Praktikum Verteilte Systeme

Wintersemester 2020/2021

Prof. Dr. Michael von Rüden

Übersicht

Im Rahmen des Praktikums *Verteilte Systeme* soll eine Anwendung aus dem Bereich „Connected Cars“ entwickelt werden. Dazu sollen die Technologien *Sockets*, *RPC* (Apache Thrift*)* sowie *Message-Oriented-Middleware* (MQTT) verwendet werden.

Rahmenbedienungen

- Das Bearbeiten der Praktikumsaufgaben findet in Zweier-Teams statt.
- **Die Aufgaben werden im Rahmen der Meilensteine oder individuell testiert**. Alle Aufgaben müssen spätestens zum letzten Meilenstein testiert sein. Andernfalls gilt die Prüfungsvorleistung als „nicht bestanden“ und eine Zulassung zur Klausur im folgenden Prüfungszeitraum ist nicht möglich.
- Der erste Meilenstein ist der 15.12.2020, der zweite Meilenstein ist am 15.01.2021, der letzte Meilenstein ist am 12.02.2021.
- Alle Lösungen müssen im **GitLab** der H-DA (https://code.fbi.h-da.de) im **Master-Branch** zur Verfügung gestellt werden.
- Es ist ein **Build Tool** (Make, Maven, etc.) zu verwenden.
- Die Lösungen müssen containerisiert sein und mittels **Skripten, Docker und Docker-Compose** zu testen sein
- **Jede Lösung muss getestet werden**. Schreiben Sie zu jeder ihrer Lösungen **mindestens einen funktionalen Test** sowie einen **Performance-Test**.
- Die Aufgaben werden im Rahmen eines Webinars per BigBlueButton mündlich mit dem Dozenten durchgesprochen. **Hierbei müssen alle Teammitglieder die Lösung erklären können**.
- **Zum Abschluss jeder Aufgabe muss ein Protokoll angefertigt werden**, welches die wesentlichen Ergebnisse beinhaltet. Weitere Informationen befinden sich in den jeweiligen Aufgabenstellungen.
- Die Aufgaben sind mittels **Java oder C/C++** zu lösen. Eine Kombination beider Sprachen ist erlaubt. Andere Programmiersprachen sind nur nach Absprache mit dem Dozenten zugelassen. Dasselbe gilt auch für andere Middlewarekomponenten als die oben erwähnten Sockets, REST, Apache Thrift, ProtoBuf und MQTT. Sockets und das HTTP Protokoll müssen nativ implementiert werden. Darüber hinaus ist das Einbinden weiterer Bibliotheken, z.B. von Loggern oder zur Konfiguration, erlaubt.

Aufgabenstellung

Im Rahmen des Praktikums sollen unterschiedliche Systeme rund um das Thema „Connected Cars“ simuliert werden. Dazu ist in mehreren Phasen jeweils ein Teil des Gesamtsystems zu erstellen, wie in Abbildung 1 dargestellt. Am Ende müssen mehrere Sensoren (mind. 3) mit einer Zentrale und Servern eines Dienstanbieters kommunizieren. Die Server der Dienstanbieters wiederum sollen aus Gründen der Performance und der Ausfallsicherheit redundant ausgelegt werden.

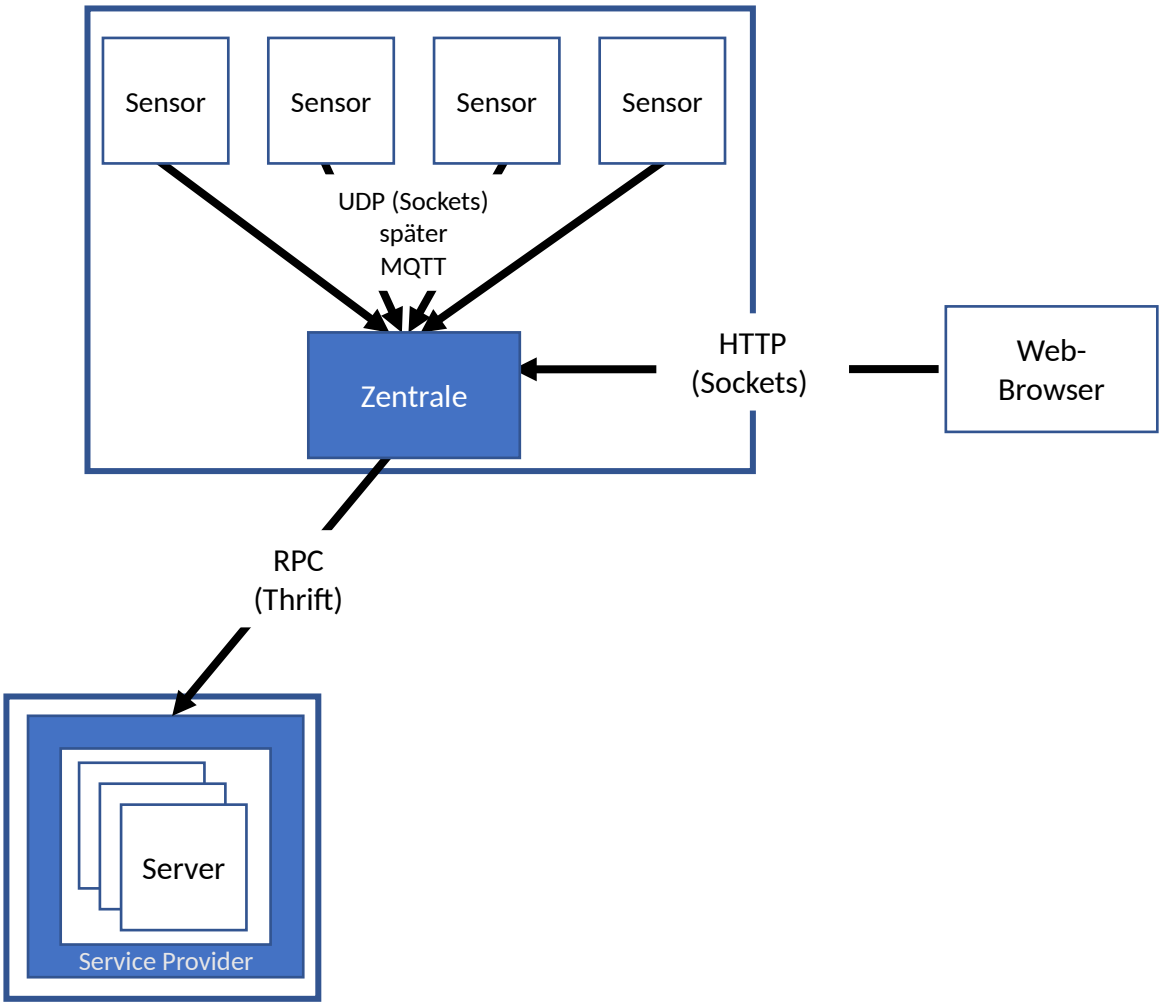

Abbildung 1: Gesamtsystem mit (nur) einer zentralen Steuereinheit, inklusive verschiedener Sensoren, die mit einem Server eines Dienstanbieters kommuniziert. Die Server des Dienstanbieteres sind redundant ausgelegt.

Beachten Sie: Es kommt in diesem Praktikum **nicht** darauf an ein möglichst realistisches Sensorverhalten zu simulieren. Der Fokus soll auf der Kommunikation der verteilten Komponenten untereinander sowie dem Software-Design, dem Testen und dem Ausrollen (Deployen) der verteilten Anwendungen liegen.

Verbindliche nicht-funktionale Anforderungen

Zusätzlich zu den von Ihnen erarbeiteten Anforderungen an die Software, sind folgende nicht-funktionale Anforderungen von Ihrem „Kunden“ vorgegeben und müssen zwingend umgesetzt werden.

1. Hygiene des Git-Repositories

Um unerwünschte Dateien aus den Git-Repositories fern zu halten, erhält jedes Repository eine vernünftige .gitignore Datei. Diese sorgt dafür, dass z.B. Bibliotheken und Kompilate nicht in das Repository eingecheckt werden können.

2. Dokumentation

Jede Software wird in einer README.md Datei dokumentiert. Diese Datei beinhaltet eine detaillierte Anleitung wie die Software kompiliert und mittels Docker und Docker-Compose gestartet und getestet werden kann.

3. Lizenzen

Jedes Repository muss über ein Lizenz-File verfügen welches die Lizenz der Software ausweist.

4. Docker und Docker-Compose

Jede Software wird containerisiert und läuft in Docker Version 19.03 sowie mit Docker-Compose Version 1.24. Das Docker-Compse File ist in Version 3.7 geschrieben.

Aufgabe 1 (Projektplan)

Zunächst soll eine Analyse der Anforderungen erstellt werden, die von der Gruppe in den Aufgaben 2 - 5 implementiert werden. Schauen Sie sich hierfür die Aufgabenstellungen 1 - 4 an und erstellen Sie die Anforderungen für jeden Termin **schriftlich** – am Besten in GitLab-Issues. Beachten Sie, dass das Bestehen der weiteren Termine davon abhängt, ob die erstellten Anforderungen umgesetzt wurden. Die Anforderungen sollen neben der zu verwendenden Programmiersprache, den Kommunikationstechnologien auch ein Systemdesign inkl. der Übertragungsformate sowie ein Konzept der Test- und Simulationsumgebung für jeden Termin enthalten.

Änderungen am Konzept sind im Laufe des Praktikums erlaubt, müssen jedoch dokumentiert werden. Am Ende des Termins müssen dem Dozenten die Anforderungen vorgestellt und ggf. ergänzt werden. Schließlich wird ein Anforderungsdokument als Protokoll abgegeben. Dieses dient als Checkliste für die folgenden Termine.

In a Nutshell

- Erstellen Sie eine Anforderungsanalyse (funktionale und nicht-funktionale Anforderungen) für das Gesamtsystem und fixieren Sie die Anforderungen schriftlich, z.B. in GitLab
- Leiten Sie aus Ihren Anforderungen ein erstes grobes Systemdesign ab und überlegen Sie, wie die verschiedenen Komponenten miteinander interagieren.
- Leiten Sie aus den Anforderungen und dem Systemdesign funktionale, nicht-funktionale sowie Performance-Tests ab.
- Leiten Sie aus den Anforderungen und dem Systemdesign einen **Projektplan** ab. Geben Sie Tasks an, die für die weiteren Aufgaben bearbeitet werden müssen, z.B. in GitLab
- Überlegen Sie, wie Sie die unterschiedlichen Systeme effizient operativ ausrollen (deployen) können. Fixieren Sie Ihr Konzept schriftlich

Die (harte) Deadline für diese Aufgabe ist der **1. Meilenstein**. Zu diesem Zeitpunkt muss eine konsistente, testierfähige schriftliche Ausarbeitung zu Aufgabe 1 vorliegen. Andernfalls gilt die Prüfungsvorleistung als „nicht bestanden“. Die weiteren Aufgaben werden – unabhängig von ihrer Güte – erst testiert, wenn Aufgabe 1 erfolgreich bestanden ist.

Lernziele

Die erste Aufgabe hat unter anderem folgende Lehr- und Lernziele:

- Selbstständiges Arbeiten
- Durchführen einer Anforderungsanalyse
- Herausarbeiten sinnvoller funktionaler und nicht-funktionaler Tests
- Erstellen eines ersten, groben Software- bzw. Systemdesigns
- Aufstellen eines validen Projektplans
- Anfertigen eines Protokolls

Aufgabe 2 (UDP und TCP Sockets)

Im ersten Schritt sollen Sensoren einem zentralen Steuerrechner Informationen über den Zustand des Fahrzeugs liefern. Dazu sollen Füllstand des Tanks (in Prozent), Kilometerstand, Verkehrssituation (frei,mäßiger Verkehr, starker Verkehr, Stau) und Durchschnittsgeschwindigkeit von jeweils einem Sensor erfasst werden. Jeder Sensor soll als eigenständiger Prozess bzw. eigenständiger Container laufen. Die Informationen sollen sich ständig ändern und in einem geeigneten Nachrichtenformat mittels UDP an die Zentrale übermittelt werden. Dort sollen die Nachrichten unter Angabe von IP, Port und Typ des Sensors auf der Standardausgabe ausgegeben werden.

Darüber hinaus muss in der Zentrale ein einfacher HTTP-Server implementiert werden, der mindestens den HTTP GET Befehl korrekt und vollständig verarbeiten kann. Die HTTP-Schnittstelle soll über eine REST-API den Zugriff auf einzelne Sensordaten, alle Sensordaten sowie die Historie der

Sensordaten (jeweils mit einer eigenen URI) ermöglichen. Dazu müssen auch die Daten aus der Vergangenheit in der Zentrale gespeichert werden.

Der HTTP Server soll **ohne Hilfsmittel (d.h. ohne vorhandene Bibliotheken)** implementiert werden und mindestens HTTP GET unterstützen. Es ist hierbei erforderlich, dass eingehenden HTTP GET Anfragen komplett und korrekt eingelesen und verarbeitet werden. Das bedeutet u.a., dass es nicht ausreichend ist, die erste Zeile eine HTTP Nachricht zu lesen. Zudem müssen die Sensoren weiter laufen. Das bedeutet, dass die Zentrale gleichzeitig mit den Sensoren als auch mit HTTP Klienten in Kontakt bleiben soll.

Lernziele

Die zweite Aufgabe hat unter anderem folgende Lehr- und Lernziele:

- Selbstständiges Arbeiten
- Software Engineering und Design
- Kommunikation mittels UDP- und TCP-Sockets
- Implementierung und Verwendung von HTTP und REST
- Effizientes Deployment unterschiedlicher Anwendungen, z.B. mittels Skript

Aufgabe 3 (RPC)

Für die dritte Aufgabe sollen die zuvor implementierten Zentrale ihren Status, d.h. die aktuellen Werte der Sensoren, über Thrift an die Cloud eines Dienstanbieters übermitteln. Hierzu muss eine per Thrift-Datei beschriebene API sowohl am Server (Server des Anbieters) als auch am Client (Zentrale) implementiert werden. Der Anbieter soll die so übermittelten Daten persistent speichern.

Lernziele

Die dritte Aufgabe hat unter anderem folgende Lehr- und Lernziele:

- Selbstständiges Arbeiten
- Software Engineering und Design
- Einbinden und Anwenden externer Software-Bibliotheken
- Einbinden und Anwenden von RPCs am Beispiel von Thrift
- Implementieren einer vorgegebenen bestehenden API

Aufgabe 4 (MoM mittels MQTT)

Ihr Kunde stellt nun fest, dass das Anbinden der Sensoren an die Zentrale mittels UDP keine gute Design-Entscheidung war. Um das System besser skalieren zu können, sollen die Sensoren nun mittels MQTT an die Zentrale angeschlossen werden.

Überarbeiten Sie Ihre in Aufgabe 2 implementierten Sensoren und die Zentrale so, dass die Daten nun mit MQTT übertragen werden.

Lernziele

Die fünfte Aufgabe hat unter anderem folgende Lehr- und Lernziele:

- Selbstständiges Arbeiten
- Software Engineering und Design
- Einbinden und Anwenden externer Software-Bibliotheken
- Einbinden und Anwenden einer Message-oriented Middleware am Beispiel von MQTT

Aufgabe 5 (Hochverfügbarkeit und Konsistenz)

Für die fünfte Aufgabe sollen die Server des Dienstanbieters aus Gründen der Ausfallsicherheit redundant ausgelegt werden. Der Anbieter betreibt daher mindestens mind. zwei Server parallel in einer Primary-Secondary Architektur im Hot-Standby Betrieb. Die Server tauschen unter Verwendung eines RPCs (Thrift oder Protobuf) untereinander die empfangenen Daten aus. Dabei muss sichergestellt werden, dass alle Daten auf allen Servern konsistent vorliegen.

Um die Ausfallsicherheit des Gesamtsystems zu testen, soll es während des Betriebs immer wieder zu zufälligen (simulierten) Ausfällen einzelner Server kommen. Darüber hinaus soll die Performance des Systems getestet werden.

Lernziele

Die fünfte Aufgabe hat unter anderem folgende Lehr- und Lernziele:

- Selbstständiges Arbeiten
- Software Engineering und Design
- Auswahl, Design und Implementierung von Hoch-Verfügbarkeits (HA) und Konsistenz-Modellen

Bonusregelung

Mit Hilfe des Praktikums kann ein **0.3-Noten-Bonus** für die Klausur erworben werden. Der Bonus ist nur einmal gültig – nämlich exakt für die in diesem Semester anstehende Klausur. Der Bonus kann dann erteilt werden, wenn Sie eine herausragende Gesamtlösung (Aufgaben 1-5) präsentieren die deutlich über den Anforderungen und den Erwartungen liegt. Der Bonus wird zudem nur dann wirksam, wenn Sie die Klausur auch ohne Bonus mit mindestens 4.0 bestehen.