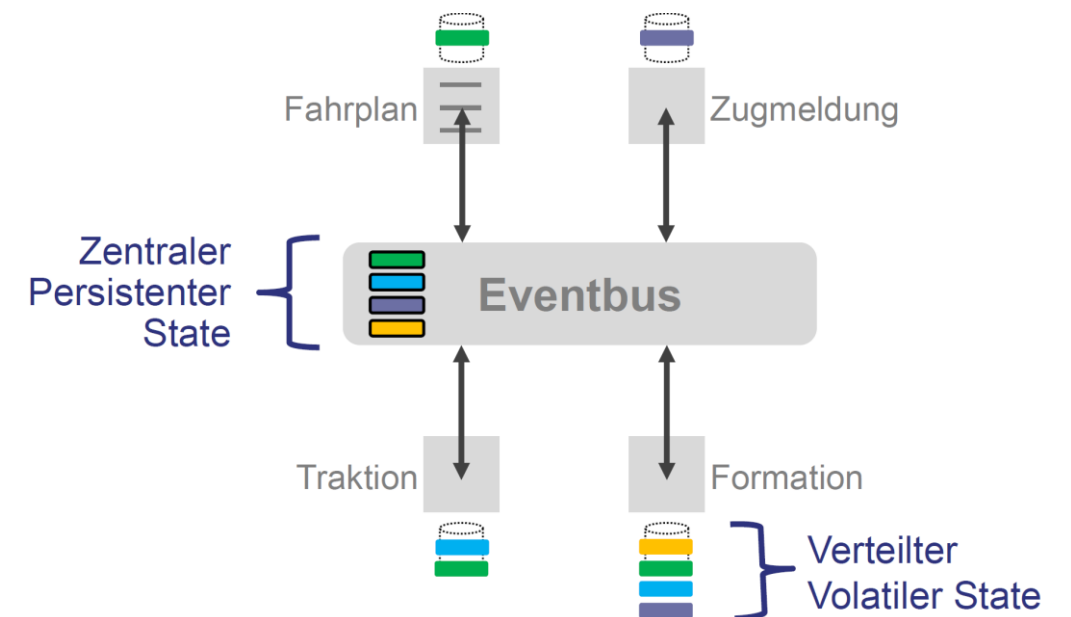
Unser Unternehmen und Microservices

★ Was passiert im Hintergrund?

- Statt monolithischer Applikationen – entkoppeln der Dienste, und einzeln als ein **Microservice**, in je einem Container betrieben und vernetzt.
- Dies ermöglicht eine massive Skalierbarkeit, Verfügbarkeit und Portierbarkeit.
- Bietet die Möglichkeit durch Modularisierung und sukzessive Aktualisierung *Big Bang Releases* zu entschärfen.

★ Aber es setzt voraus:

- Orchestrierung der Microservices untereinander
- Container Images für Microservices
- Ein hoher Grad an Automation
- Mehr Middleware (Web Cache, Message Broker für Eventbus, Workflow Systeme etc.)

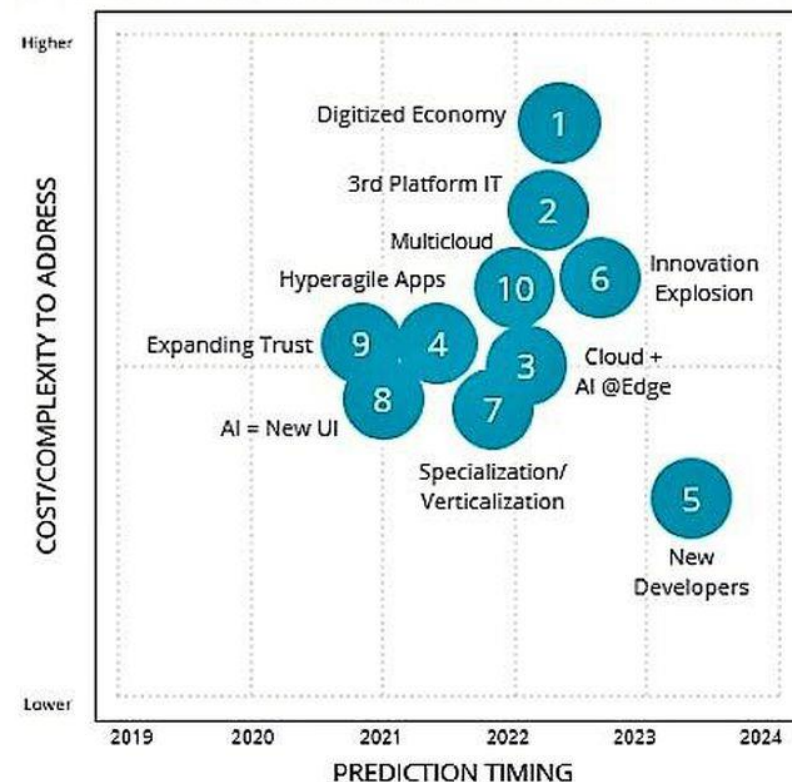


Quelle: https://www.jug.ch/html/events/2019/architektur_mit_apache_kafka_bs.html

Quelle: Buch Skalierbare Container-Infrastrukturen, Kapitel 2.2

IT-Industrie und Microservices?

IDC FutureScape: Worldwide IT Industry 2019 Top 10 Predictions



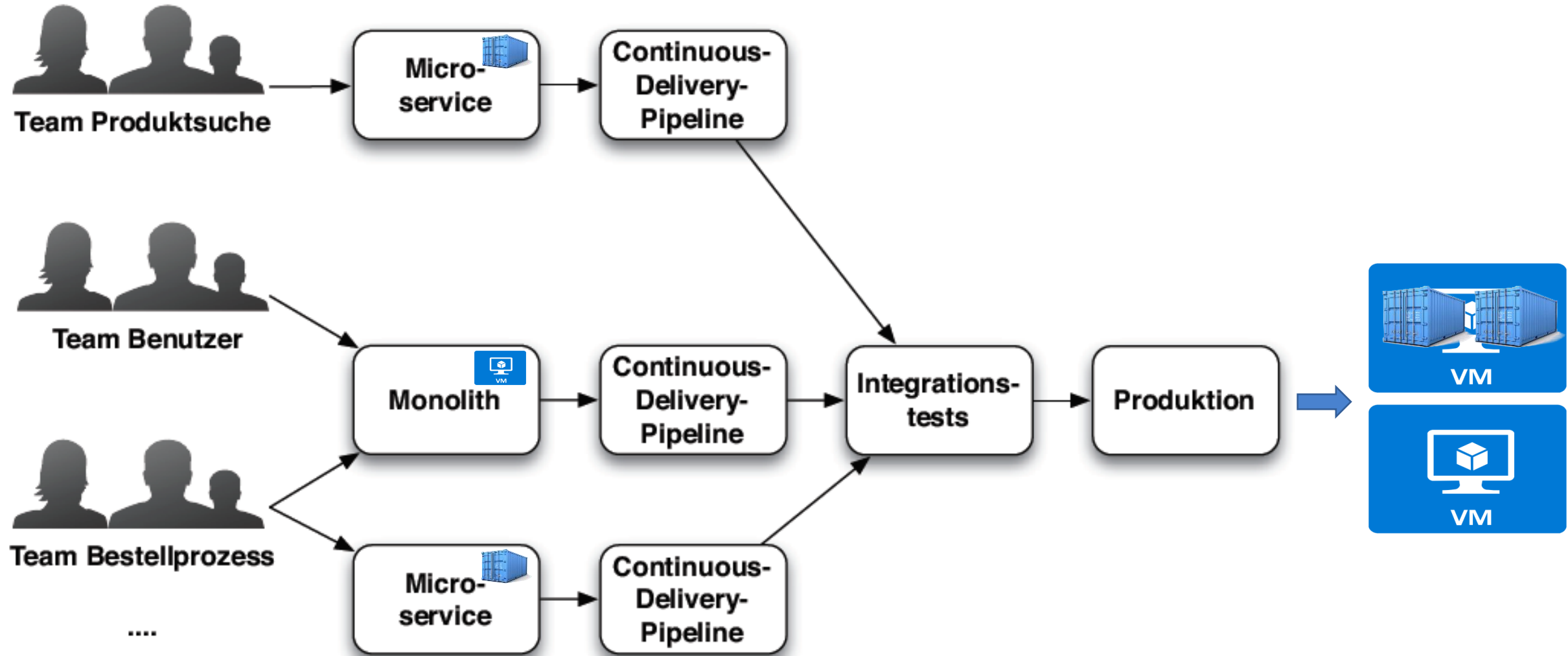
Note: Marker number refers only to the order the prediction appears in the document and does not indicate rank or importance, unless otherwise noted in the Executive Summary.

Source: IDC, 2018

★ Vorhersage 4: AppDev-Revolution / Hyperagile Apps

- Bis 2022 werden 90% aller neuen Applikationen mittels dem Architekturmuster Microservices implementiert.
- Das wird die Fähigkeit verbessern, Applikationen von Drittanbietern zu verwenden, zu debuggen, zu aktualisieren und zu nutzen.
- 35% aller Produktions-Applikationen werden Cloud-nativ sein (vereinfacht: Microservices, Container, DevOps, Kubernetes).
- Quelle: [IDC Top 10 Prognosen IT, 2019](#)

Wie bestehende Applikationen einbinden?



Perspektive der CEOs/Entscheider

★ Produkt/die Marke

- Hohe Qualität
- Umsetzung Geschäftsanforderungen in Technologie
- **Neue Variationen und Prototypen in kürzester Zeit ausrollen** (Time-to-Market).

Reflexion

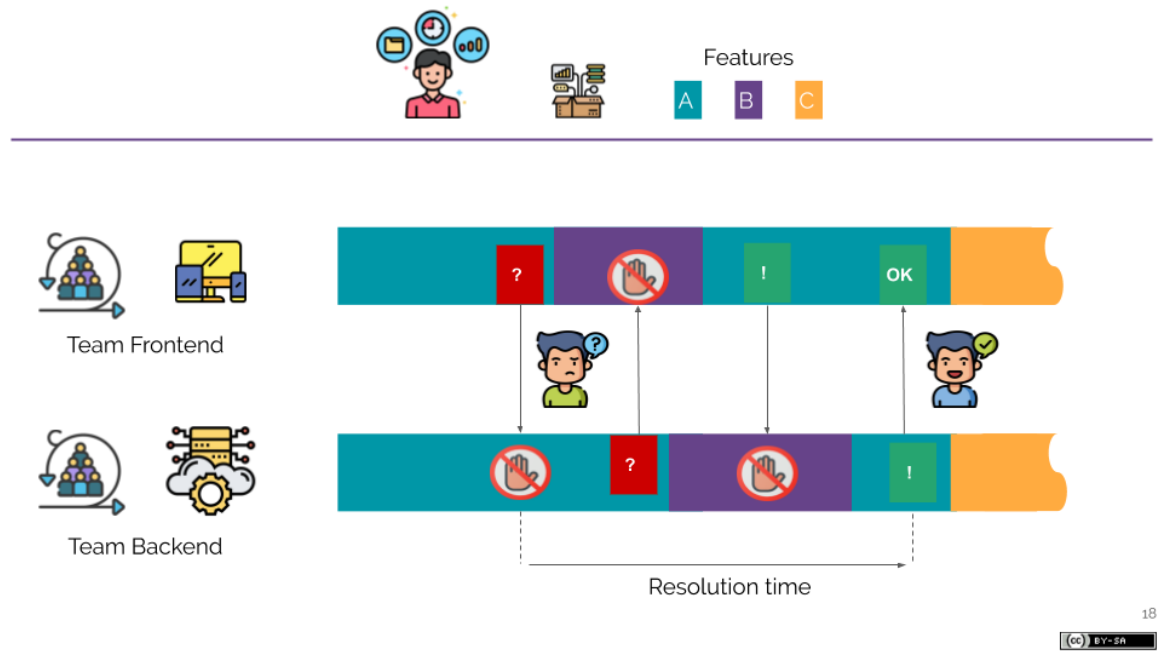
- ★ Container und Microservices verändern die IT-Landschaft Technologisch und Organisatorisch (DevOps).
- ★ Container und Microservices Beschleunigen den Time-to-Market von neuen Produkten.
- ★ Container und Microservices brauchen eine effiziente Continuous Integration / Continuous Delivery (CI/CD) Pipeline.
- ★ In der Entwicklung reden wir von Microservices meinen aber modulare Applikationen in Container Images verpackt und betrieben als Container.

Lernzielkontrolle

- ★ Sie haben einen ersten Überblick über die neue Welt der Container und Microservices.
- ★ Sie haben Microservices in einer Docker und Kubernetes Umgebung zum laufen gebracht.

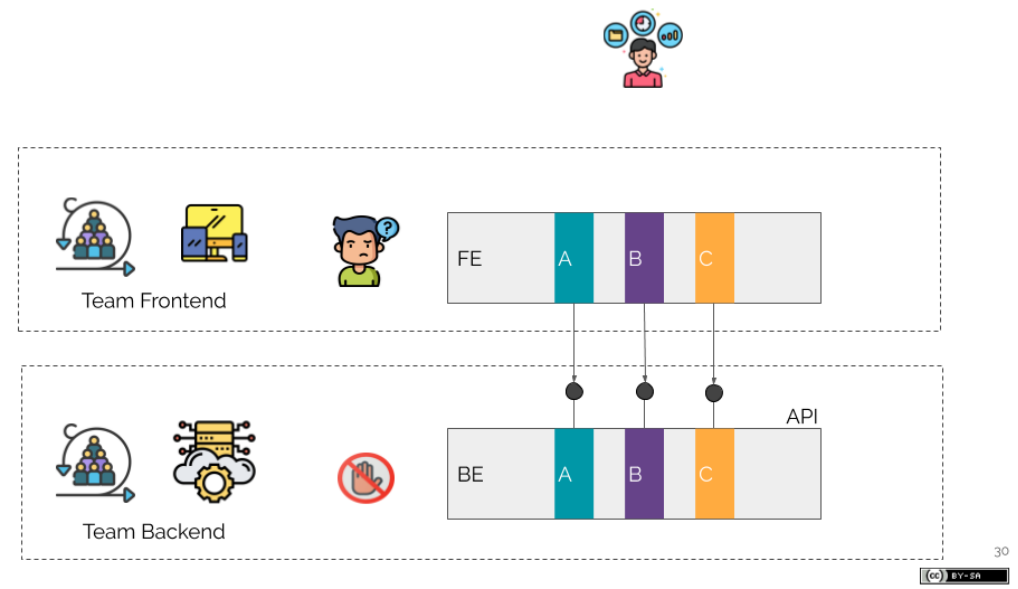
Microservices und Teambildung

★ Teambildung alt



★ Gefahr von Verzögerungen durch Abstimmung der Teams

★ Teambildung neu



★ Funktionsübergreifende Teams welche ähnlich wie Startup - agieren.

Quelle: <https://blog.mia-platform.eu/en/how-microservices-facilitate-feature-teams-work>

Domain Driven Design (DDD)

- ★ DDD ist ein Satz von Werkzeugen, die beim Entwerfen und Implementieren von hochwertiger Software helfen.
- ★ *Ihre Organisation kann nicht in allen Bereichen exzellent sein, deshalb sollte sie umsichtig wählen, womit sie herausstechen will.*
- ★ Die **strategischen Werkzeuge** von DDD helfen, die aus Wettbewerbssicht besten Entwurfs- und Integrationsentscheidungen für Ihr Geschäft zu treffen.
- ★ Die **taktischen Werkzeuge** von DDD können unterstützen, nützliche Software zu entwerfen, die das für den Geschäftserfolg notwendige Vorgehen exakt modelliert.

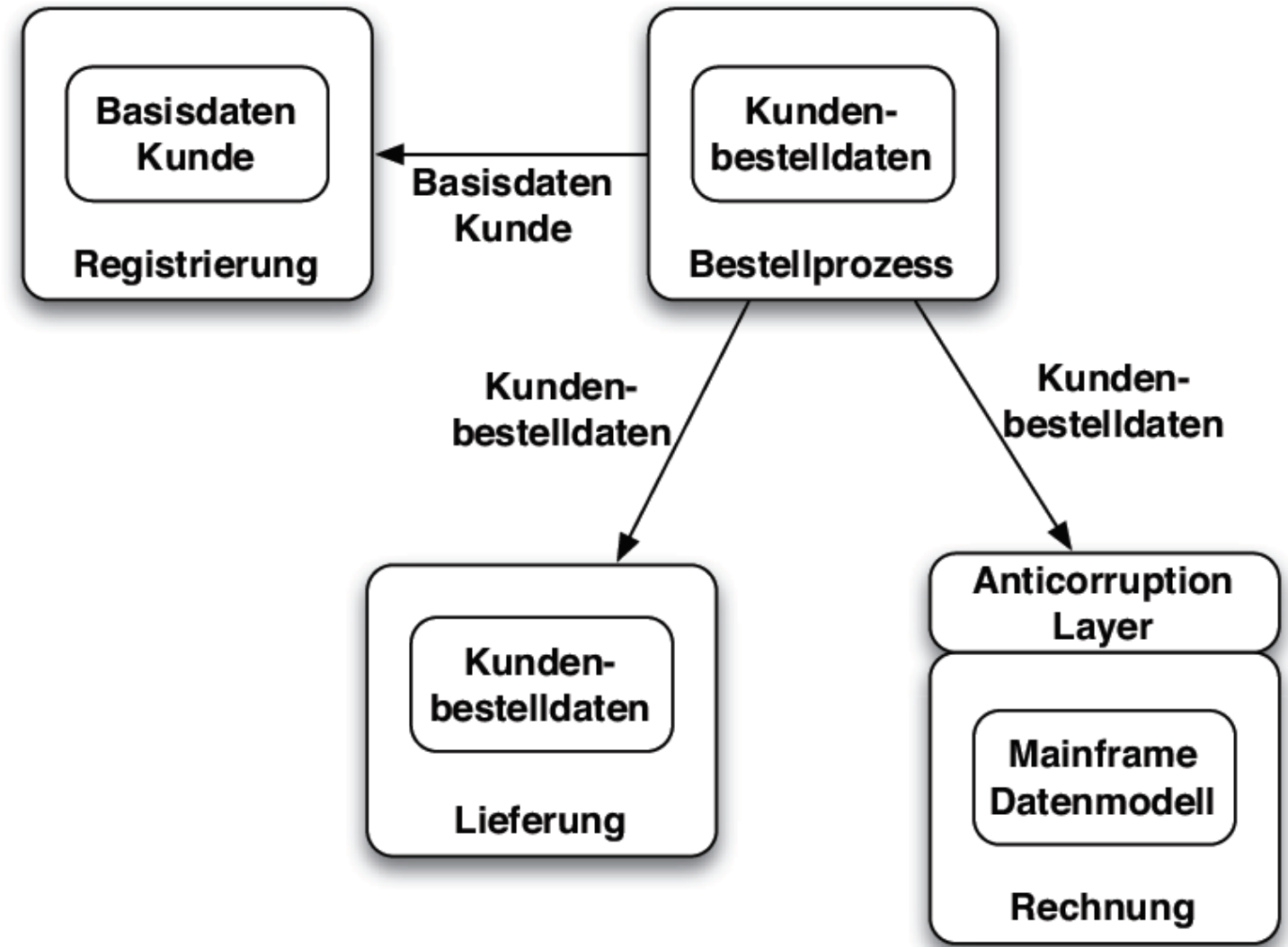
Bounded Context (dt.: begrenzter Kontext)



Quelle: Buch Microservices, Grundlagen flexibler Softwarearchitekturen

Context Mapping (dt.: Abbilden von Kontexten)

- ★ Abbildung Integration von Bounded Contexten



Quelle: Buch Microservices, Grundlagen flexibler Softwarearchitekturen

Arten von Mapping

- ★ **Partnership (dt.: Partnerschaft)** - Koordination zweier Teams mit abhängigen Zielen aber eigenem Bounded Context.
- ★ **Shared Kernel (dt.: geteilter Kern)** - Zwei (oder mehr) Teams, die sich ein kleines, aber gemeinsames Modell teilen.
- ★ **Customer-Supplier (dt.: Kunde-Lieferant)** – Lieferant stellt Kunde Modell zur Verfügung.
- ★ **Conformist-Beziehung (dt.: Konformist, Mitläufer)** – Team übernimmt Modell von Partner, weil das Modell bereits gut eingeführt ist.
- ★ **Anticorruption Layer (dt.: Antikorruptionsschicht, Antiverfälschungsschicht)** – Team baut eine Übersetzungsschicht.
- ★ **Open Host Service (dt.: offen angebotener Dienst)** – [ProgrammableWeb](#)
- ★ **Published Language (dt.: veröffentlichte Sprache)** - Wohldokumentierte Sprache zum Informationsaustausch, z.B. <http://www.ech.ch/xmlns/>.

Übung

- ★ Betrachten Sie ein Ihnen bekanntes Projekt
 - Wie sieht die Teamstruktur aus?
 - ★ Ist sie technisch oder fachlich getrieben
 - ★ Muss die Struktur für einen Microservice-Ansatz geändert werden?
 - ★ Wie muss sie geändert werden?
 - Ist die Architektur auf Teams sinnvoll verteilbar? Schliesslich soll jedes Team unabhängige fachliche Komponenten verantworten und dort Features umsetzen können.
 - ★ Welche Änderungen an der Architektur wären notwendig?
 - ★ Wie aufwendig sind die Änderungen?

Quelle: Buch Microservices, Grundlagen flexibler Softwarearchitekturen