IT-ARCHITEKTURBÜRO
ZT DI Dr. Johannes Weidl-Rektenwald

Staatlich befugter und beeideter
Ingenieurkonsulent für Informatik

Roseggergasse 13, 3412 Kierling

+43 660 572 34 55
office@jwr.at
www.jwr.at

Übungsangabe
184.153 Distributed Systems Engineering
2022S, VU, 2.0h, 3.0EC
Technische Universität Wien

Autor	**Univ.-Lektor ZT Dipl.-Ing. Dr. techn. Johannes Weidl-Rektenwald** Staatlich befugter und beeideter Ziviltechniker für Informatik
Datum	16.02.2022
Version	1.0
Status	Final
Copyright	Diese Beispielangabe darf weder vollständig noch in Auszügen ohne Zustimmung des Autors für einen anderen Anwendungszweck als die Teilnahme und Absolvierung dieser Übung verwendet werden.
Kontaktadressen	jwr@dsg.tuwien.ac.at dse@dsg.tuwien.ac.at

Änderungshistorie

VERSION	DATE	DESCRIPTION	AUTHOR
0.1	23.01.2022	Draft #1	JWR
0.2	13.02.2022	Draft #2	JWR
0.3	15.02.2022	Draft #3	JWR
1.0	16.02.2022	Final	JWR

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Wikipedia definiert „Autonomes Fahren“ wie folgt:

„Unter **autonomem Fahren** (manchmal auch **automatisches Fahren**, **automatisiertes Fahren** oder **pilotiertes Fahren** genannt; eigentlich nur die höchste Stufe nach Level 5) ist die Fortbewegung von Fahrzeugen, mobilen Robotern und fahrerlosen Transportsystemen zu verstehen, die sich weitgehend autonom verhalten.[1] Vom Ursprung des Wortes her ist das Kraftfahrzeug schon immer autonom: Der Begriff *Automobil* besagt, dass sich das Gefährt ohne geschoben oder von Tieren gezogen zu werden *wie* von alleine fortbewegt. Seit in Fahrzeugen Mikroprozessorsysteme, Sensoren und Aktoren zusammenwirken, erfährt der Autonomiebegriff eine Präzisierung: Das Fahrzeug macht nicht den Fahrer autonom, sondern es fährt selbständig.“

[https://de.wikipedia.org/wiki/Autonomes_Fahren, kopiert am 03.12.2017 16:26, Prüfung auf Aktualität am 30.01.2022 13:52]

Für autonom fahrende Fahrzeuge verlangen gesetzliche Vorgaben die Einrichtung eines Unfalldatenspeichers:

Unfalldatenspeicher

§ 5. (1) In den Fällen des § 1 Abs. 1 Z 2 ist jedes Fahrzeug mit einem Unfalldatenspeicher auszurüsten, der während der Testfahrt auch zu verwenden ist.

(2) Mit dem Unfalldatenspeicher dürfen lediglich Daten aus den elektronischen Steuergeräten des Testfahrzeuges aufgezeichnet werden. Diese Daten dürfen nicht veränderbar sein.

(3) Diese Daten dürfen ausschließlich für Testzwecke und der Rekonstruktion von kritischen Situationen oder Unfällen verwendet werden. In Zusammenhang mit Unfällen sind die unfallbezogenen Daten für den Zeitraum von 30 Sekunden vor und nach dem Unfall auf Verlangen den Ermittlungsbehörden und dem Bundesminister für Verkehr, Innovation und Technologie zur Verfügung zu stellen.

[https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20009740, kopiert am 03.12.2017 16:31, Prüfung auf Aktualität am 30.01.2022 13:53]

Autonome Fahrzeuge sind also zwingend mit einem Unfalldatenspeicher auszurüsten, der relevante Fahrzeugdaten Daten für den Zeitraum von 30 Sekunden vor und nach dem Unfall unveränderlich aufzeichnet. Das bedeutet, dass ohne funktionsfähigen Unfalldatenspeicher keine autonome Fahrfunktion aktivierbar sein darf und kein autonomes Fahren stattfinden darf.

Zurzeit gibt es in Österreich und Deutschland noch keinen für autonomes Fahren nach „Level 3“ (oder höher) straßenzugelassenen PKW. Gängige Fahrerassistenzsysteme ermöglichen autonomes Fahren nach „Level 2“ (für die Definition der „Levels“ siehe z.B. [https://cdn.bimmertoday.de/wp-content/uploads/2017/03/BMW-Group-2016-Bilanz-Pressekonferenz-Personal-CoPilot-autonomes-

Fahren-Level-5.jpg]). In Zukunft sollen autonome Fahrfunktionen wie z.B. der „Autobahn-Pilot“ in der Lage sein, Fahrzeuge auf Autobahnen vollständig autonom in allen gängigen Verkehrssituationen zu bewegen. Dabei werden die sogenannte Längs- und Querführung des Fahrzeugs von der politierten Fahrfunktion übernommen, d.h. die pilotierte Funktion lenkt und beschleunigt bzw. bremst selbständig. Bei Systemen nach „Stufe 3“ beispielsweise muss der Fahrer in der Lage sein, das Fahrzeug nach einer kurzen Orientierungsphase wieder übernehmen zu können, kann sich aber während der aktiven Pilotierung anderen Tätigkeiten widmen. Der Fahrer darf aber z.B. nicht schlafen oder den Fahrersitz verlassen, was von einer Vielzahl von Sensoren überprüft wird.

Siehe zum Beispiel: [https://www.auto-motor-und-sport.de/news/ai-staupilot-im-audi-a8-autonom-fahren-im-stau-6086907.html]

Die verschiedenen Stufen des autonomen Fahrens („Levels“) sind z.B. auch hier erklärt: [http://www.kicker.de/news/auto/fahrberichte/startseite/707897/artikel_in-sechs-levels-zum-autonomen-fahren.html]

Durch die immer bessere Ausstattung von Fahrzeugen mit drahtlosen Mobilfunkstandards der neuesten Generation (Stichwort „5G“), schnellerer Übertragungsraten und minimaler Latenzen wird eine Quasi-Echtzeit Kommunikation zwischen den Fahrzeugen (genannt Vehicle-to-Vehicle, kurz „V2V“), aber auch von Fahrzeugen in Backendsysteme in hoher Bandbreite möglich. Bei der Kommunikation von Bordsystemen zu Cloudsystemen – dem oft so genannten „Backend“ – spricht man von Vehicle-to-Infrastucture oder „V2I“ Kommunikation. Die generelle Kommunikation aus dem Fahrzeug zu Fremdsystemen wird „V2X“ oder „Car2X“ genannt. Zum Thema „V2X/Car2X“ wird zurzeit intensiv geforscht und es entstehen laufend neue Business Cases, Produkte und Value Chains.

Siehe z.B. [https://www.businessinsider.de/wirtschaft/mobility/vw-und-microsoft-staerken-kooperation-die-cloud-azure-hilft-auf-dem-weg-zum-autonomen-fahren/]

Daten, die während eines autonomen Fahrvorgangs ohnehin gesetzlich im Fahrzeug aufgezeichnet werden müssen, können also in Quasi-Echtzeit auch in der Cloud gesammelt, ausgewertet, und der Fahrzeugflotte wieder zur Verfügung gestellt werden, um einen Mehrwert zu generieren (siehe z.B. [https://www.telecomitalia.com/tit/it/notiziariotecnico/edizioni-2018/n-3-2018/N10-Avoiding-car-crashes-using-cellular-V2I-communication.html]). Für die Auswertung im Backend können beispielsweise Techniken der Big Data Analyse oder des maschinellen Lernens angewandt werden, um neuartige Services zu erproben und disruptive Innovationen zu ermöglichen. Geltende länderspezifische Datenschutzrichtlinien wie z.B. die DSGVO sind dabei natürlich unbedingt zu befolgen.

Einen möglichen Use Case der V2I Kommunikation im Kontext einer Cloud-basierten Verkehrssteuerung definieren und bearbeiten wir in dieser Übung. Das richtige Know-How und die Kompetenz, die „Time-To-Market“ in der Entwicklung innovativer Services weiter nach unten zu drücken werden darüber

entscheiden, welche Player erfolgreich auf dem riesigen Markt des automatisierten Fahrens und der Car2X Technologien reüssieren werden.

2 Aufgabenstellung

Im Rahmen dieser Übung werden Sie eine hochmodernes Backend für die V2I Kommunikation im Anwendungsberiech des autonomen Fahrens kombiniert mit einer intelligenten Verkehrsflusssteuerung umsetzen.

Die Module des Backend-Systems sollen – dem starken Trend der letzten Jahre folgend - als „Microservices“ ausgeführt werden. Dadurch ergibt sich – im Gegensatz zu einem monolithisch aufgebauten System - ein stark verteiltes System, wodurch grundlegende Probleme verteilter Systeme im Design, in der Umsetzung, im Deployment und im Betrieb gelöst werden müssen.

Die „Orchestrierung“ der Microservices soll über „Kubernetes“ [https://kubernetes.io/] erfolgen, um Herausforderungen wie automatisiertes Deployment, Auto-Skalierung und Fehlertoleranz möglichst automatisiert unterstützen zu können.

Das Benutzer-Interface soll als Web-Applikation ausgelegt werden. Die Datenfeeds aus den Fahrzeugen werden simuliert, durch eine Lösungs-Architektur mit offenen und dokumentierten Schnittstellen soll es aber leicht möglich sein, die Feeds realer Fahrzeuge an das in der Übung erstellte System anzuflanschen.

Der Use-Case der Übung versucht, für autonome fahrende Fahrzeuge eine „grüne Welle“ auf einer Hauptverkehrsstraße mit mehreren Ampelanlagen zu erzeugen. Der autonome Fahrmodus im Fahrzeug kann ein Fahrzeug zwar vollständig autonom durch den „Ampelwald“ chauffieren, die Sensoren im Fahrzeug können aber den Ampelstatus (hier vereinfacht „Rot“ oder „Grün“) nur mit einem sehr eingeschränkten räumlichen (und damit zeitlichen) Vorlauf erkennen. Auch die Positionen und Richtungsvektoren von Fahrzeugen außerhalb der Fahrzeug-Sensorreichweite sind im lokal agierenden autonomen Fahrmodus nicht bekannt. Damit kann lokal im Fahrzeug keine „globale Optimierung“ des Verkehrsflusses erfolgen, d.h. die Geschwindigkeit und Fahrspur so gewählt werden, dass eine „grüne Welle“ entsteht und das Fahrzeug nicht energieintensiv immer wieder abbremsen und beschleunigen muss. In diesem Use-Case interagiert das Backend also mit der autonomen Fahrfunktion im Fahrzeug und stellt Daten zur Verfügung, die in Zusammenspiel mit einer Geschwindigkeits- und ev. auch Fahrspurregelung einen Mehrwert darstellt, indem Energieverbrauch, eventuell auch CO2- und Lärm-Emissionen gesenkt werden und das Fahrerlebnis verbessert wird.

2.1 Funktionale Anforderungen

Die funktionalen Anforderungen stellen sich in einer Use-Case Beschreibung wie folgt dar (Akteure werden unterstrichen dargestellt) [https://de.wikipedia.org/wiki/Anwendungsfall]:

1) Autonome Fahrzeuge unterschiedlicher Hersteller (Fahrzeughersteller werden auch „OEM"s genannt – „Original Equipment Manufacturer") mit jeweils aktiviertem Autopiloten
    a. senden eine Untermenge der Daten, die die Funktion „Datenaufzeichnung für automatisiertes Fahren" (in Folge kurz „DAF" genannt) generiert, ins Backend. Folgende Daten werden periodisch gesendet:
        i. Fahrgestellnummer (Vehicle Identification Number „VIN", deutsch Fahrgestellnummer) – siehe [https://de.wikipedia.org/wiki/Fahrzeug-Identifizierungsnummer]
            1. Die VIN identifiziert also auch eindeutig den Fahrzeughersteller (OEM) und den Modelltyp des Fahrzeugs
        ii. GPS-Lokation
        iii. Geschwindigkeit
        iv. Die Daten werden jeweils mit einem UTC Zeitstempel gesendet

    Wir gehen davon aus, dass die Autopiloten aller am Szenario teilnehmenden Fahrzeuge ständig aktiviert sind.

2) Vernetzte „intelligente" Ampelanlagen senden bei Statuswechsel den aktuellen Ampelfarbenstatus ins Backend:
    a. Farbe: „Grün" oder „Rot" (der Einfachheit halber verzichten wir auf das Senden von „Gelb")
    b. UTC Zeitstempel des Statuswechsels

3) Der Leitstand
    a. kann in einer Web-Applikation Fahrzeuge in einer Kartenansicht darstellen lassen (das Einverständnis der Fahrzeughalter setzen wir hier voraus), die Position wird dabei kontinuierlich aktualisiert. Die Art der Kartenansicht (Google Maps, OpenStreetMap, andere Kartenansicht, schematisch) ist eine freie Designentscheidung in der Umsetzung.
    b. In der Kartenansicht sind die Ampeln eingezeichnet, der aktuelle Ampelstatus (Grün, Rot) und die Scanreichweite pro Ampel ist ablesbar und wird kontinuierlich aktualisiert.
    c. In der Kartenansicht sind alle Fahrzeuge eingezeichnet, die Ist-Geschwindigkeit (v-IST) und Soll-Geschwindigkeit (v-SOLL) wird pro Fahrzeug angezeigt, außerdem die Ampel-ID, falls sich das Fahrzeug in deren Scanreichweite befindet und die VIN.

2.2 Anforderungen zu den Services und Schnittstellen

1. Humanen Benutzer bedienen das System über eine Web-Applikation in einem Browser.
2. Maschinenbenutzer (Fahrzeuge und Ampeln) kommunizieren über den Message Broker bzw. das API Gateway mit dem Backend.
3. Für jedes Microservice ist eine geeignete Schnittstelle zu entwerfen mit Rücksicht auf optimales Design in Bezug auf Coupling und Cohesion bzw. „Bounded Context“ (siehe Vorlesung).
4. Die Microservices sollen jeweils in einem Container laufen, um die Vorteile der Containerisierung zu nutzen.
5. Für jedes Microservice ist eine geeignete Datenhaltung zu entwerfen. Es ist **nicht** erlaubt, dass Microservices direkt (d.h. unter Umgehung des „public API“ des Partner-Microservices) auf die Datenhaltung anderer Microservices zugreifen.

2.3 Anforderungen an die grundlegende Lösungs-Architektur

Die grundlegende Lösungs-Architektur besteht aus einem web-basierten *User Interface („Cockpit“)*, einem *Simulator für Maschinenbenutzer des Backends („Endpoint“)*, einem *API Gateway („Gatekeeper“)*, einem *Message Broker („MoM“)* und einer Menge von *Microservices* mit jeweils eigener *Datenhaltung*.

1) **Endpoint**
   a. Dieses Programm simuliert das Senden von Daten aus Fahrzeugen und von Ampeln an das TrackingService und das Empfangen von Daten vom FlowControlService. Der „Endpoint“ läuft naturgemäß nicht im Backend.
2) **Gatekeeper (API Gateway)**
   a. Aufgabe des Gatekeeper Services ist die Zurverfügungstellung eines geeigneten API für die verschiedenen Clients.
   b. Das API Gateway soll über eine REST oder GraphQL Schnittstelle verfügen. Streaming via GRPC ist ebenfalls zulässig.
3) **MoM (Message Broker)**
   a. Zur asynchronen Kommunikation zwischen den Fahrzeugen und Ampeln (hier simuliert durch das Endpoint Service) mit dem Backend kommt ein Message Broker – genannt „MoM“ - zur Anwendung.
4) **Cockpit**
   a. Dieses Modul implementiert ein web-basiertes User Interface für die Visualisierung/Bedienung unseres Simulationsszenarios.
5) **EntityService**
   a. Dieses Service verwaltet die Daten aller Aktoren, in unserem Fall also Fahrzeuge und Ampelanlagen. Für jedes Fahrzeug sind OEM, Modelltyp und VIN zu speichern. Für jede Ampelanlage ist eine ID und die statische GPS-Lokation des Aufstellungsorts zu speichern.

6) **TrackingService**
   a. Dieses Service speichert alle empfangenen Daten in Bezug auf alle Aktoren bzw. Endpoints.
7) **FlowControlService**
   a. Dieses Service setzt die Verkehrsflusssteuerung um, d.h. es sendet Daten an Fahrzeuge, um die autonomen Fahrfunktion zu unterstützen bzw. zu beeinflussen.
8) **Datenhaltung (DB)**
   a. Die Datenhaltung muss nicht in Kubernetes laufen, Querzugriffe eines Microservices in die Datenhaltung eines anderen Microservices sind jedoch strikt verboten.

1.1.1 Skizzierung der logischen Software-Architektur

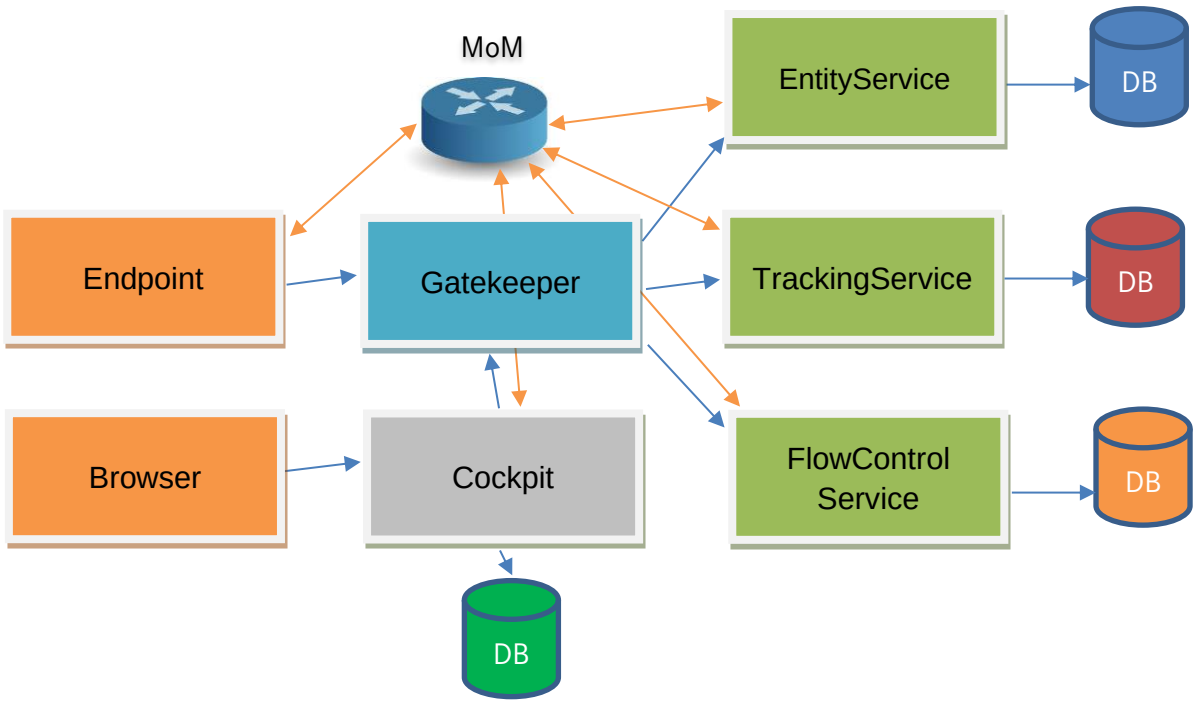

Abbildung 1: Skizze der logischen Software-Architektur

- Jedes Microservice, das Daten permanent speichern muss, muss über seine eigene/separate Datenhaltung verfügen! Kein Microservice darf also auf die „Datenbank“ eines anderen Microservices direkt zugreifen!
- Die Microservices können direkt untereinander kommunizieren, entweder synchron (was aber ein starkes zeitliches Coupling zwischen den Services auslösen würde) oder asynchron über „MoM“.

2.4 Anforderungen zum Deployment

In dieser Übung wird die Google Cloud Platform™ (GCP) benutzt. Nach dem Ende der Gruppenanmeldung wird pro Gruppenmitglied ein 50$ Coupon zur Verfügung gestellt (siehe auch Information auf TUWEL). Google® unterstützt diese Übung dankenswerterweise mit einem kostenlos vergebenen Budget für die Benutzung der GCP.

ACHTUNG: Lösen sie ihr Guthaben unbedingt unmittelbar nach Erhalt des Coupons ein, da es ein Ablaufdatum für die Einlösung gibt. **Nicht eingelöste Coupons verfallen leider komplett, der Einlösezeitraum wird von Google nicht verlängert!**

- Container
  - Jedes Microservice soll in einem eigenen **Container** laufen.

- Orchestrierung
  - Zur Orchestrierung der Container ist die **Google Kubernetes Engine (GKE)** zu verwenden und ein entsprechender Kubernetes Cluster anzulegen.
  - Der Cluster **muss** im Clustermodus „Standard“ erstellt werden, nicht im Modus „Autopilot“.
- Datenbank
  - Da Docker Container im Allgemeinen *stateless* sind - was dem Ziel einer persistenten Datenspeicherung nicht unbedingt entgegenkommt - kann ein Google Datenbank-Service zur Speicherung der Daten verwendet werden. D.h. die Datenhaltung muss nicht innerhalb eines Microservices implementiert werden.
  - Die Microservices in den Containern greifen dann auf abgegrenzte Bereiche in diesem Datenbank-Service zu – wie schon erwähnt, darf es keine Querzugriffe von Microservices auf abgegrenzte Datenbereichen geben, die nicht zum entsprechenden Microservice gehören.
- Messaging (MoM)
  - Der Message Broker (z.B. RabbitMQ, die Auswahl ist aber eine freie Designentscheidung) kann in einem eigenen Container laufen oder es können Google Services (z.B. Cloud Pub/Sub) genutzt werden, um die Message-basierte Kommunikation umzusetzen.

Die Codeabgabe muss komplett deploybar sein im Sinne von IaC (Infrastructure as Code), sprich Sie müssen Ihre Kubernetes Manifeste (YAML Files für die Deployments, Services, etc.) deploybar abgeben.

1.1.2 Lokaler Betrieb von Kubernetes

Für eine – optionale - lokale Entwicklung basierend auf Kubernetes eignen sich die folgenden Frameworks:

1. Minikube (https://github.com/kubernetes/minikube)
2. microk8s (https://microk8s.io/)
3. Docker-for-Mac (https://docs.docker.com/docker-for-mac/kubernetes/)
4. Docker-for-Windows (https://docs.docker.com/docker-for-windows/kubernetes/)

a. Achtung: Der Einsatz von Docker-for-Windows macht Probleme, da die Installation automatisch Hyper-V aktiviert und damit VirtualBox nicht mehr funktioniert. Wenn man also VirtualBox benötigt, ist Docker-for-Windows leider keine Option.

GCP/Kubernetes/StatefulSet Tutorials:

1. https://cloud.google.com/kubernetes-engine/docs/
2. https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app
3. https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook
4. https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk
5. https://cloud.google.com/kubernetes-engine/docs/concepts/statefulset

2.5 Anforderungen zur technischen Umsetzung

Verpflichtend zu verwenden sind:

1. Kubernetes (Google Kubernetes Engine auf der Google Cloud Platform)
2. Eine Container Technologie (z.B. Docker)
3. REST/GraphQL/GRPC Schnittstelle für das API Gateway

Die folgenden Frameworks können, müssen aber nicht verwendet werden:

1. Spring Boot [http://projects.spring.io/spring-boot]
    a. Spring Boot eignet sich besonders für die Implementierung von Microservices.
2. DropWizard [http://dropwizard.io]
    a. DropWizard eignet sich besonders für die Implementierung von Microservices.
3. Quarkus [https://quarkus.io/]
4. Circuit Breaker Hystrix [https://github.com/Netflix/Hystrix]
5. Mongo DB mit Spring Data MongoDB [http://projects.spring.io/spring-data-mongodb/]
    a. Spring Data MongoDB bietet eine Java Repository Abstraktion zum Zugriff auf MongoDB Funktionen.
6. Service Mesh Istio [https://istio.io]
7. Service Mesh linkerd [http://linkerd.io]
8. resilience4j [https://github.com/resilience4j/resilience4j]

Weitere Tutorials und Quellen:

1. Docker Tutorial
    a. https://www.docker.com/tryit/
2. Hystrix Tutorial
    a. https://github.com/Netflix/Hystrix/wiki/How-To-Use
3. Spring Boot Tutorial
    a. https://spring.io/guides/gs/spring-boot/

4. MongoDB Geospatial Queries:
    a. https://docs.mongodb.com/manual/geospatial-queries/
5. Building Microservices with Spring Boot
    a. http://www.infoq.com/articles/boot-microservices
    b. https://blog.scottlogic.com/2019/10/31/building-microservices-with-spring-boot.html
6. GKE – Google Kubernetes Engine
    a. https://cloud.google.com/kubernetes-engine/docs/
7. Docker Get Started
    a. https://docs.docker.com/get-started/
8. Docker Overview
    a. https://docs.docker.com/engine/docker-overview/
9. Kubernetes „Hello World“ Example
    a. https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app
10. MongoDB on GCP
    a. https://codelabs.developers.google.com/codelabs/cloud-mongodb-statefulset/
    b. https://cloud.google.com/solutions/deploy-mongodb?hl=de
11. k8s – OpenShift - GKE comparison
    a. https://www.slideshare.net/deview/d2-community-open-container-seoul-meetup-kubernetes-openshift
12. OpenShift / Kubernetes difference
    a. https://www.redhat.com/de/blog/openshift-and-kubernetes-whats-difference

2.6 Anforderungen an das Simulationsszenario

Erstellen sie einen Simulator, um den folgenden Ablauf zu demonstrieren (siehe auch Abbildung 2):

1. Eine lange gerade Straße weist in regelmäßigen Abständen Kreuzungen auf, die jeweils durch eine Ampelanlage geregelt werden.
2. Die Ampelanlagen schalten zu gewissen vordefinierten Zeiten auf von „Grün“ auf „Rot“ und nach einer gewissen Zeit wieder zurück auf „Grün“.
3. Das Backend überprüft aufgrund der von den Fahrzeugen übermittelten Geodaten periodisch, ob sich ein (in unserem Fall autonom gesteuertes) Fahrzeug einer Ampel nähert. Die Scan-Reichweite in Richtung des sich nähernden Fahrzeugs ist pro Ampel konfiguriert und in der Regel unterschiedlich.
    a. Anmerkung: MongoDB zum Beispiel unterstützt geospatiale Abfragen, die genau für diesen Zweck herangezogen werden können.
4. Falls ein sich näherndes Fahrzeug erkannt wird, wird aufgrund der übermittelten Ist-Geschwindigkeit dem Autopiloten des Fahrzeugs genau jene Soll-Geschwindigkeit übermittelt, bei deren Einhaltung die nächste Ampel bei Erreichen Grün zeigen wird.

   a. Der Einfachheit halber kann man davon ausgehen, dass das Fahrzeug die Ist-Geschwindigkeit nach Erhalten der Soll-Geschwindigkeit instantan - also sofort - auf die Soll-Geschwindigkeit ändert. Weiters gehen wir vereinfachend davon aus, dass ein Fahrzeug beliebig oft pro Zeiteinheit die Geschwindigkeit wechseln kann (auch wenn die Insassen das in der Praxis wohl nicht als angenehmes Fahrerlebnis interpretieren würden).
   b. *Optional:* Simulation einer realistischen Anpassung der Ist-Geschwindigkeit nach Erhalt der Soll-Geschwindigkeit
   c. *Optional:* Simulieren sie einen geordneten Brems- und Beschleunigungsvorgangs des Fahrzeugs im Falle, dass das Fahrzeug vor dem Ampel halten muss.
5. In dem in Abbildung 2 beschriebenen Szenario nähert sich ein Fahrzeug von Süden der Ampel-Kaskade.
6. Bei Identifikation des Fahrzeugs vor Ampel a1 wird dem Autopiloten vom Backend eine Soll-Geschwindigkeit vs1 übermittelt, bei deren Einhaltung die Ampel a1 bei Erreichen Grün zeigen wird.
7. Nach Erreichen der Ampel a1 wird dem Autopiloten bei Erreichen der Scanreichweite von Ampel a2 vom Backend eine Soll-Geschwindigkeit übermittelt, bei deren Einhaltung die Ampel a2 bei Erreichen Grün zeigen wird und so weiter.
8. Zu Beginn des Szenarios können die Parameter a) über das Cockpit manuell ausgewählt werden bzw. b) eine Zufallsauswahl vom Cockpit selbst generiert werden (z.B. über einen Button „Parameter zufällig wählen“).
   a. Beispiel für eine Auswahl:
      i. 4 Ampeln [Wertebereich: 1-4]
         1. Abstand der Ampeln beginnend von gps0 von Süden nach Norden
         2. Scanreichweite pro Ampel (die Mindestscanweite muss definiert und dem System bekannt sein, die Scanreichweite darf bis maximal zur nächsten Ampel reichen!)
         3. Rot bei t0+x Sekunden für y Sekunden pro Ampel
      ii. 2 Fahrzeuge [Wertebereich: 1-4]
         1. gps0 kann immer konstant gehalten werden
         2. t0 (Eintreffzeitpunkt) pro Fahrzeug
         3. vi1 (Ist-Geschwindigkeit bei Eintreffen) pro Fahrzeug (die erlaubte Mindest- und Maximalgeschwindigkeiten – global, nicht pro Fahrzeug – müssen definiert und dem System bekannt sein. Falls die definierten Grenz-Geschwindigkeiten über- bzw. untertroffen werden, muss das Fahrzeug an der Ampel bis zur nächsten Grünphase halten.)
         4. gps1 darf nicht vor Ende der Scanreichweite der letzten Ampel liegen
9. Mindestanforderungen
   a. Wie oben ersichtlich müssen Szenarios bis zu 4 Ampeln und 4 Fahrzeuge unterstützt werden.
10. Der Verkehrssteuer-Algorithmus muss Fahrzeuge zeigen, die sich aus beiden Richtungen den Ampeln nähern – also von Süd nach Nord und von Nord nach Süd. Bei mehr als einem Fahrzeug muss sowohl die Nord- als auch die Südrichtung im Szenario dargestellt werden.

11. Zeigen Sie das beschriebene Szenario in Echtzeit und – falls diese Funktion aktiviert ist - in Zeitraffer (z.B. Faktor 10, der Faktor muss nicht konfigurierbar sein) auf einer Kartendarstellung unter der Annahme, dass der Autopilot die übermittelten Soll-Geschwindigkeiten erhält, akzeptiert und die Ist-Geschwindigkeit dementsprechend anpasst.

Anmerkung: Zur besseren Simulation des Szenarios auf einer kontrollierbaren Zeitachse kann die vom Autor beschriebene Flux-Kompensator Technologie zum Einsatz kommen – siehe http://coopxarch.blogspot.co.at/2011/05/der-fluxkompensator-und-seine-bedeutung.html

Anmerkung: In der Zukunft werden komplexe, skalierbare und fehlerrobuste Algorithmen benötigt, die den Verkehrsfluss tausenden von Verkehrsteilnehmern in komplexen Verkehrsszenarien in Quasi-Echtzeit steuern und optimieren.

Optional: Zeigen Sie, dass das Stoppen eines Cluster Nodes eines Microservices (Compute Engine VM-Instanz) keine Auswirkung auf die Verfügbarkeit des Gesamtsystems hat, d.h., dass das Demo-Szenario unbeeindruckt vom Ausfall durchläuft.

Optional: Installieren Sie das Service Mesh ‚linkerd' und zeigen Sie, wie man von linkerd gesammelte Daten bezüglich der Microservice Kommunikation darstellen kann.

Optional: Implementierung von Autorisierung und Authentifizierung.

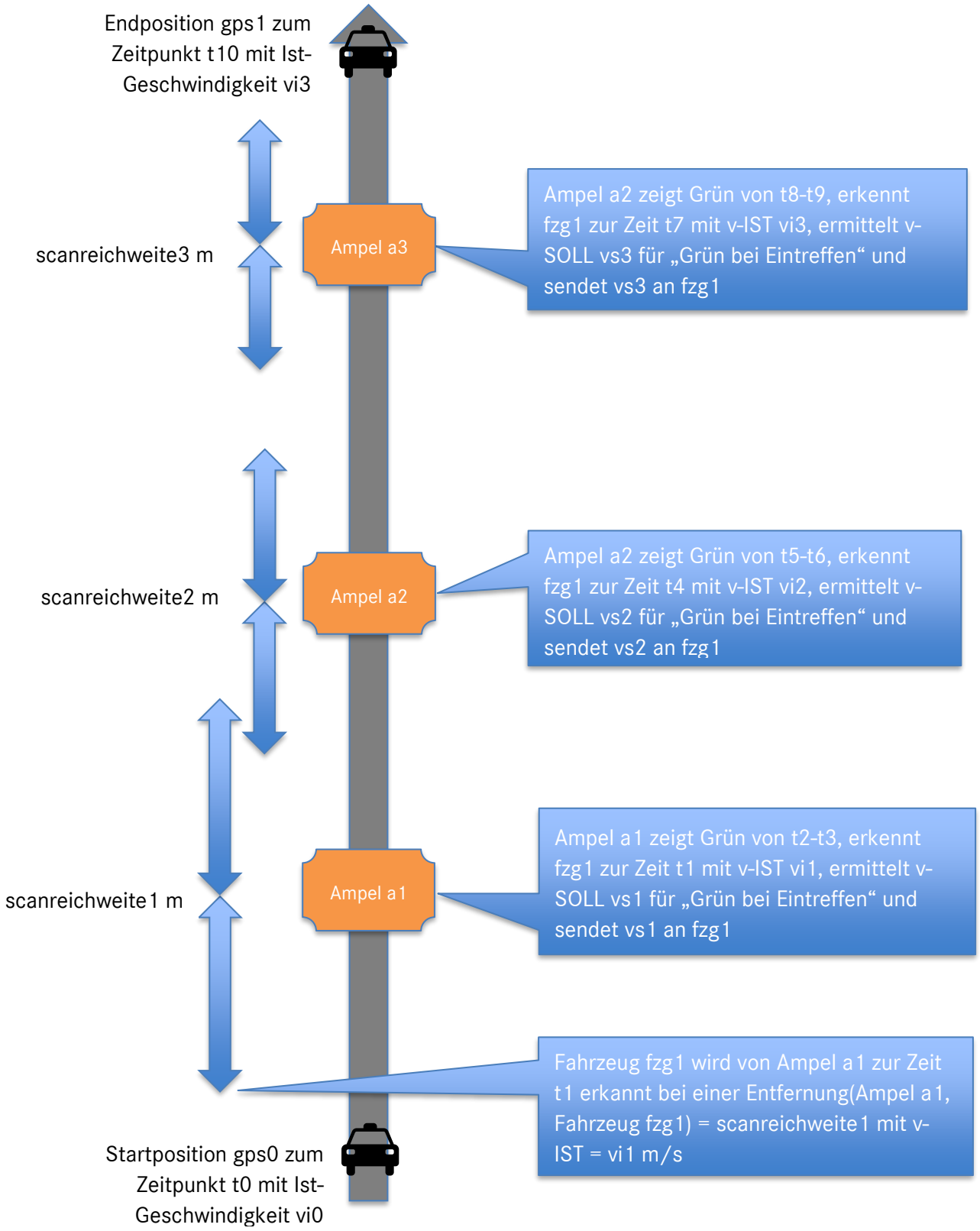

Abbildung 2: Ein ausgewähltes Simulationsszenario mit einem Fahrzeug von Süd nach Nord und drei Ampeln

2.7 Anforderungen zum Abgabegespräch

Im Abgabegespräch werden Sie den Aufbau und die Funktion Ihrer Lösung anhand der Live-Durchführung des in Kapitel 2.6 spezifizierten Simulationsszenarios in der Google Kubernetes Engine präsentieren. Weitere Details zu den Abgabegesprächen werden in Kapitel 5 dargelegt.

2.8 Anforderungen zur Codeabgabe

Die Codeabgabe muss komplett deploybar sein im Sinne von IaC (Infrastructure as Code), sprich Sie müssen Ihre Kubernetes Manifeste (YAML Files für die Deployments, Services, etc.) deploybar abgeben.

2.9 Anforderungen zu Tests

Für jedes Microservice sind mindestens zwei (2) Modul-Tests zu schreiben, die die Funktionalität über das Microservice API testen. Die Verwendung eines Test-Frameworks wird freigestellt.

3 Lieferinhalte (Deliverables)

Die gelieferten Dokumente können entweder in Deutsch oder Englisch verfasst werden. Source Code Dokumentation und API-Dokumentation müssen in Englisch verfasst werden. Alle Deliverables müssen erstellt und abgegeben werden - es gibt keine optionalen Deliverables.

3.1 Projektplan

Erstellen Sie ein Dokument mit folgendem Inhalt:

Deliverable 1: Dokument: Projektplan

1. Projektzieleplan: Enthält eine kurze Projektbeschreibung mit den Projektzielen und Nicht-Zielen
2. Projektorganigramm: Enthält die Rollenvergabe im Projektteam
    a. Wer übernimmt welche Rolle bzw. Aufgaben? Wer hat welche Verantwortlichkeit?
3. Work Breakdown Stucture (WBS): Die WBS enthält die Arbeitspaketspezifikationen
    a. Was sind die einzelnen Aufgaben, die innerhalb der Übung zu lösen sind?
4. Aufwandsschätzung: Welchen Aufwand planen Sie für die Tätigkeiten der WBS ein?
    a. Wie hoch ist der geschätzte Aufwand für jedes Projektteammitglied (in Personenstunden)?
    b. Wie hoch ist der geschätzte Gesamtaufwand (in Personenstunden)?

5. Ressourcenplanung und Meilensteinplanung
    a. Wer im Team erledigt welches Arbeitspaket bis wann?
    b. Festlegung der Fertigstellungstermine für die Arbeitspakete und Deliverables
6. Technische Planung: Welche Methoden, Technologien und Werkzeuge werden Sie verwenden und wozu?
7. GCP Budget Schätzung: Schätzen sie, wie viel GCP Budget sie für die Absolvierung der Übung in der Google Cloud Platform benötigen werden. Evaluieren sie das GCP Kostenmodell – eventuell gibt es ja Services, die bis zu einer gewissen Nutzung gratis angeboten werden.
8. *Optional:* Projektbalkenplan (klassisch) oder Sprintplanung (agil)

Beachten Sie die **zeitgerechte** Abgabe des Projektplans!

3.2 Architektur- und Designdokument

Erstellen Sie ein Dokument mit dem folgenden Inhalt:

Deliverable 2: Dokument: Architektur- und Designdokument

1. Beschreibung der Software-Architektur
    a. Welche Komponenten / Microservices existieren?
    b. Welche Schnittstellen existieren?
    c. Wie kommunizieren die Services miteinander?
2. Stellen Sie den „Logical View“ und den „Physical bzw. Deployment View“ wie im „4+1 Architectural Model“ von Philippe Kruchten dar - siehe dazu auch [http://en.wikipedia.org/wiki/4%2B1_Architectural_View_Model]

3.3 Implementierung

Erstellen Sie das in Übungsangabe beschriebene verteilte System.

Deliverable 3: Codeabgabe: Implementierung der angeforderten Module

3.4 Wartungshandbuch

Erstellen Sie ein Dokument mit dem folgenden Inhalt:

Deliverable 4: Dokument: Wartungshandbuch

1. Beschreibung der Entwicklungsumgebung
2. Beschreibung der eingesetzten Frameworks und Libraries
3. Beschreibung des Build Prozesses

4. Beschreibung des Testprozesses
5. Beschreibung des Deployments
6. Dokumentation des API Gateway APIs mittels Swagger (https://swagger.io) oder einem vergleichbaren API-Dokumentations-Framework

3.5 Projektabschlussdokumentation

Erstellen Sie einen Projektabschlussbericht.

Deliverable 5: Lessons Learned und Conclusio

1. Beurteilen Sie den Ablauf des Projektes
    a. Was hat gut funktioniert, was weniger gut? Begründen Sie ihre Beurteilung.
    b. Wenn zutreffend, warum haben Sie den ursprünglich geschätzten Aufwand überschritten?
    c. Was würden Sie aufgrund der gewonnenen Erfahrung anders bzw. besser machen?
2. Beurteilen Sie die finale Software
    a. Was ist gut geglückt, welche Teile würden Sie beim nächsten Mal anders/besser lösen.
3. Verfassen Sie ihr persönliches Fazit zur Lehrveranstaltung.

4 Bewertung der Lieferinhalte

Deliverable 1: Dokument: Projektplan

5 Punkte – Abzüge bei Unvollständigkeit bzw. fehlender Plausibilität

- **0,5 Punkte** - Enthält eine kurze Projektbeschreibung mit den Projektzielen und Nicht-Zielen (Projektzieleplan)
- **0,5 Punkte** - Enthält die Rollenvergabe im Projektteam (Projektorganigramm)
    - Wer übernimmt welche Rolle bzw. Aufgaben? Wer hat welche Verantwortlichkeit?
- **1 Punkt** - Enthält die Arbeitspaketspezifikation oder Work Breakdown Structure (WBS)
- **1 Punkt** – Enthält die Aufwandsschätzung
- **1 Punkt** - Enthält die Ressourcenplanung und Meilensteinplanung
    - Wer im Team macht was und bis wann?
    - Festlegung der Fertigstellungstermine für die Arbeitspakete und Deliverables
- **1 Punkt** – Enthält die technische Planung und die GCP Budgetschätzung
    - Welche Methoden und Tools werden Sie verwenden und wozu?
    - Die Technische Planung ist detailliert ausgeführt.
    - Wie viel GCP Budget planen sie zu verbrauchen?

Deliverable 2: Dokument: Architektur- und Designdokument

5 Punkte – Abzüge bei Unvollständigkeit bzw. nicht nachvollziehbaren Designentscheidungen, die im Gegensatz zu dem in der Vorlesung vermittelten state-of-the-art stehen.

Deliverable 3: Codeabgabe: Implementierung der angeforderten Module

45 Punkte – Abzüge bei Unvollständigkeit: fehlende API Dokumentation, fehlende Inline Kommentare, fehlende Funktionalität, fehlende Kommunikation, fehlende Unabhängigkeit der Komponenten, nicht funktionierendes User Interface, unerlaubte Trivialisierung, fehlende Tests, fehlende Verwendung vorgeschriebener Frameworks.

Deliverable 4: Dokument: Wartungshandbuch

3 Punkte – Abzüge bei Unvollständigkeit gegenüber der Aufgabenstellung.

Deliverable 5: Dokument: Conlcusio und Lessons Learned

2 Punkte – Abzüge bei Unvollständigkeit gegenüber der Aufgabenstellung.

Höchstpunktezahl: **60 Punkte**
Mindestpunktezahl**: 30 Punkte**

Anmerkung: Für die Umsetzung von optionalen Anforderungen (gekennzeichnet mit dem Wort „Optional:“) werden keine Punkte vergeben. Es können damit auch keine Punkte für Optionen gegen nicht erreichte Punkte verpflichtender Teile „eingetauscht“ werden.

5 Übungsablauf und -organisation

- Über die Gruppenanmeldung im TUWEL werden Übungsgruppen zu je drei (3) Personen gebildet.
- Laden Sie den Projektplan (Deliverable 1) **bis spätestens 10.04.2022, 23:55** im TUWEL hoch.
    - **Alle** Dokumente müssen in einem der folgenden Formate abgegeben werden: .rtf, .doc, .pdf oder .html! Im speziellen ist für den Projektplan kein Microsoft Project® oder anderes Projektmanagementtool-spezifisches Format erlaubt! Achten Sie darauf, dass keine Links auf externe Objekte (Bilder, Diagramme) in den Dokumenten vorhanden sind. Achten Sie darauf, dass die Dokumente nicht exorbitant groß sind (z.B. durch Einbinden sehr großer Bilder).
    - **Alle** abgegebenen Dokumente müssen als Dateinamen bzw. Dokumenttitel die Nummer des Deliverables und den Dokumenttitel (wie in der Aufgabenstellung spezifiziert) tragen! Der Titel muss auf der ersten Seite sowie über Fußzeilen im ganzen Dokument ersichtlich sein.
        - Der Dateiname für die Projektplanabgabe als pdf ist also **„DSE22Gruppe<nn>-Projektplan.pdf“**
        - <nn> entspricht ihrer Gruppennummer formatiert auf zwei Stellen
    - Falls Sie z.B. HTML als Format wählen und das Dokument mehrere HTML Dateien umfasst, packen Sie die Dateien in ein ZIP Archiv. Achten Sie darauf, dass das Dokument direkt aus dem ZIP Archiv durchgebrowst werden kann, d.h. das ZIP Archiv nicht auf die Festplatte entpackt werden muss.
        - Der Dateiname für die Projektplanabgabe als zip Archiv ist also **„DSE22Gruppe<nn>Projektplan.zip“**
- Buchen Sie pro Gruppe im TUWEL einen Termin für Ihr Abgabegespräch **bis spätestens 29.05.2022, 23:55**.
    - Die Abgabegespräche finden vom **20.06.2022** bis **24.06.2022** statt.
- Laden Sie die Gesamtabgabe aller Deliverables im TUWEL hoch **bis spätestens 17.06.2022, 23:55**.
    - Alle Files der Abgabe müssen in einer sinnvollen Verzeichnisstruktur abgelegt werden, die der Nummerierung der Deliverables entspricht
        - Legen Sie der Abgabe unbedingt auch wieder Deliverable 1 (Projektplan) bei.
    - Der Dateiname der Abgabe muss **DSE22Gruppe<nn>-Abgabe.zip** lauten
    - Der Dateiname der Projektplan Datei in der Gesamtabgabe ist also „**1_Projektplan.pdf**“ oder **„1_Projektplan.zip“**
- Ablauf des Abgabegesprächs
    - Zur Präsentation innerhalb des Abgabegesprächs ist die Anwesenheit **aller** Gruppenmitglieder erforderlich.
    - Sie präsentieren Ihre Lösung mittels
        - Kurzüberblick über die Lösung
        - Deployment der Lösung in die GCP im Sinne von IaC (Infrastructure as Code) mittels Kubernetes Manifeste (YAML Files)
            - Das System muss für die Präsentation auf GCP deployed sein (keine lokalen Installationen)!
        - Live Demo des Simulations-Szenarios

        - Einblick in das laufende System (Logging, Google Cloud Console, Error handling, etc.)
        - Code Walkthrough
        - Build Durchlauf mit erfolgreicher Ausführung aller Tests
    - Jedes Gruppenmitglied muss die von ihm erarbeiteten Teile präsentieren
    - Jedes Gruppenmitglied muss das Gesamtprojekt - d.h. auch die Teile, die die anderen Gruppenmitglieder präsentieren – kennen, vollständig verstehen und auch Fragen dazu beantworten können.
    - Wir behalten uns vor, Gruppenmitgliedern weniger Punkte zu vergeben, wenn der Eindruck entsteht, dass diese nicht adäquat am Gesamtumfang der Übung mitgearbeitet haben.

5.1 Wichtige Anmerkungen und Tipps

- **Studieren Sie die Angabe genau!** Es werden immer wieder Deliverables vergessen bzw. Themen verfehlt. Um die Übung positiv zu absolvieren, müssen alle Deliverables abgegeben werden. Wir behalten uns vor, formal nicht entsprechende Deliverables nicht zu akzeptieren.
- Verwenden sie **keine öffentlich zugänglichen** Groupware Accounts oder öffentliche Source Code Repositories! Andere Gruppen könnten Zugriff auf Ihren Source Code bzw. Ihre Dokumente erlangen!
    - Verwenden Sie z.B. Bitbucket (https://bitbucket.org/) – hier können sie kostenlose private Arbeitsbereiche einrichten.
    - GitHub bietet mittlerweile ebenfalls private Repositories für max. 3 Collaborators (https://github.com/pricing) an.
    - Gitlab (www.gitlab.com) ist eine weitere Möglichkeit.
- Teilen Sie Rollen bzw. Zuständigkeiten innerhalb der Gruppe ein (für z.B. Projektleitung, Koordination, Technik, Tools, Dokumentation). Planen Sie Projektmeetings auch während der Laufzeit, z.B. zu bestimmten Milestones. Heben Sie nicht alles für den Schluss auf.
- Bei der Präsentation muss man über das Gesamtprojekt informiert sein – nicht nur über seinen eigenen Teil!
- Wenn Sie bei der Lösung aufgrund von unvollständigen Angaben bzw. Informationen *Annahmen* treffen, dokumentieren Sie diese unbedingt! Annahmen dürfen in keinem Fall die Lösung trivialisieren.
- Plagiate: Bitte kopieren Sie keine Lösungen anderer Gruppen oder Lösungen, die im Internet angeboten werden. Wenn wir Plagiate entdecken, vergeben wir 0 Punkte und Sie können die Übung nicht positiv absolvieren. Natürlich ist es erlaubt, Problemstellungen und mögliche Lösungen mit Ihren Kollegen zu diskutieren, aber die abgegebene Lösung muss zu 100% von Ihnen erstellt worden und einzigartig („unique“) sein. Auch wenn Gruppen gemeinsam arbeiten, dürfen die Lösungen zueinander nicht zu ähnlich sein.
- Deadline Verschiebungen: Deadline Verschiebungen werden nur in speziellen Situationen zugestanden, nachdem eine individuelle Vereinbarung mit der Kursleitung getroffen wurde. Wenn Sie Ihr Beispiel nicht rechtzeitig beenden können, geben sie Ihr Beispiel unvollendet ab – das ist immer noch besser, als gar nichts abzugeben. Achten Sie darauf, dass bei der Abgabe immer etwas schiefgehen kann und planen Sie das entsprechend ein. Geben Sie ihre Lösung

rechtzeitig ab und beachten Sie, dass kurz vor der Deadline die Server überlastet sein können. Wir können keine E-Mail-Abgaben oder andere Uploads akzeptieren. Wenn Sie Probleme mit der Abgabe haben, senden sie einen SHA-512 Hash Ihrer Abgabe an die Kursleitung rechtzeitig vor der Abgabe-Deadline. Mittels dieses Hashes können wir überprüfen, ob die Abgabe nach der Deadline modifiziert worden ist (nachdem Sie die Abgabe nach der Deadline hochgeladen haben, sobald das Upload Problem behoben ist).

5.2 Wichtige Termine

So, 10.04.2022, 23:55: Deadline Abgabe Projektplan via TUWEL
So, 29.05.2022, 23:55: Deadline Buchung Abgabegesprächstermin
Fr, 17.06.2022, 23:55: Deadline Gesamtabgabe aller Deliverables via TUWEL
Mo 20.06.2022 bis **Fr, 24.06.2022:** Abgabegespräche

6 Zusammenfassung der Übungsinhalte

1. Projektplanung
2. Architektur- und Designentwurf eines verteilten Systems im Kontext von Microservices, Containern und Orchestrierungsmechanismen
3. Implementierung
4. Test
5. Deployment
6. Erstellung eines Wartungshandbuchs
7. Projektarbeit: Gruppenkommunikation- und Zusammenarbeit
8. Präsentationstechnik