



Hochschule Karlsruhe – Data Engineering WS 2021/2022 DSCB330 – Vorlesung 8 – Microservices



Data Integration

- Data Formats (csv, XML, json)
- Extract, Transform, Load
- Object Relation Mapper (ORM)
- Staging

Data Processing

- Relationale Datenbanken
- **nicht-relationale Datenbanken**
- Resource Description Framework (RDF)
- Ontologien
- Data Warehouse

Data Modelling

- Serialisierung
- OPC UA
- MQTT
- Pub/Sub
- **Data pipelines**
 - Apache Airflow
 - **gRPC**

Web-Service Architektur

- Front-End
- **Backend for Frontend (BFF)**
- **Micro Services**
- **Docker Container**

Security

- Security ist wichtig in allen Phasen der Softwareentwicklung und Datenbereitstellung.

Übung

- Erstellung eines Daten-Modells einer prozesstechnischen Anlage
- Statische Daten
- Dynamische Daten
- Auswertung der Daten

Ziel

- + Erstellung eines Web-Services mit folgenden Eigenschaften
- + **Benutzer**
 - Sehr gute User Experience
- + **Entwicklung**
 - Kurze Entwicklungszeiten
 - Gute Wartbarkeit
- + **Production Operations**
 - Einfaches Deployment
 - Hohe Stabilität und Verfügbarkeit
 - Skalierbar
 - Deployment möglich in der Cloud und on-premises
- + **Enterprise**
 - Kostengünstig
- + **Security**

Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

Web-Service Architektur

Docker für Production Operations



Containerization

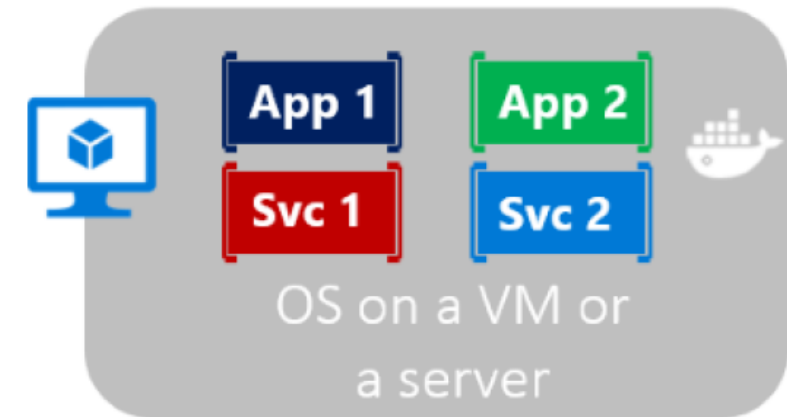
- + Applikationen bzw. Services, ihre Abhängigkeiten und die Konfiguration werden in ein Container Image gepackt.

Vorteile

- + Einfacher Unit Test
- + Einfacher Integrationstest
- + Einfaches Deployment
- + Einfache Skalierbarkeit
- + Reliability
 - Ausfall eines Containers wirkt sich nicht auf andere aus
- + Portability
 - Nutzung der Container on-premises (customer datacenter) und auf verschiedenen Cloud-Plattformen (AWS, Google, Azure, MindSphere) möglich
- + Agility
 - Entwicklung durch **unabhängige** Teams

- + Isolation
 - zwischen Entwicklung und Operations (DevOps)
- + Docker Container haben einen kleineren Footprint als virtuelle Maschinen

Docker Host



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

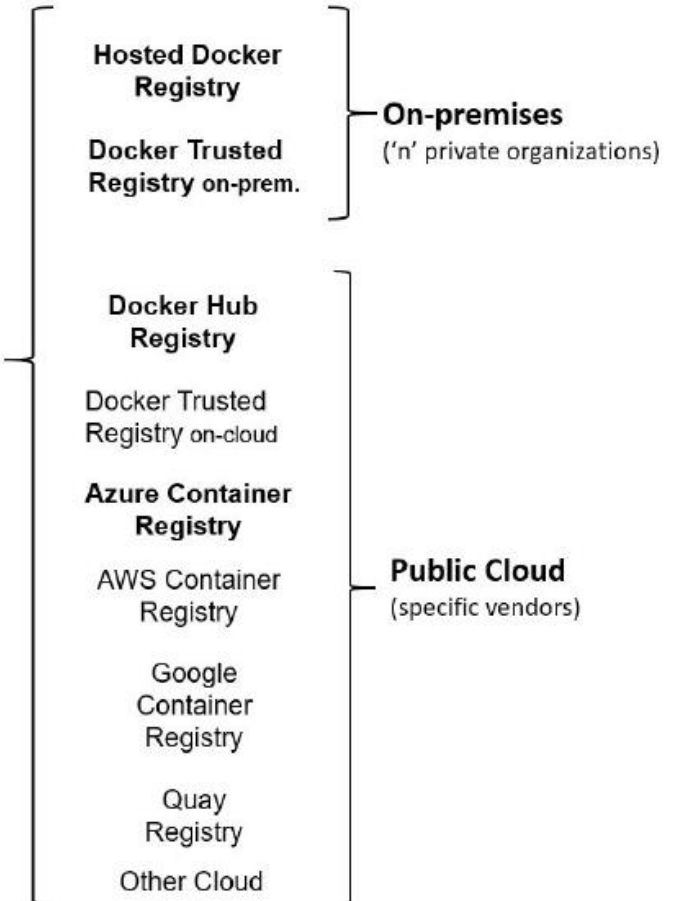
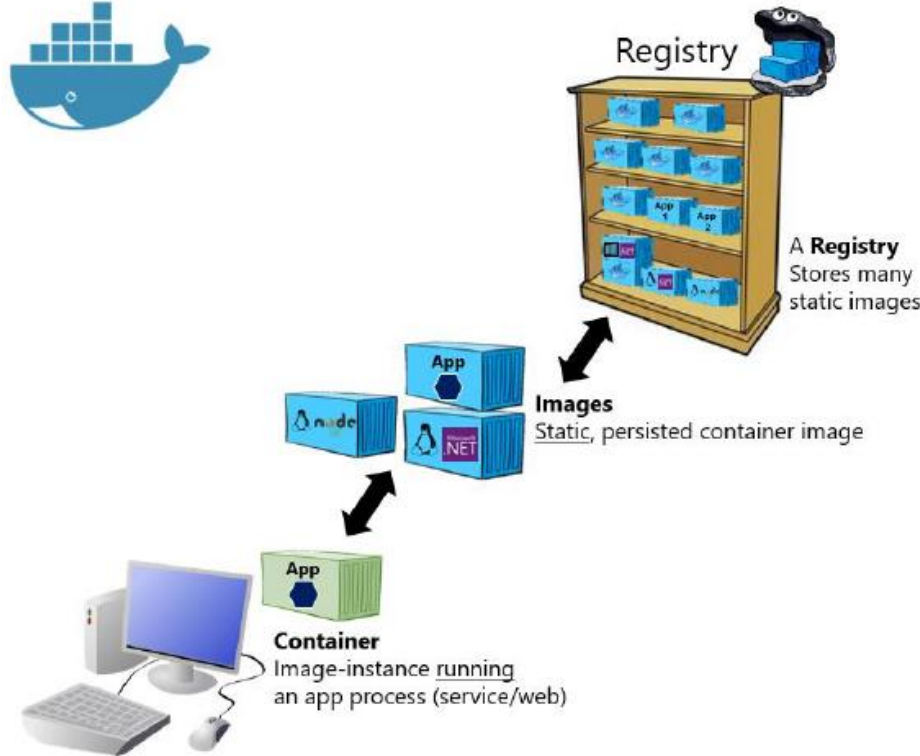
Web-Service Architektur

Verwaltung der Docker Container



- + Für erweiterte Sicherheitsanforderungen können docker images in einer private image registry verwaltet werden.

Basic taxonomy in Docker



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

5

Web-Service Architektur

Unit Test und Integrationstest

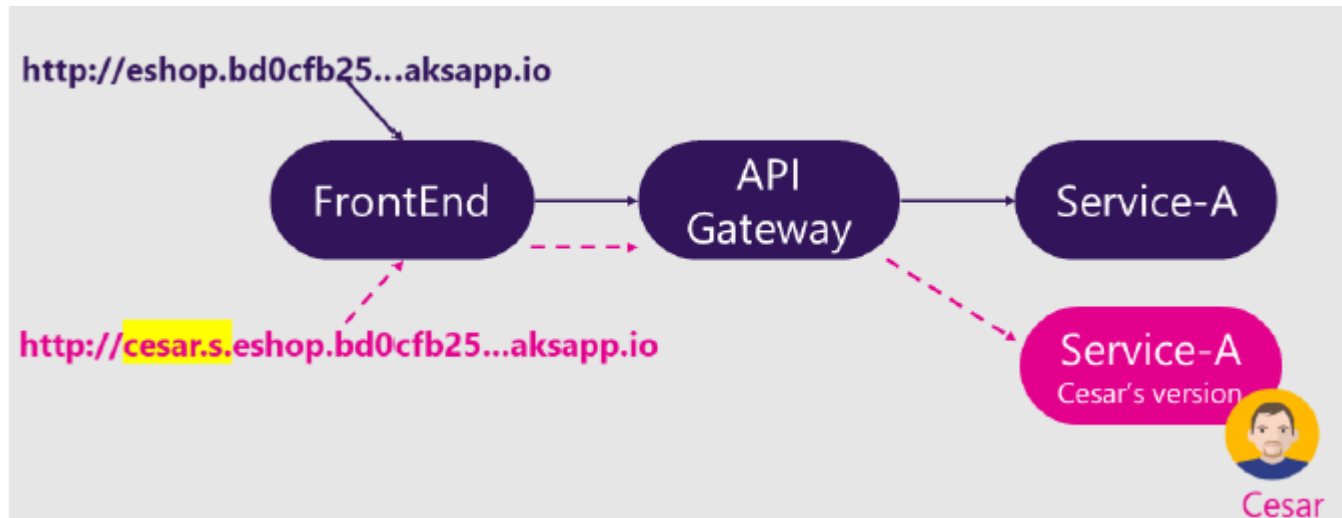


Unit Test

- + Test eines Docker Containers

Integrationstest

- + Möglich nahe am Produktivsystem
- + Nutzung von „spaces“ für die Entwicklung, die das Testen von Services integriert in das globale Deployment erlauben.



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

Microservice architecture

- + Aufbau einer Server-Architektur aus kleinen Services
- + Nutzung von HTTP, AMQP und gRPC zur Kommunikation zwischen Services
- + Jeder Service implementiert eine end-to-end-domain Fähigkeit (capability) innerhalb einer context boundary (context: Zusammenhang, boundary: Grenze)
- + Jeder Service kann autonom entwickelt und unabhängig ausgeführt werden.
- + Jeder Microservice bestimmt über seine Daten

Vorteile

- + Skalierbarkeit
- + Bessere Wartbarkeit in großen, komplexen und hochskalierbaren Systemen
- + Voneinander unabhängig ausführbare Services
- + Continuous integration and continuous delivery
- + Test and run microservices in isolation

- + Unabhängige Weiterentwicklung unter Berücksichtigung vorhandener Schnittstellen (interfaces and contracts)
- + Einfaches Monitoring, Health checks, Watch dogs für jeden Microservice
- + Skalierbare Infrastruktur (über Cloud und Orchestrators)
- + Schnelle Entwicklung und Auslieferung durch unterschiedliche, parallel arbeitende Teams
- + DevOps and CI/CD (Developing and Operations, Continuous Integration and continuous deployment)

Domain-Driven Design

- + Herangehensweise zur Modellierung komplexer Software
- + Iterative Softwareentwicklung
- + Enge Zusammenarbeit zwischen Entwicklern und Fachexperten

Bounded Context (Kontextgrenzen)

- + beschreiben die Grenzen jedes Kontexts in vielfältiger Hinsicht wie beispielsweise Teamzuordnung, Verwendungszweck, dahinter liegende Datenbankschemata.

Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>; https://de.wikipedia.org/wiki/Domain-driven_Design

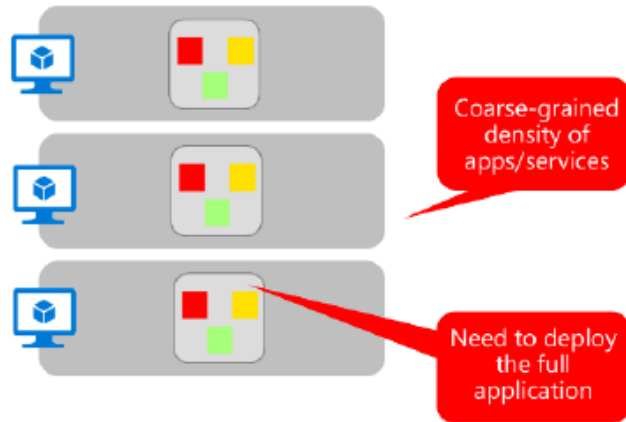
Web-Service Architektur

Monolithic deployment versus Microservices approach



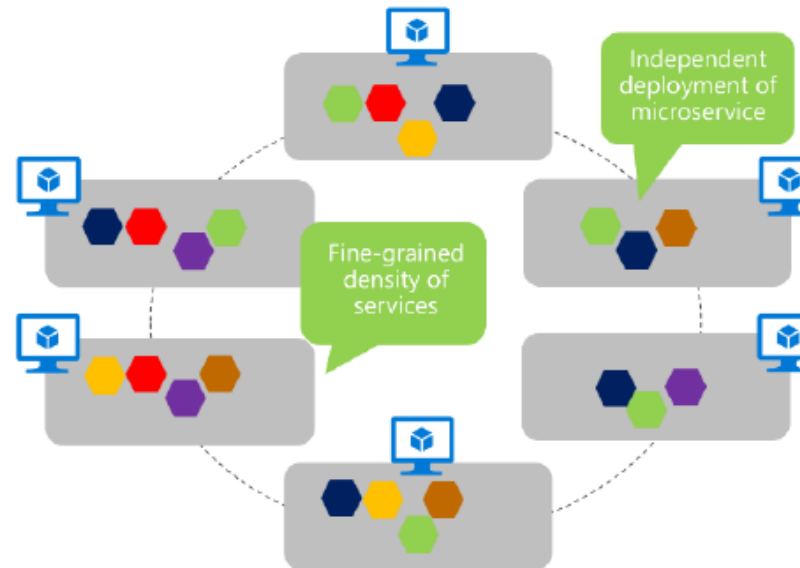
Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

Data sovereignty per microservice

- + Jeder Microservice verfügt über die Logik und Daten seiner Domäne
- + Im traditionellen Ansatz gibt es eine gemeinsame Datenbank für alle Services
- + Im Ansatz für Microservices hat jeder Microservice eine eigene Datenbank für seine Modelle und Daten.

Vorteile des traditionellen Ansatzes

- + Der monolithische Ansatz mit typischerweise einer relationalen Datenbank erlaubt eine einfache Umsetzung von ACID Transaktionen unter Nutzung von SQL

Nachteile des Microservice Ansatzes

- + Bei ACID Transaktionen muss berücksichtigt werden, dass jeder Microservice seine eigenen Daten verwaltet

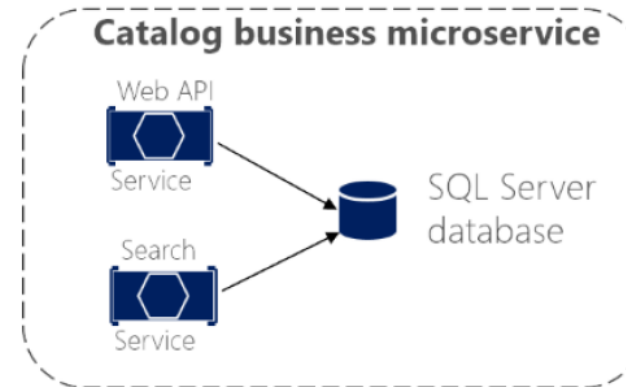
ACID

- + atomicity, consistency, isolation, durability

Vorteile des Microservice Ansatzes

- + Kapselung der Daten stellt sicher, dass Microservices nur schwach gekoppelt und sich unabhängig voneinander entwickeln können.
- + Jeder Microservice kann für seinen Zweck spezialisierte Datenbanken nutzen (SQL, Graph, nosql)
- polyglot-persistent architecture

Verschiedene Services teilen sich eine Datengrundlage, wenn sie auf den gleichen Bounded Context (business domain) zugreifen.



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

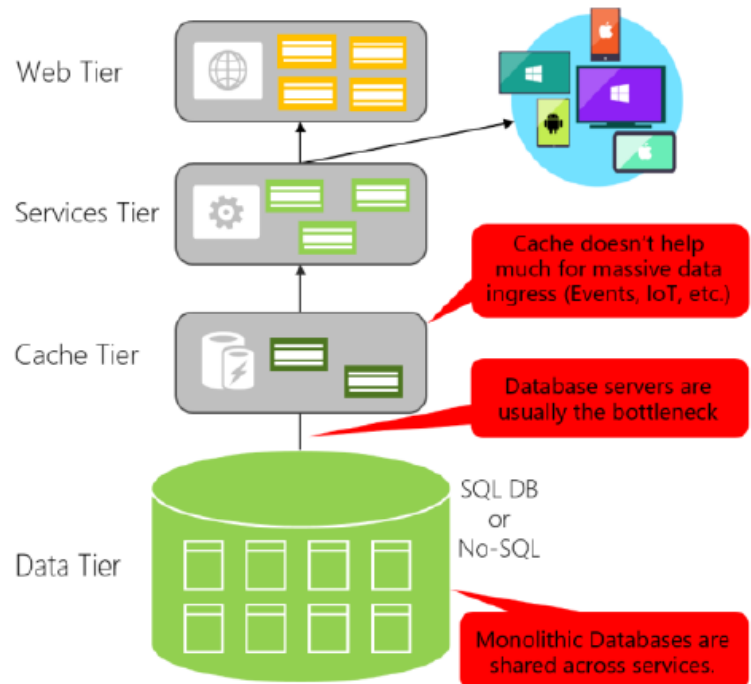
Web-Service Architektur

Data in traditionellem Umfeld und im Microservices-Ansatz



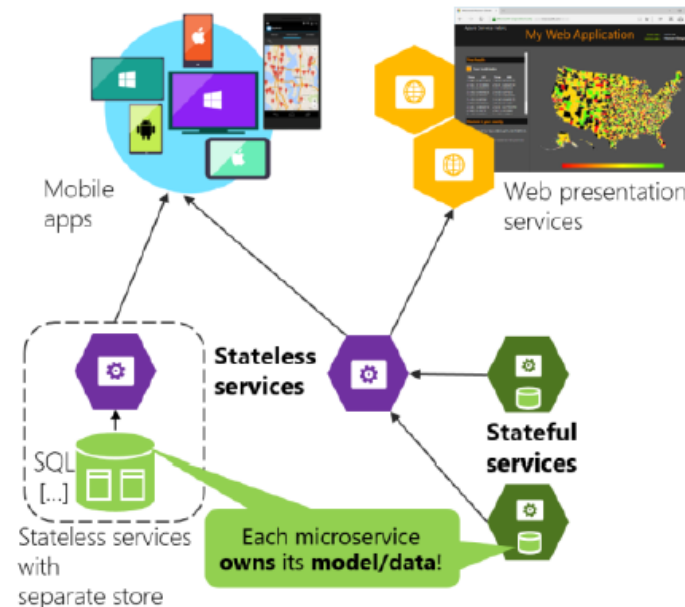
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies



Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

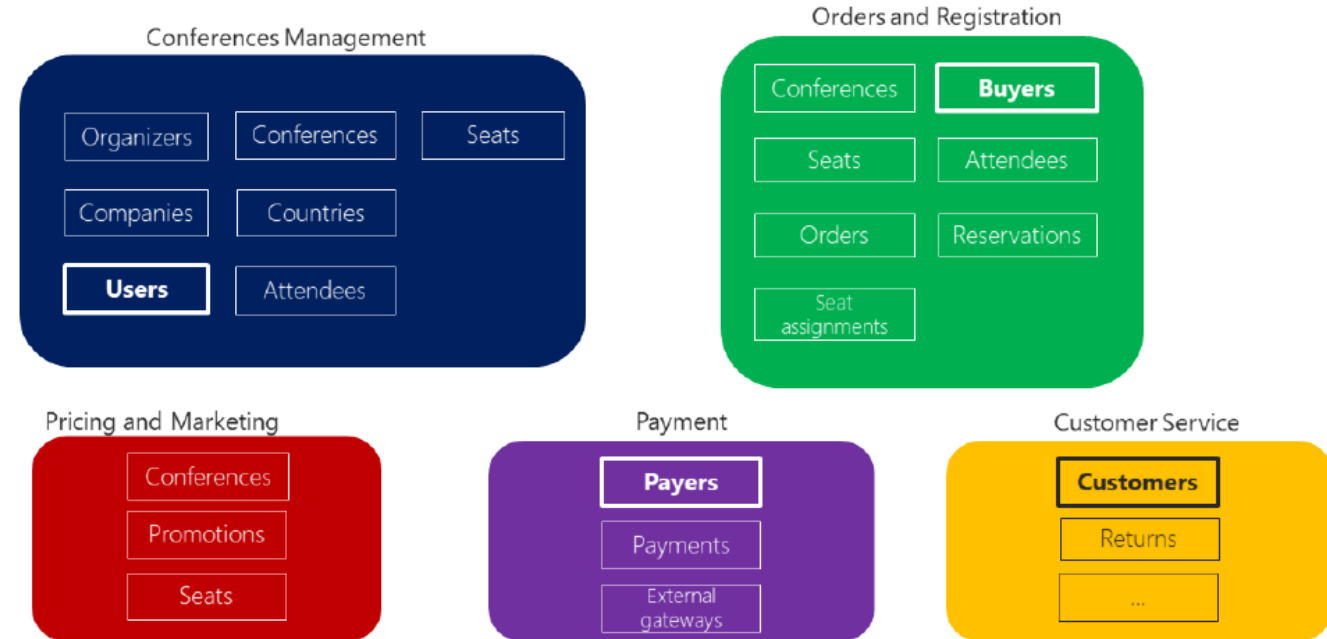
Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

10

Definition der Grenzen von Microservices

- + Ein Ansatz besteht darin, Grenzen entlang von Business terms zu ziehen
- + Beispiel: Unterschiedliche Bezeichnungen für einen User
 - User im Identitätsservice
 - Customer in Customer-Relationship-Management-Service
 - Buyer im Bestell-Kontext
- + Grenzen von Microservices können durch das Bounded Context pattern gefunden werden (Sam Newman)
- + Applikationen reflektieren die sozialen Grenzen der Organisation, durch die sie entwickelt wurden (Conway's law).



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

11

Abfragen über Grenzen von Microservices

- + Vermeidung von unnötiger Kommunikation
- + **API Gateway**
 - aggregiert Informationen von verschiedenen Microservices
- + **Materialized View pattern**
 - Erzeugung einer read-only-Tabelle mit Daten aus mehreren Microservices
- + Beispiel
 - Verkaufszahlen für Elektronikprodukte inkl. Anzahl der Kunden, die diese Käufe getätigt haben

Order table

Partition key	Row key	Order date	Shipping address	Total invoice	Order status
001 (Customer ID)	1 (Order ID)	11082013	One Microsoft way Redmond, WA 98052	\$400	In process
005	2	11082013	One Microsoft way Redmond, WA 98052	\$200	Shipped

OrderItem table

Partition key	Row key	Product	Unit Price	Amount	Total
1 (Order ID)	001_1 (OrderItem ID)	XX	\$100	2	\$200
1	001_2	YY	\$40	5	\$200
2	002_1	ZZ	\$200	1	\$200

Customer table

Partition key	Row key	Billing Information	Shipping address	Gender	Age
US East (region)	001 (Customer ID)	*****0001	One Microsoft way Redmond, WA 98052	Female	30
US East	002	*****2006	One Microsoft way Redmond, WA 98052	Male	40

Materialized View

Partition key	Row key	Product Name	Total sold	Number of customers
Electronics (Product category)	001 (Product ID)	XX	\$30,000	500
Electronics	002	YY	\$100,000	400

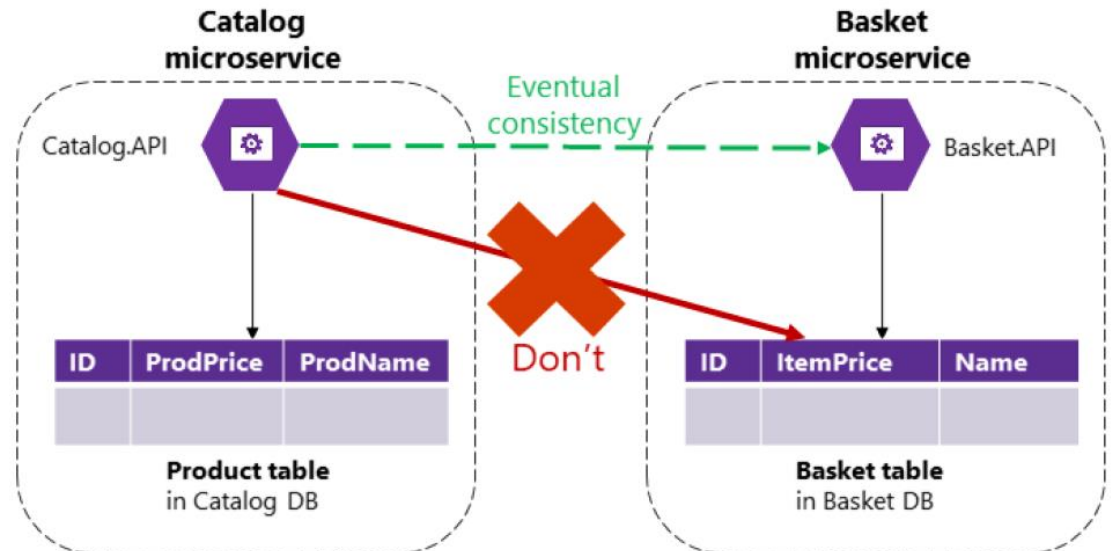
Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>; <https://docs.microsoft.com/en-us/azure/architecture/patterns/materialized-view>

Erhaltung der Konsistenz über verschiedene Microservices

- + Jeder Microservice ist für seine Daten verantwortlich
- + Daten müssen dennoch über verschiedene Microservices konsistent gehalten werden
- + **Beispiel**
- + Microservice 1: Katalog-Service inklusive Preise
- + Microservice 2: Bestell-Service mit aktuellen Preisen
- + **Herausforderung**
- + Eine Änderung der Preise im Katalog muss vom Bestell-Service aktualisiert werden. Die Aktualisierung muss im UI dem Kunden transparent gemacht werden.
- + CAP theorem:
Availability contrasts with ACID strong consistency
- + Most microservice-based scenarios demand availability and high scalability as opposed to strong consistency.

+ Lösung

- + eventual consistency
- informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value
- + Event-driven communication und publish-subscribe-systems



Databases are private per microservice

Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>; https://en.wikipedia.org/wiki/Eventual_consistency

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

13



Kommunikation über Grenzen von Microservices

- + Komponenten können ausfallen
- + Ausfälle von Teilen dürfen nicht zum Ausfall des gesamten Systems führen
- + Microservices sollten nicht über Abfragen implizit gekoppelt werden
- + Verkettete Abfragen über Microservices hinweg sollen vermieden werden
- + Nutzung asynchroner Abfragen
- + Microservices sollen autonom handeln können (d.h. nicht von anderen Microservices abhängen)

Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

14



Web-Service Architektur



Synchrone versus asynchrone Kommunikation

+ HTTP: synchrones Protokoll

+ AMQP: asynchrones Protokoll

Anti-pattern

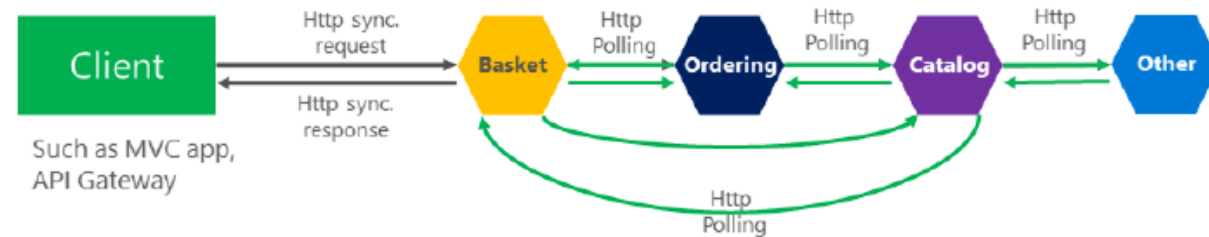
Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)



"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

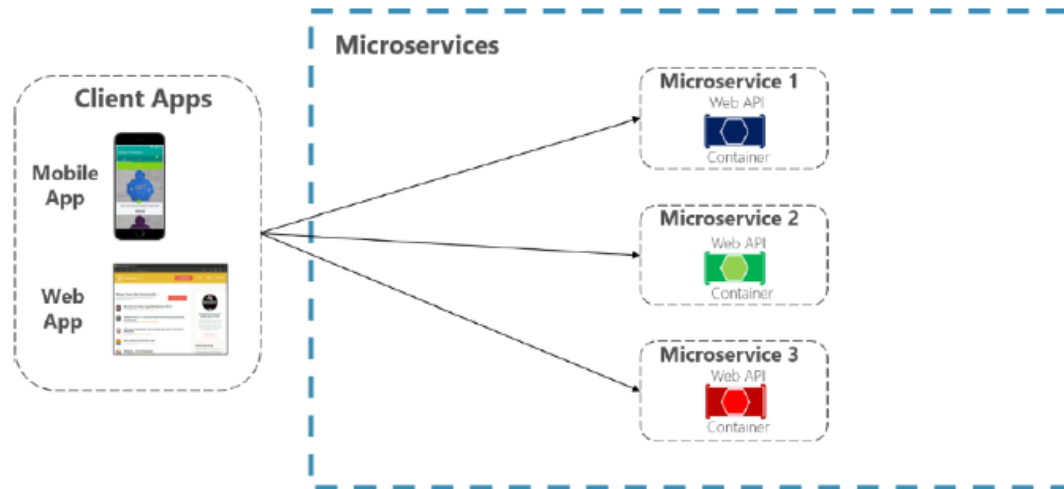
Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

15

Direct client-to-microservice communication

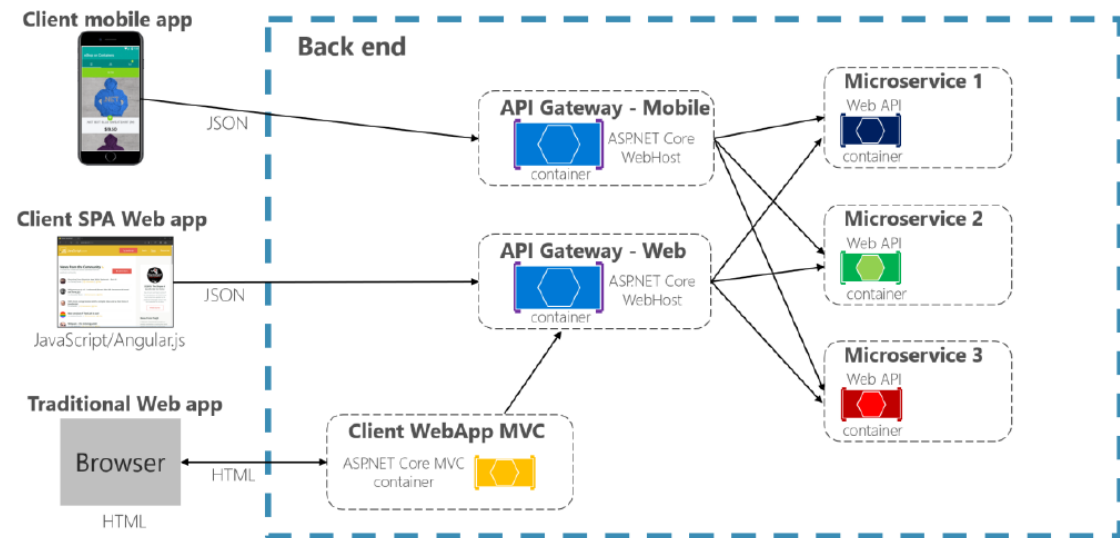
- + Load Balancer verteilt die Anfragen an die verschiedenen Microservices
- + Hohe Anzahl von round-trips verbunden mit Latenz und Komplexität in der Client App
- + Authentifizierung der Client App gegenüber jedem Microservice
- + Anforderungen einer App für mobile Geräte unterscheiden sich von Web-Applikationen



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

The API gateway pattern (backend-for-frontend)

- + Zwischenschicht zwischen der Client App und den Microservices zur Bündelung der Abfragen
- + Verringerung der round-trips
- + Client App ist nicht an verschiedene Microservices gebunden, dadurch werden z.B. Updates von Client Apps reduziert
- + Bessere Security: Client App kommuniziert nur über eine Schnittstelle

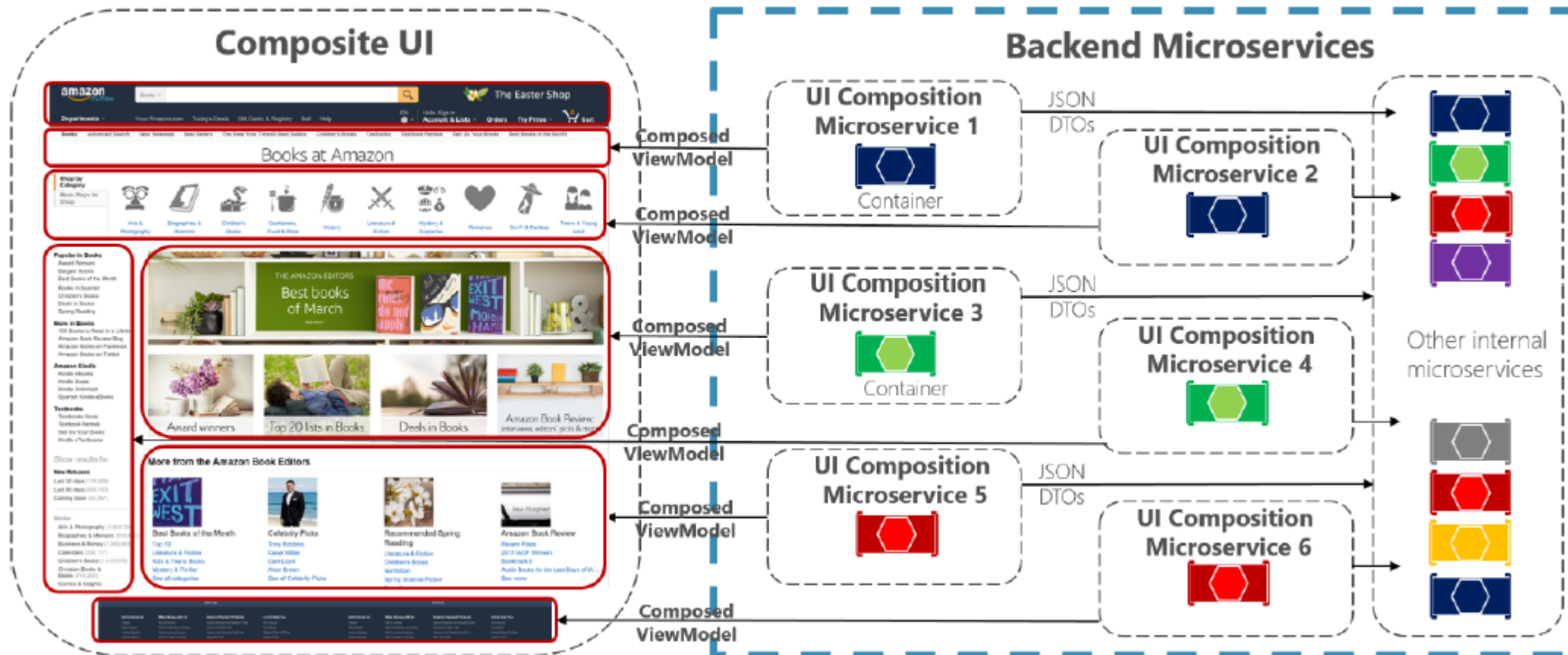


Web-Service Architektur



Composite UI generated by microservices

- + Jeder Service ist verantwortlich für ein Teil des UI



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>

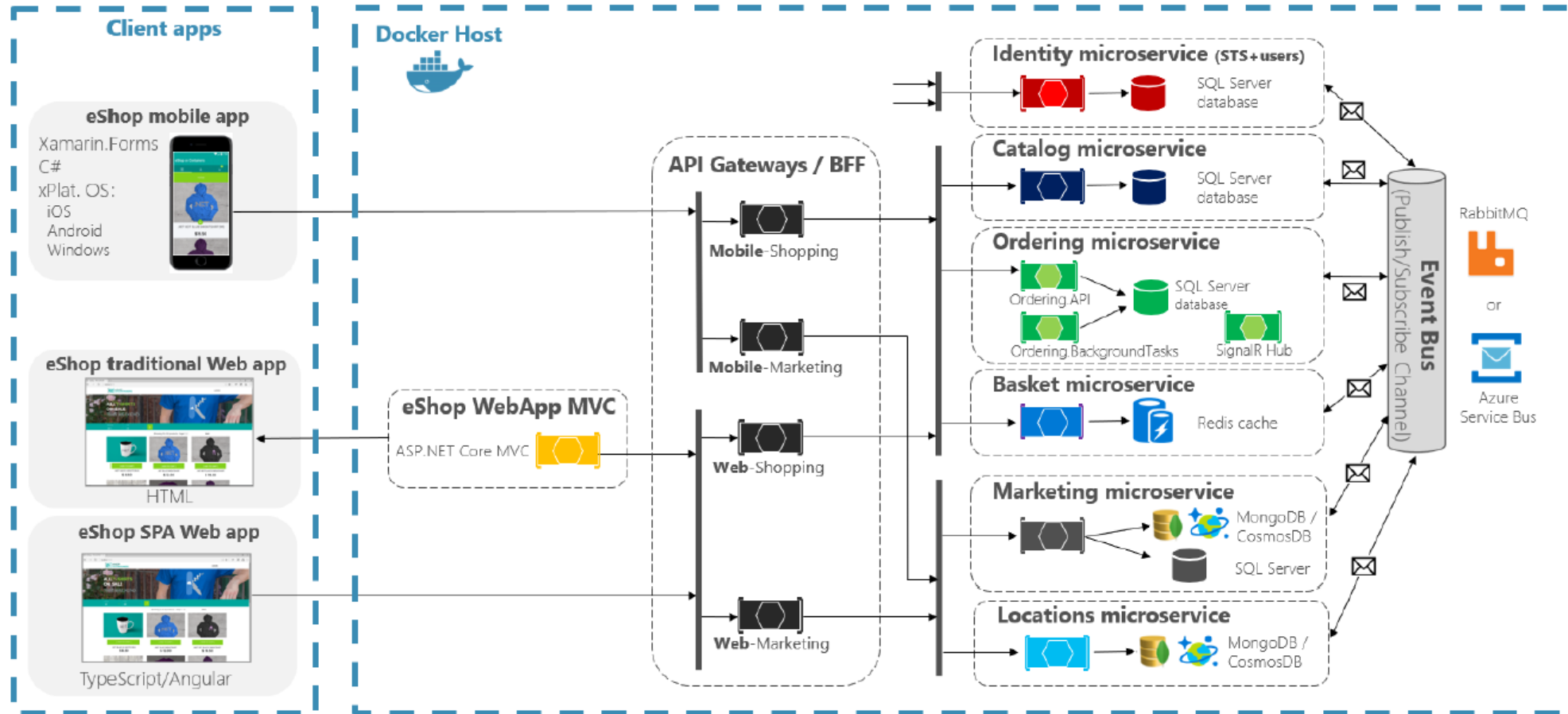
Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

17

Web-Service Architektur

Beispiel für eine Gesamtarchitektur



Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf> (eShopOnContainers)

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

18



Literatur

- + .NET Microservices: Architecture for Containerized .NET Applications, <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>
- + Containerized Docker Application Lifecycle with Microsoft Platform and Tools, <https://dotnet.microsoft.com/download/e-book/microservices-devops/pdf>

Quelle: <https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf>; <https://dotnet.microsoft.com/download/e-book/microservices-devops/pdf>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

18.11.2021

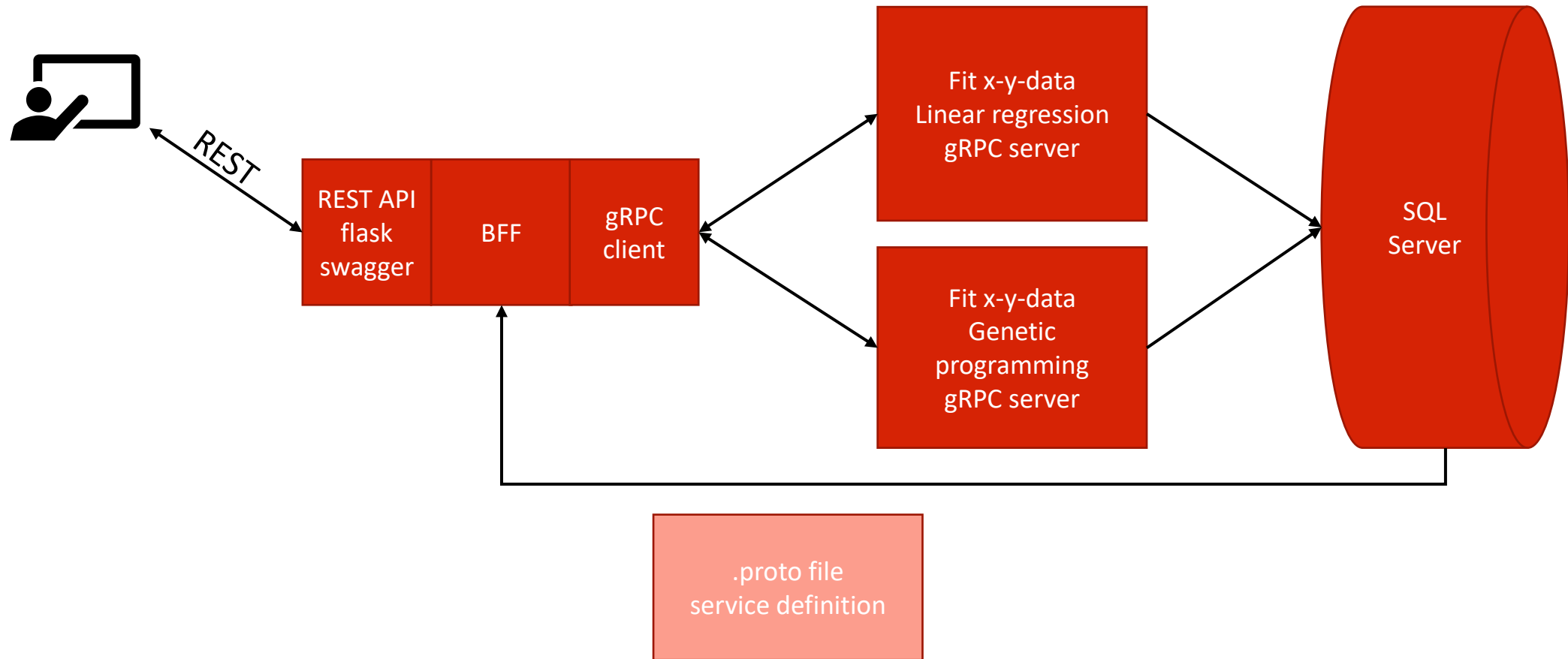
19



Web-Service Architektur

Übung 26

+ Übung 26-web-service



Web-Service Architektur

Übung 26 – Datenbanken



Database 1 (created by BFF)

HKA_functionguessing

- dbo.datapoints
 - Spalten
 - Id (int, Nicht NULL)
 - FunctionId (int, Nicht NULL)
 - x (float, Nicht NULL)
 - y (float, Nicht NULL)

	Id	FunctionId	x	y
1	1	1	-5	-11
2	2	1	-4,98498498498499	-10,954954954955
3	3	1	-4,96996996996997	-10,9099099099099
4	4	1	-4,95495495495496	-10,8648648648649

dbo.Functions

- Spalten
 - Id (int, Nicht NULL)
 - Name (nvarchar(max), Nicht NULL)
 - Description (nvarchar(max), NULL)

	Id	Name	Description
1	1	$y=3*x+4$	linear function
2	2	$y=3*x^2+1$	quadratic function
3	3	$y=2*(x+1)^3+3$	cubic function

Database 3 owned by microservice genetic programming

HKA_gp

- dbo.gpresult
 - Spalten
 - Id (int, Nicht NULL)
 - FunctionId (int, Nicht NULL)
 - scheduled (bit, Nicht NULL)
 - ready (bit, Nicht NULL)
 - function (nvarchar(max), NULL)

	Id	FunctionId	scheduled	ready	function
1	7	1	0	1	$3*x + 4.0$
2	8	2	0	1	$3*x^2 + 1.0$
3	9	3	0	1	$2*(x + 2)*(x*(x + 1) + 1.0)$

Database 2 owned by microservice linear regression

HKA_linearregression

- dbo.linregresult
 - Spalten
 - Id (int, Nicht N
 - FunctionId (int, Nicht NULL)
 - scheduled (bit, Nicht NULL)
 - ready (bit, Nicht NULL)
 - function (nvarchar(max), NULL)

	Id	FunctionId	scheduled	ready	function
1	10	1	0	1	$y=3.0000000000$
2	11	2	0	1	$y=-2.9999999999$
3	12	3	0	1	$y=52.707840711$

Web-Service Architektur

Übung 26 – Vorbereitung der Umgebung



- + Übung 26-web-service
- + Erstellen eines virtual environments für jeden service (BFF, genetic_programming, linear_regression)
 - z.B. Wechsel in das Verzeichnis src/BFF
`C:\Python310\python.exe -m venv venv`
erzeugt ein virtual environment im Verzeichnis `venv`.
 - Aktivierung des virtual environments
`.\venv\Scripts\Activate.ps1`
 - Upgrade von pip
`python -m pip install --upgrade pip`
 - Installation der Pakete (`python -m pip install ...`)
siehe `requirements.txt`
- + Wechsel in das Verzeichnis `src/python_proto`
- + Optional: Kompilieren der proto-Datei:
`python -m grpc_tools.protoc -I../.. /protos --python_out=. --grpc_python_out=. ..\..\protos\mathfunc.proto`
- Zwei Dateien werden erzeugt:
`mathfunc_pb2.py` und `mathfunc_pb2_grpc.py`
- Kopieren der beiden Dateien in jeden Service (BFF, genetic_programming, linear_regression)
- + Anpassen der Datenbankbindung in `config.son` für jeden Service (BFF, genetic_programming, linear_regression)
- + Öffnen des Verzeichnisses `src\BFF` mit Visual Studio Code
- + Falls Visual Studio Code das erzeugte virtual environment nicht erkennt, Auswahl des virtual environment



Web-Service Architektur

Übung 26 – BFF und Swagger



- + Übung 26-web-service
- + Starten des Backend-for-frontend in Windows Powershell
`python .\flask_BFF.py`
- + The swagger API documentation may be accessed via
<http://localhost:5000/apidocs/>
- + Swagger dient als online-Dokumentation des REST-API
- + Source code

```
from flasgger import Swagger
# Create Flask app Instance
app=create_flask_app(config)
Swagger(app)
```

```
"""Endpoint describes available data sets.
Information is retrieved from database.
---
responses:
    200:
        description: Information about ...
"""
```

The screenshot shows the Swagger UI for the endpoint `GET /api/v1/datasets`. The description is "Endpoint describes available data sets. Information is retrieved from database." There are no parameters. The response section shows a single response with status code 200 and description "Information about available data sets." The response content type is set to `application/json`.

Code	Description
200	Information about available data sets.

Quelle: <https://pypi.org/project/flasgger/0.5.4/>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 8 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

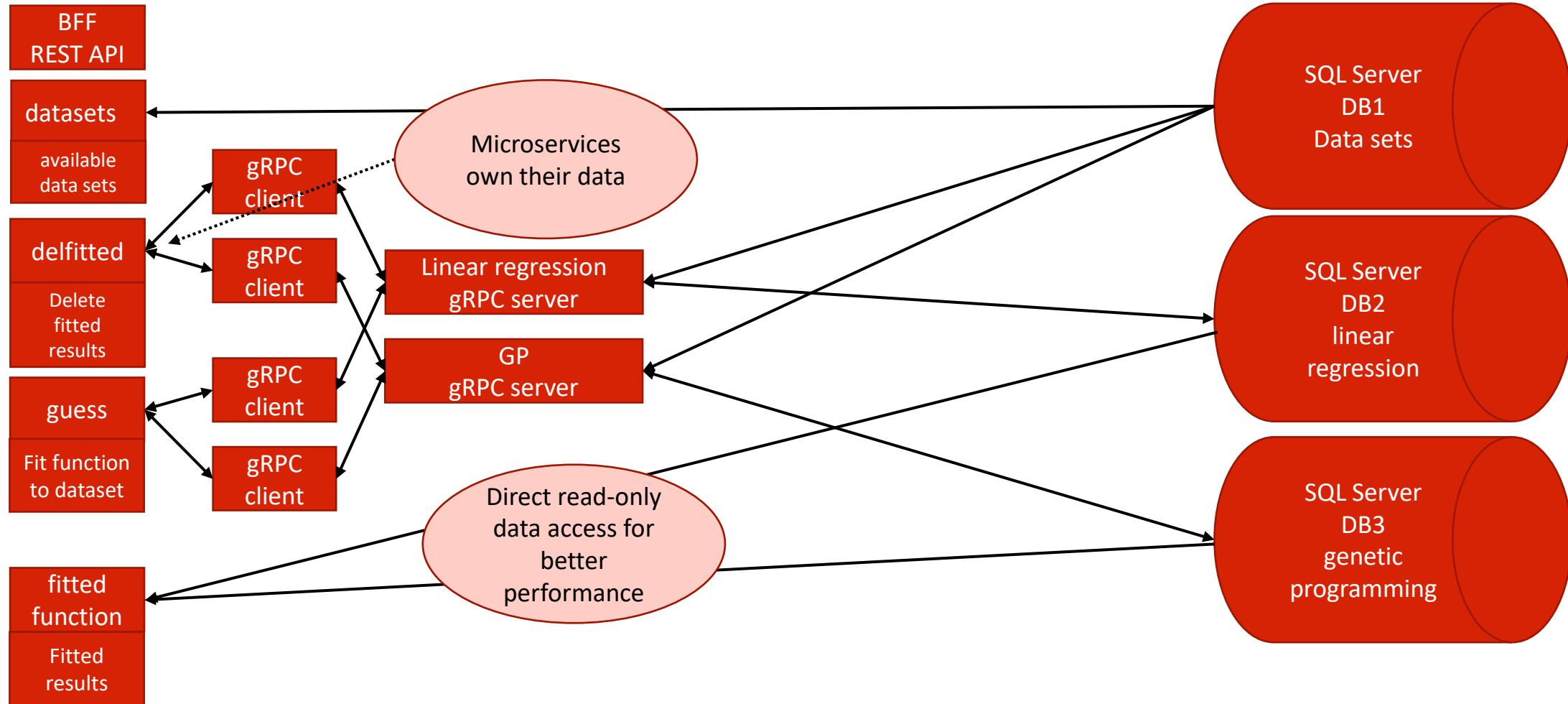
18.11.2021

23



Web-Service Architektur

Übung 26 – BFF und gRPC services



Web-Service Architektur

Übung 26 – protocol buffer



```
// Interface exported by the server
service MathFunc {
    // method fits a function to a given data set
    rpc GuessFunction(FunctionDescription) returns (State) {}
    // method deletes the results (fitted function) for all data sets
    rpc Delete(DelMsg) returns (DelState) {}
}

// class definition for FunctionDescription
message FunctionDescription {
    // field 1, function identifier
    int32 id = 1;
}

message State {
    // field 1
    bool scheduling = 1;
    // field 2
    bool scheduled = 2;
    // field 3
    bool resultready = 3;
}

message DelMsg{
}

message DelState{
}
```

.proto file
service definition

Web-Service Architektur

Übung 26 – Starten der Services



- + Übung 26-web-service

Starten des Services linear programming

- + Wechsel in das Verzeichnis `src\linear_regression`
- + Aktivierung des Virtual Environments
- + Starten des Services mit
`python grpc_linear_regression_server.py`

Starten des Services genetic programming

- + Wechsel in das Verzeichnis `src\genetic_programming`
- + Aktivierung des Virtual Environments
- + Starten des Services mit
`python grpc_gp_server.py`



