

Hochschule Karlsruhe – Data Engineering WS 2021/2022 DSCB330 – Vorlesung 7 – gRPC



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

Serialisierung von Daten



JSON

- + ISO/IEC 21778
- + text-basiert

YAML

- + text-basiert

XML

- + Standardisierung durch W3C
- + Binärformat teilweise unterstützt

SOAP

- + Simple Object Access Protocol
- + Standardisierung durch W3C
- + Binärformat teilweise unterstützt

Pickle (Python)

- + PEP 3154
- + Binärformat

OPC-UA Binary

- + OPC Foundation
- + Binärformat

Protocol Buffers (protobuf)

- + Google
- + Binärformat

Quelle: https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

3



Kommunikation zwischen Programmen (M2M)



- + Bereits behandelt: MQTT, AMQP und OPC UA

REST (Representational State Transfer)

- + Zustandslos, weder Server noch Anwendung sollen Zustandsinformationen speichern
- + Vorteile
 - Infrastruktur im WWW vorhanden
 - Dokumentation der Schnittstelle z.B. über Swagger direkt vom Service abrufbar
 - Lose Kopplung zwischen Client und Server
 - Kommunikation über Uniform Resource Identifier (URI)
- + Nachteile
 - Inhalt der Daten ist nicht eindeutig definiert
 - Dokumentation hängt vom „Fleiß“ des Entwicklers ab
 - Overhead in der Serialisierung (XML, JSON)
 - In der Regel wird HTTP/1.1 genutzt
 - Streaming wird nicht unterstützt

gRPC

- + Remote Procedure Call (RPC)-System
- + Vorteile
 - Starke Typisierung (Klassendefinition, Beschreibung der Schnittstelle mit Protocol Buffers)
 - Konsistenz über Plattformen und Implementierungen hinweg
 - Binäre Serialisierung (effizient)
 - Nutzung des HTTP/2-Standards
 - Übertragung einzelner Daten, Streaming bidirektional möglich
 - Kommunikation über Uniform Resource Identifier (URI) – im Gegensatz zu REST wird der Entwickler nicht mit URIs konfrontiert.
- + Nachteile
 - Im Gegensatz zu REST können Webseiten nicht einfach zum Browser geschickt werden (gRPC-Web notwendig, Nutzung eines Proxies)

Quelle: <https://grpc.io/docs/platforms/web/>



Protocol Buffers (Protobuf)



- + Serialisierung/Deserialisierung strukturierter Daten
- + Übertragung und Speicherung strukturierter Daten
- + Entwicklung von Programmen, die über ein Netzwerk kommunizieren
- + Interface Description Language
 - Beschreibung der Daten (neutrale Klassendefinition)
 - Compiler zur Generierung von Sourcecode für verschiedene Programmiersprachen

Unterstützte Programmiersprachen (Auswahl)

- + Python
- + Julia
- + C#
- + C++
- + Java
- + Go
- + Ruby
- + Objective-C

- + C
- + Perl
- + PHP
- + R
- + Rust
- + Swift
- + Erlang
- + D
- + JavaScript
- + Matlab
- + TypeScript
- + Prolog
- + Fortran

Quelle: https://en.wikipedia.org/wiki/Protocol_Buffers; <https://grpc4bmi.readthedocs.io/en/latest/server/Cpp.html>; https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats



Protocol Buffers (Protobuf)



Beispiel einer proto-Definitionsdatei

+ polyline.proto

```
syntax = "proto2";

message Point {
    required int32 x = 1;
    required int32 y = 2;
    optional string label = 3;
}

message Line {
    required Point start = 1;
    required Point end = 2;
    optional string label = 3;
}

message Polyline {
    repeated Point point = 1;
    optional string label = 2;
}
```

+ Vom Compiler für die Klasse Point erzeugter python-Code (gekürzt)

```
_POINT = _descriptor.Descriptor(
    name='Point',
    ...
    fields=[
        _descriptor.FieldDescriptor(
            name='x', full_name='Point.x', index=0,
            number=1, type=5, cpp_type=1, label=2,
            ...),
        _descriptor.FieldDescriptor(
            name='y', full_name='Point.y', index=1,
            number=2, type=5, cpp_type=1, label=2,
            ...),
        _descriptor.FieldDescriptor(
            name='label', full_name='Point.label', index=2,
            number=3, type=9, cpp_type=9, label=1,
            ...),
    ],
    ...
    serialized_start=18,
    serialized_end=62,
)
```

Quelle: https://en.wikipedia.org/wiki/Protocol_Buffers

- + Entwickelt von Google seit 2015
- + System für Remote Procedure Calls
- + HTTP/2 als Transportmechanismus
- + Protocol Buffers als Interface Description Language
- + Features
 - Authentifizierung
 - Bi-direktionales Streaming
 - Flow control
 - Blocking und Non-blocking Bindings
 - Cancellation
 - Timeouts
 - Compiler unterstützt viele Sprachen

Nutzung

- + Kommunikation zwischen Microservices
- + Verbindung mobiler Geräte zum Backend

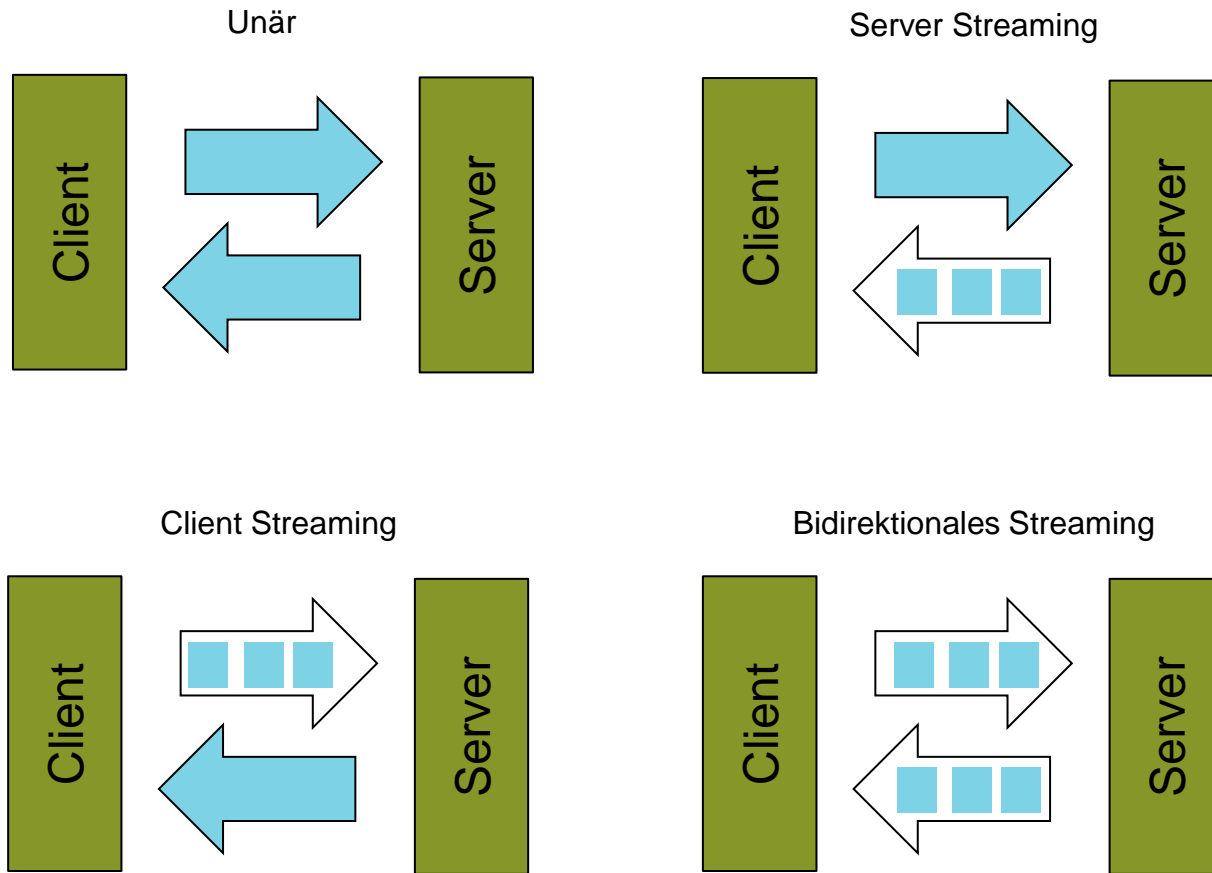
Quelle: <https://en.wikipedia.org/wiki/gRPC>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

+ API Varianten in gRPC auf Basis von HTTP/2

+ Unterstützung synchroner als auch asynchroner RPC-Aufrufe



```
service GreetService {
  // Unary
  rpc Greet(GreetRequest) returns (GreetResponse) {};

  // Streaming Server
  rpc GreetManyTimes(GreetManyTimesRequest) returns (stream GreetManyTimesResponse) {};

  // Streaming Client
  rpc LongGreet(stream LongGreetRequest) returns (LongGreetResponse) {};

  // Bi Directional Streaming
  rpc GreetEveryone(stream GreetEveryoneRequest) returns (stream GreetEveryoneResponse) {};
}
```

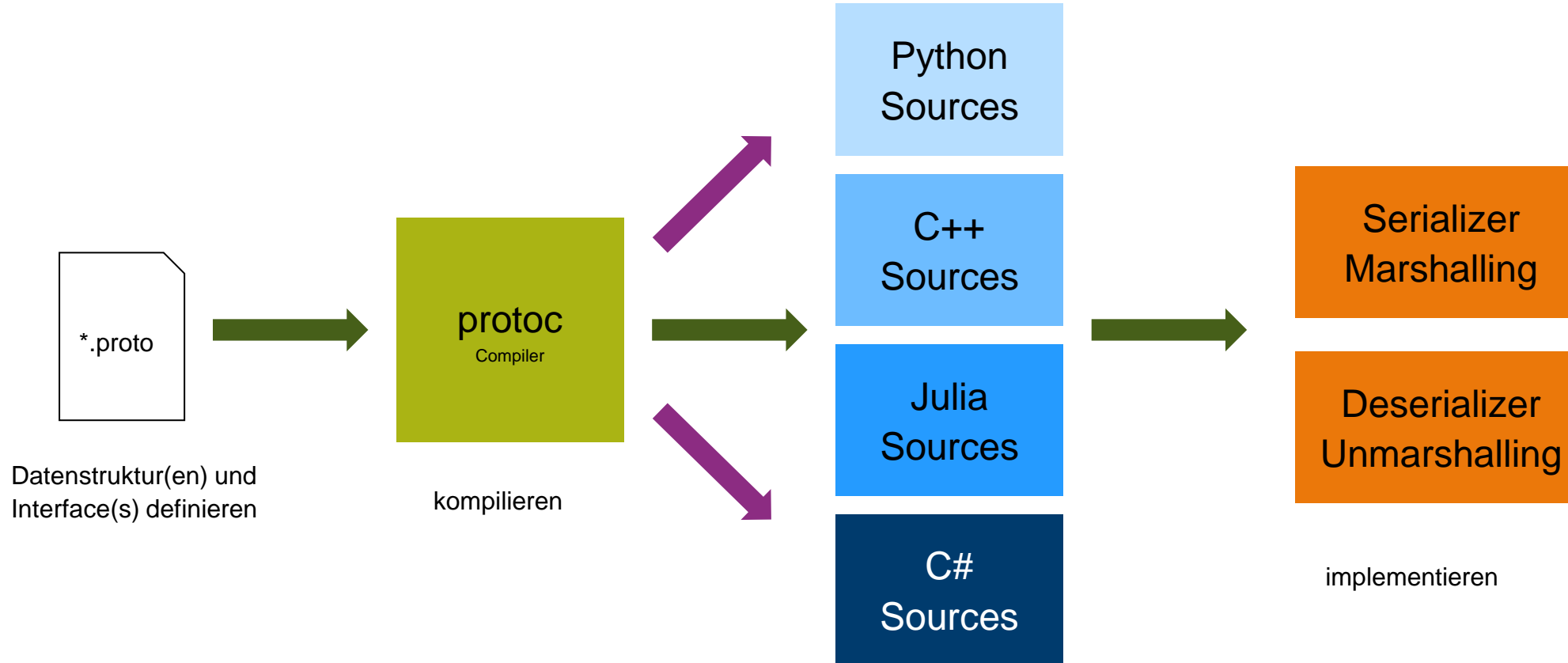
Quelle: <https://en.wikipedia.org/wiki/gRPC>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

8

Protocol Buffers Workflow

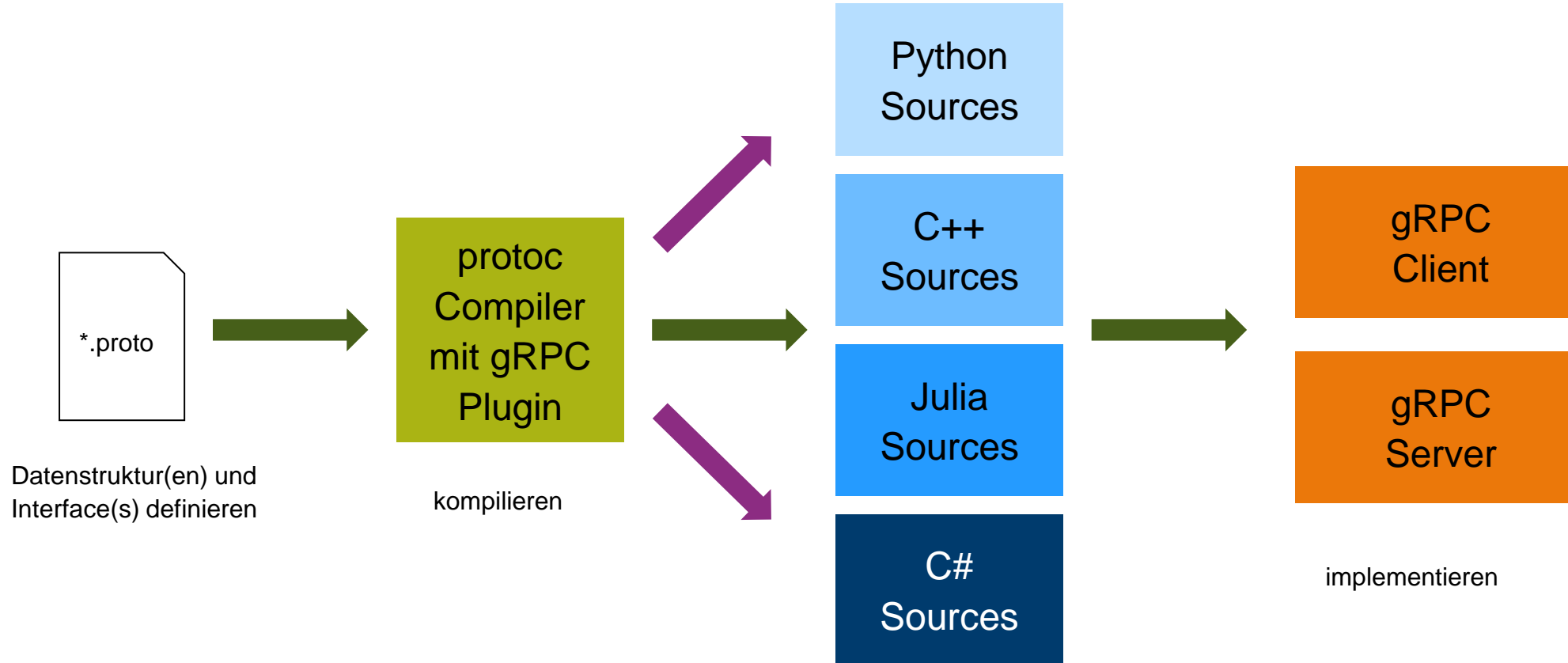


Quelle: <https://en.wikipedia.org/wiki/GRPC>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

gRPC Workflow



Quelle: <https://en.wikipedia.org/wiki/GRPC>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

10

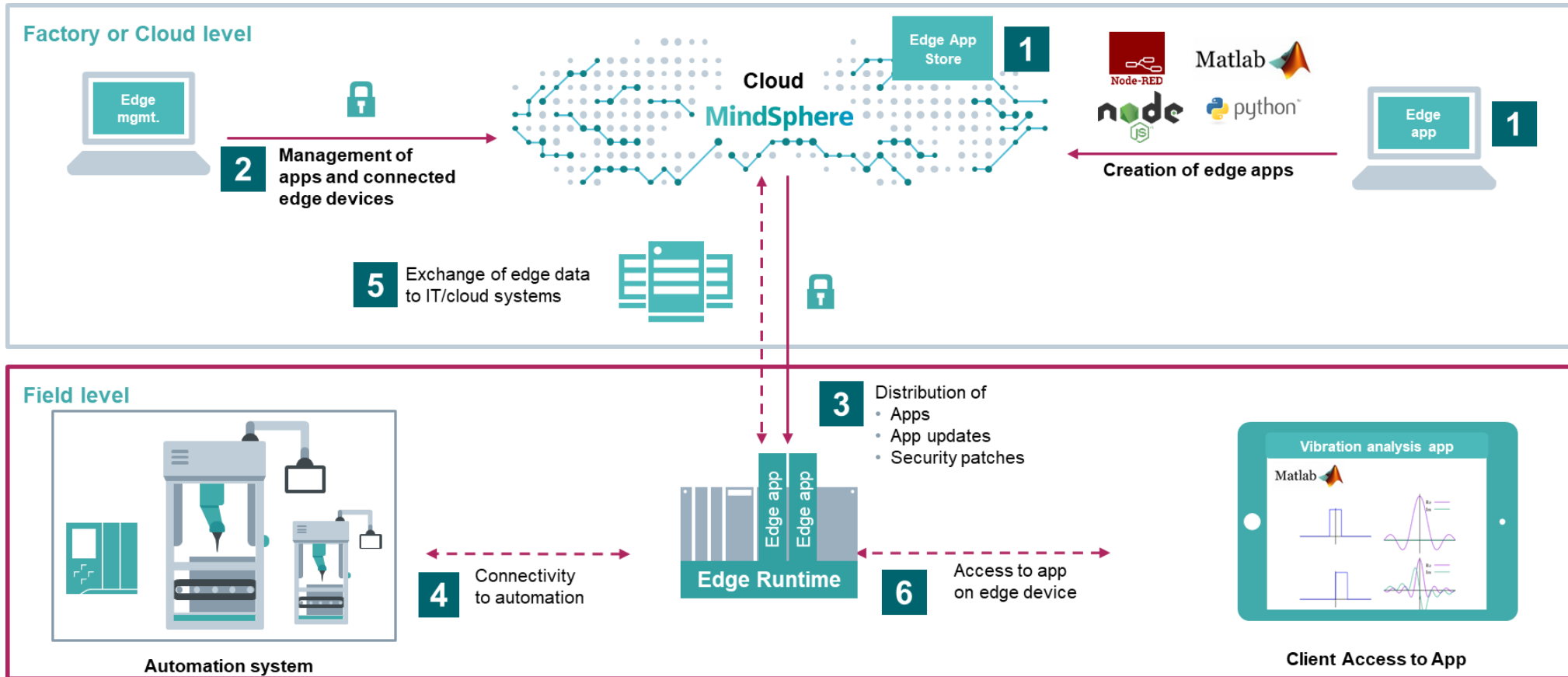


gRPC im IIOT-Umfeld



+ IIOT: Industrial Internet of Things

+ Nutzung von gRPC zur Kommunikation zwischen Edge Apps und Edge devices



Quelle: <https://documentation.mindsphere.io/resources/html/Industrial+Edge+Developer+Environment/en-US/user-docu/industrialedge.html>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

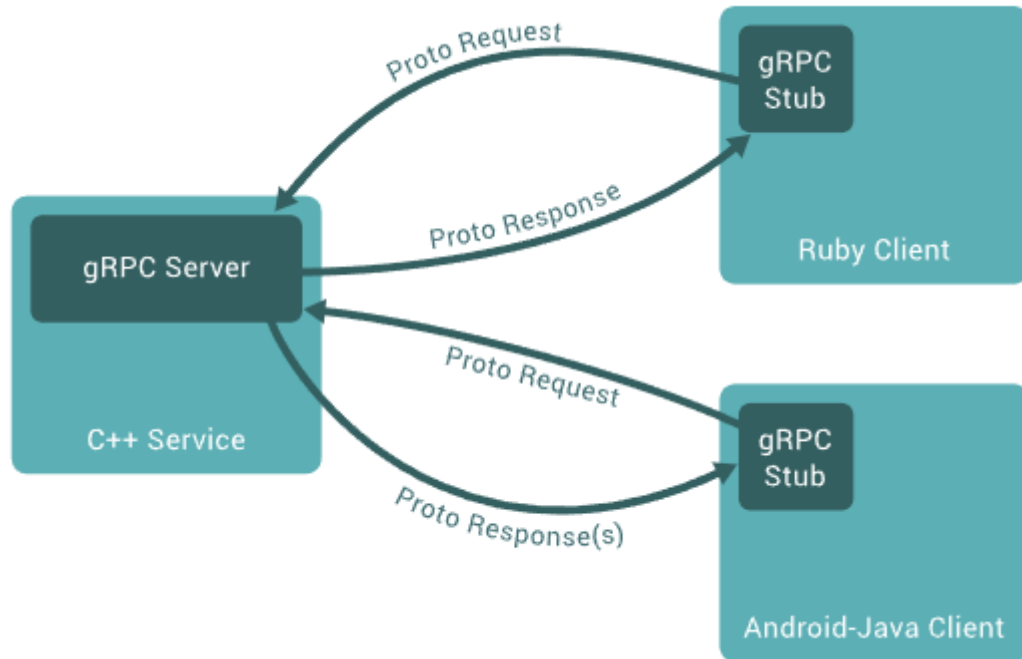
11.11.2021

11

Ausführen von Remote Procedure Calls



- + Mit gRPC kann ein Client eine Methode auf einem Server (auf einem anderen Rechner) wie ein lokales Objekt aufrufen



Quelle: <https://grpc.io/docs/what-is-grpc/introduction/>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

12

Quickstart mit gRPC



- + Nutzung von python 3.5 oder höher
- + Windows und Linux möglich

```
# Erzeugen und Wechsel in ein neues Verzeichnis
mkdir HKAgRPC
cd HKAgRPC

# Installation des Pakets virtualenv
<PFAD>python -m pip install virtualenv

# Erstellen des Virtual Environments im Pfad venv
<PFAD>python -m virtualenv venv

# Aktivieren des Virtual Environments
# mit der Windows Powershell
.\venv\Scripts\activate.ps1

# Aktualisieren des Paket Managers
python -m pip install --upgrade pip

# Installieren von gRPC
python -m pip install grpcio

# Installieren der gRPC tools (compiler)
python -m pip install grpcio-tools
```

- + Herunterladen des Beispielcodes
 - `git clone -b v1.41.1 https://github.com/grpc/grpc`
- + Beispiel helloworld
 - `cd grpc/examples/python/helloworld`

Quelle: <https://grpc.io/docs/languages/python/quickstart/>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 7 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler

11.11.2021

13



Beispiel Studenten an der Hochschule Karlsruhe



+ 24-gRPC-Introduction

+ Proto-File student.proto

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.hka.students";
option java_outer_classname = "StudentsProto";
option objc_class_prefix = "HKAS";
// Interface exported by the server
service Students {
    rpc ListStudent(Name) returns (Student) {}
    rpc ListStudents(Faculty) returns (stream Student){}
}
// class definition for Name
message Name{
    // field 1
    string surname = 1;
    // field 2
    string givenname = 2;
}
```

```
// class definition of Student
message Student{
    // field 1
    Name name = 1;
    // field 2
    Faculty faculty = 2;
    // field 3
    int32 yearOfBirth = 3;
}
```

```
// class definition of Faculty
message Faculty{
    enum FacultyName{
        Unspecified=0;
        ArchitekturBauwesen=1;
        ElektroInformationstechnik=2;
        InformatikWirtschaftsinformatik=3;
        InformationsmanagementMedien=4;
        MaschinenbauMechatronik=5;
        Wirtschaftswissenschaften=6;
    }
    // field 1
    FacultyName facultyname=1;
}
```



- + 24-gRPC-Introduction
- + Wechsel in das Verzeichnis
`src/python`
- + Erstellen der stubs mit dem Befehl
 - `python -m grpc_tools.protoc -I../../protos --python_out=. --grpc_python_out=.`
`..\..\protos\student.proto`
- + Zwei Dateien werden erzeugt:
 - Beschreibung der Klassen
`student_pb2.py`
 - Beschreibung der Services
`student_pb2_grpc.py`



Beispiel Studenten an der Hochschule Karlsruhe



- + 24-gRPC-Introduction
- + Datei `student_server.py`
- + Datei `student_client.py`



Übungsaufgabe 10



gRPC value of index

- + In der Übung „24-gRPC-Introduction“ wird in der Datei `student_client.py` in Zeile 42 als „faculty“ die Zahl 3 bzw. 1 ausgegeben, also
Retrieving students enrolled at faculty 3.
- bzw.
Retrieving students enrolled at faculty 1.
- + Ändern Sie den Sourcecode so ab, dass mit Hilfe der im Proto-File definierten Klasse (`student_pb2.Faculty`) der entsprechende Name der Enumeration ausgegeben wird, so dass die Ausgaben lauten
Retrieving students enrolled at faculty
InformatikWirtschaftsinformatik.
- bzw.
Retrieving students enrolled at faculty
ArchitekturBauwesen.
- + Erstellen Sie eine einseitige Präsentationsfolie, auf der Sie das Ziel, die Vorgehensweise und ihr Ergebnis skizzieren.



Übungsaufgabe 11



gRPC yield students

- + Starten Sie den `student_server.py` aus Übung „24-gRPC-Introduction“.
- Können zwei oder mehr Clients (`student_client.py`) gleichzeitig Verbindung zum Server aufnehmen?
- + Der Server gibt für die Studenten der Fakultät Informatik eine Liste von Studenten mit 27 Einträgen zurück.
- + Verzögern Sie die Ausgabe in der Datei `student_client.py`, indem Sie in Zeile 46 den Ausdruck `time.sleep(0.5)` aktivieren.
- + Starten Sie zwei Clients (nahezu) gleichzeitig.
- + Erhält jeder Client eine halbe Liste der Studenten der Fakultät Informatik oder die vollständige Liste?
- + Erstellen Sie eine einseitige Präsentationsfolie, auf der Sie das Ziel, die Vorgehensweise und ihr Ergebnis für die beiden Fragestellungen skizzieren.



Übungsaufgabe 12



gRPC list lecturers

- + Erweitern Sie die Datei `student.proto` und entsprechend den Sourcecode in `student_server.py` und `student_client.py` um eine Methode `ListLecturers`, die alle Dozenten (z.B. zufällige Namen) der Hochschule Karlsruhe zurückgibt.
- + Erstellen Sie eine einseitige Präsentationsfolie, auf der Sie das Ziel und die Vorgehensweise skizzieren.



