Praktikum Verteilte Systeme

Sommersemester 2021

Prof. Dr. Michael von Rüden

Übersicht

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

Rahmenbedienungen

- Das Bearbeiten der Praktikumsaufgaben findet in Zweier-Teams statt.
- Die Praktikumsaufgaben müssen zuhause vor- und nachbereitet werden, da die Bearbeitungszeit während des Praktikumstermins nicht ausreicht, um eine vollständige Lösung zu implementieren.
- **Die Anforderungsanalyse muss spätestens zum 3. Praktikumstermin testiert sein.** Alle Aufgaben müssen spätestens zum letzten Termin Ihrer Praktikumsgruppe testiert sein. Andernfalls gilt die Prüfungsvorleistung als „nicht bestanden“ und eine Zulassung zur Klausur im folgenden Prüfungszeitraum ist nicht möglich.
- Alle Lösungen müssen im **GitLab** der H-DA (https://code.fbi.h-da.de) in Ihrem Repo im **Master-Branch** zur Verfügung gestellt werden. Ein Repo wird Ihnen zur Verfügung gestellt.
- 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 Tests sollen einen Aspekt der Anwendung testen, der mit der Verteiltheit zu tun hat (also z.B. Durchsatz, Antwortzeit, Verhalten beim Senden/Empfangen von fehlerhaften Daten, etc.).
- Die Aufgaben werden im Rahmen eines Webinars per BigBlueButton mündlich mit dem Dozenten durchgesprochen. **Hierbei müssen alle Teammitglieder die Lösung erklären können**.
- 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 „Stromerzeugung/ Kraftwerke“ simuliert werden. Dazu ist in mehreren Phasen jeweils ein Teil des Gesamtsystems zu erstellen, wie in Abbildung 1 dargestellt.

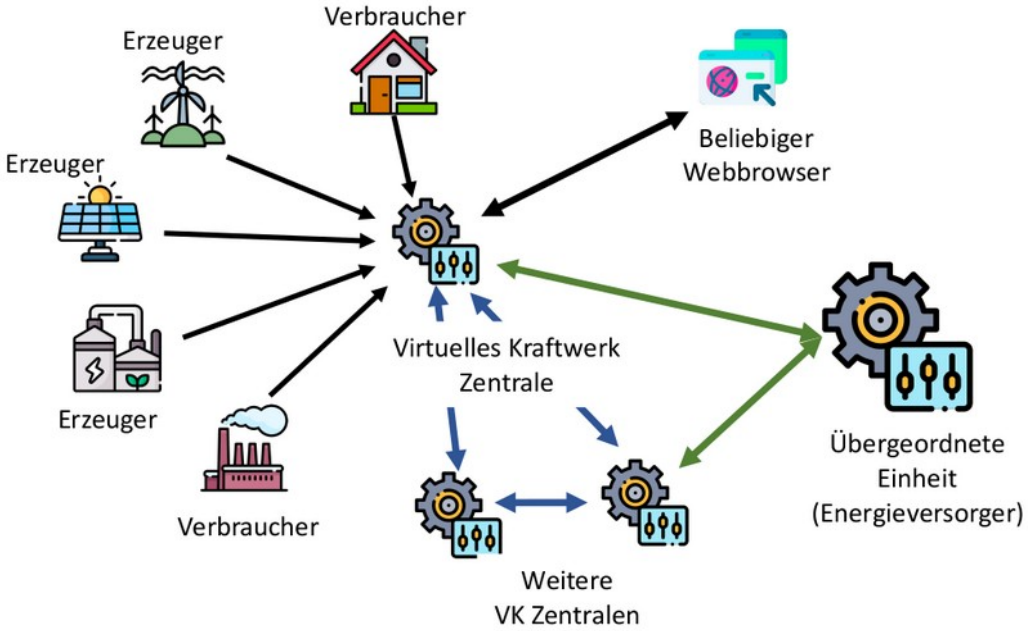

Abbildung 1: Gesamtsystem mit Erzeugern und Verbrauchern sowie Virtuellen Kraftwerk-Zentralen und Energieversorgern.

Die Thematik der Aufgaben ist ein virtuelles Kraftwerk, darunter versteht man eine Anzahl von Stromerzeugern und -verbrauchern, die lokal zusammengeschaltet und verwaltet werden. Genauer besteht ein virtuelles Kraftwerk also aus einer Menge an Komponenten, die jeweils Erzeuger (z.B. Solar-, Wind-, Biomassekraftwerke) oder Verbraucher (z.B. Haushalte, Unternehmen) sind. In realen Umgebungen sind häufig auch noch Stromspeicher vorhanden, dies soll hier jedoch ignoriert werden. In einem virtuellen Kraftwerk werden alle Beteiligten idealerweise so gesteuert, dass Bedarf und Angebot an Strom immer ausgeglichen sind. Man kann auch mehrere solcher virtueller Kraftwerke zusammenschalten, um so flexiblere Einheiten mit größeren Kapazitäten zu erhalten.

Im Rahmen des Praktikums sollen Sie mit den in der Vorlesung diskutierten Methoden virtuelle Kraftwerke simulieren. Der Detailgrad der Simulation kann relativ einfach sein und muss nicht der Realität entsprechen, die in der Aufgabenstellung genannten Anforderungen müssen aber zwingend erfüllt sein. Es kommt in diesem Praktikum **nicht** darauf an ein möglichst realistisches Verhalten 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. Zudem enthält die Dokumentation Ihre Testergebnisse.

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 Aufgaben 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 **3. Praktikumstermin**. 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)

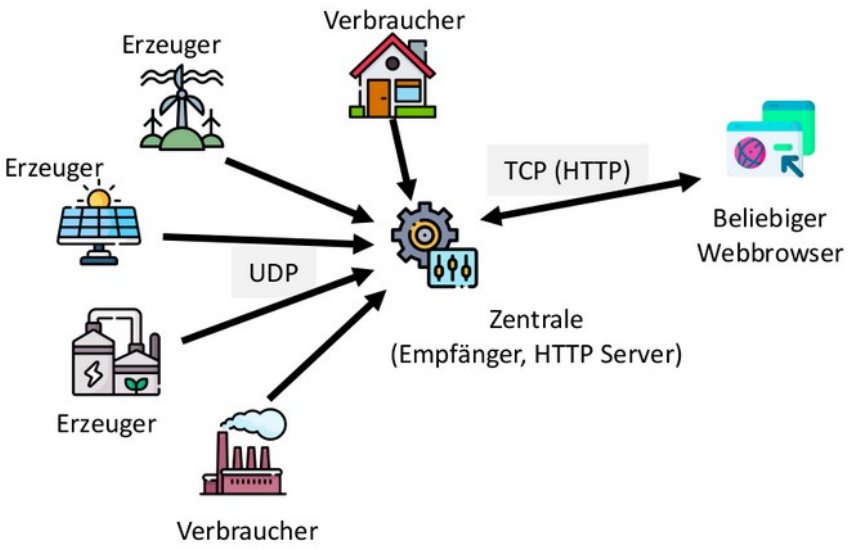

Abbildung 2: Übersicht über ein virtuelles Kraftwerk.

Programmieren Sie in der ersten Aufgabe die folgenden Komponenten (als individuell laufende Prozesse) eines virtuellen Kraftwerks:

1. eine Zentralsteuerung eines virtuellen Kraftwerks
2. mindestens 4 unabhängige Erzeuger/Verbraucher, davon mindestens 1 Verbraucher

Von allen Erzeugern und Verbrauchern soll die a**ktuell produzierte bzw. benötigte Strommenge** (Einheit kW) an die Zentrale gemeldet und von der Zentrale sowohl auf der Konsole als auch über HTTP für Webbrowser zugreifbar dargestellt werden. Dabei sollen die Erzeuger/Verbraucher als Sensoren agieren, die mindestens jeweils die folgenden **Informationen über UDP** an die Zentrale verschicken (weitere können hinzugefügt werden, wenn es Ihnen sinnvoll erscheint):

1. Art des Teilnehmers (Erzeuger, Verbraucher)
2. Eindeutige ID oder Name des Teilnehmers
3. Aktuelle Strommenge (in kW oder einer anderen Einheit

Simulieren Sie hier wechselnde Strommengen geeignet, d.h. es sollen sich im Lauf der Simulation deutliche Veränderungen ergeben, z.B. produzieren Solarkraftwerke produzieren nachts keinen Strom, bei Windstille gibt es keinen Strom aus Windkraft, etc. Außerdem sollen einzelne Komponenten wegen Defekts ausfallen können. Außerdem können Erzeuger natürlich nicht beliebig viel Strom produzieren, sondern besitzen eine Maximalkapazität.

In die Zentrale soll ein HTTP integriert sein. Dieser HTTP Server soll über eine REST-API Informationen zu den Erzeugern/Verbrauchern inklusive des aktellen Verbrauchs und der Verbrauchshistorie (jeweils mit einer eigenen URI) liefern.

Der HTTP Server soll **ohne Hilfsmittel (d.h. ohne vorhandene Bibliotheken)** implementiert werden und mindestens HTTP GET unterstützen. Der Server soll außerdem mit beliebigen Browsern arbeiten und dabei den User-Agent des Browsers protokollieren, der die Anfrage geschickt hat. 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 Erzeuger/Verbraucher weiter laufen. Das bedeutet, dass die Zentrale gleichzeitig mit den Erzeugern/Verbrauchern 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 in Docker -Containern mit Docker-Compose

Aufgabe 3 (RPC)

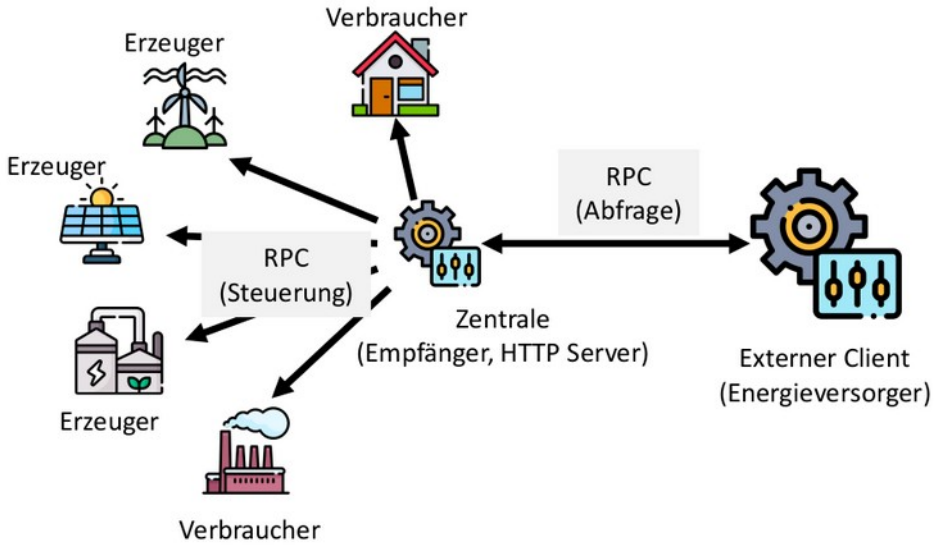

Abbildung 3: Steuerung der Komponenten, Abfrage des Status über RPC (Funktionalität aus Aufgabe 1 bleibt erhalten).

Für die zweite Aufgabe soll die zuvor implementierte Zentralsteuerung des Kraftwerks den aktuellen Status sowie die gesamte Historie (das sind mglw sehr viele Daten) aller Komponenten des Kraftwerks an einen externen Client (das kann z.B. eine übergeordnete Einheit eines großen Energieversorgers sein) übermitteln. Diese Datenübermittlung soll mittels RPC (Apache Thrift oder gRPC/Protobuf) erfolgen. Außerdem soll es der Zentrale möglich sein, über eine geeignete RPC Schnittstelle möglich sein, einzelne Komponenten an- und abzuschalten (oder allgemein bei Erzeugern die Leistung zu erhöhen), um einen Lastausgleich herbeizuführen. Dazu benötigen die Komponenten (also Erzeuger und Verbraucher) ebenfalls eine RPC Schnittstelle.

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 oder gRPC/Protobuf

Aufgabe 4 (MoM mittels MQTT)

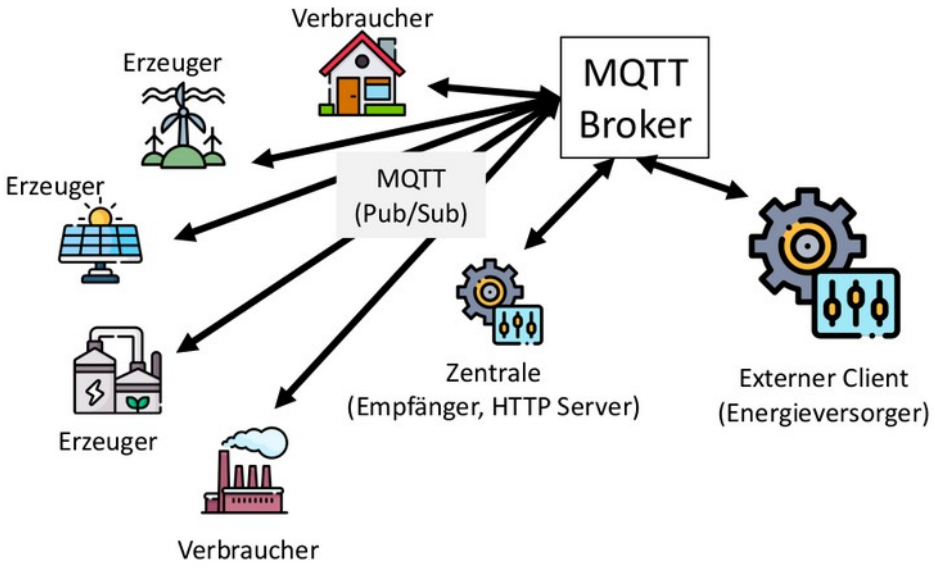

Abbildung 4: Austausch der Statusinformationen über MQTT.

Ihr Kunde stellt nun fest, dass das Anbinden der Erzeuger/Verbraucher an die Zentrale mittels UDP keine gute Design-Entscheidung war. Um das System besser skalieren zu können, sollen in einer fortgeschrittenen Version Ihrer Software, die Erzeugern/Verbrauchern nun mittels MQTT, also mittels Publish/Subscribe an die Zentrale angeschlossen werden.

Überarbeiten Sie Ihre in Aufgabe 2 implementierten Erzeuger/Verbraucher und die Zentrale so, dass die Daten nun mit MQTT übertragen werden. Führen Sie zudem einen **sinnvollen Performance-Vergleich** zwischen der alten (UDP) und der neuen (MQTT) Implementierung durch.

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)

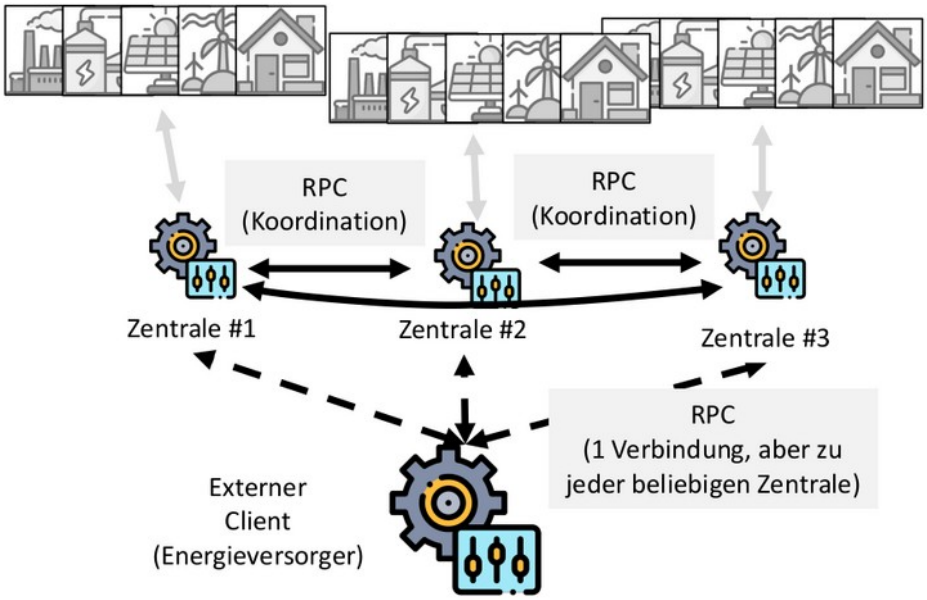

Abbildung 5: Kooperierende Zentralen die mittels RPC miteinander kommunizieren.

Für die fünfte Aufgabe sollen mehrere virtuelle Kraftwerke zu einem größeren System zusammengeschaltet werden, so dass die jeweiligen Zentralen miteinander kommunizieren und kooperieren. Es soll nun möglich sein, dass jede einzelne Zentrale (über RPC) vom externen Client nach dem Status aller Komponenten im gesamten Kraftwerkssystem abgefragt werden kann, unabhängig vom Standort der jeweiligen Komponenten. Dazu muss in jeder Zentrale der gesamte Status aller Komponenten vorhanden sein. Die Abfrage soll jeweils ein konsistentes Bild des gesamten Systems liefern und vom Client angezeigt werden.

Überlegen Sie dazu, welche Art von Konsistenz hier implementiert muss und wie man erreichen kann, dass der Status der Komponenten einzelner Kraftwerke an andere Zentralen verteilt werden kann.

Um die Ausfallsicherheit des Gesamtsystems zu testen, soll es während des Betriebs immer wieder zu zufälligen (simulierten) Ausfällen einzelner Zentralen kommen. Überlegen Sie sich, wie der externe Client mit Ausfällen oder Verzögerungen bei der Übertragung von Zentralen bzw. deren Wiederverfügbarkeit umgehen soll. Das Ziel ist die jederzeit konsistente Darstellung des Systemzustands (Konsistenz ist hier im Sinne der Vorlesung gemeint). Überlegen Sie, inwieweit dieses Ziel erreicht werden kann und welche Abstriche man ggfs. machen muss. Für diese Aufgabe können Sie ebenfalls Apache Thrift oder gRPC/Protobuf verwenden.

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.