Verteilte Systeme Praktikum

Bearbeiter:
- Matthias Adrian (Mtrk. 752237)
- Jan Zipprich (Mtrk. 757956)

1.2 Aufgabe 1a - UDP Sockets:

Generell:

Deployment:
Das Deployment geschieht mittels Docker in einzelnen Containern. Somit sind alle Teile voneinander getrennt. Jeder Sensor befindet sich in einem eigenen Container. Um die Container zu starten wird das Tool docker-compose verwendet um mehrere Dienste auf einmal starten und verwalten zu können. Um die Logs der einzelnen Dienste sehen und verwalten zu können wird zusätzlich Portainer aufgesetzt.

- Programmiersprache(n): Java
- Build Tool: Maven
- Übertragungsformat: JSON
  - { “timestamp”:long, (Current System milliseconds)
  - “name”:string, (Name des Sensors)
  - ”sensor_type”:string, (Typ des Sensors)[“Temperatur”,”Helligkeit”,”Niederschlag”,”Wind”]
  - ”value”:float (Wert des Sensors) }
- Beispiel:

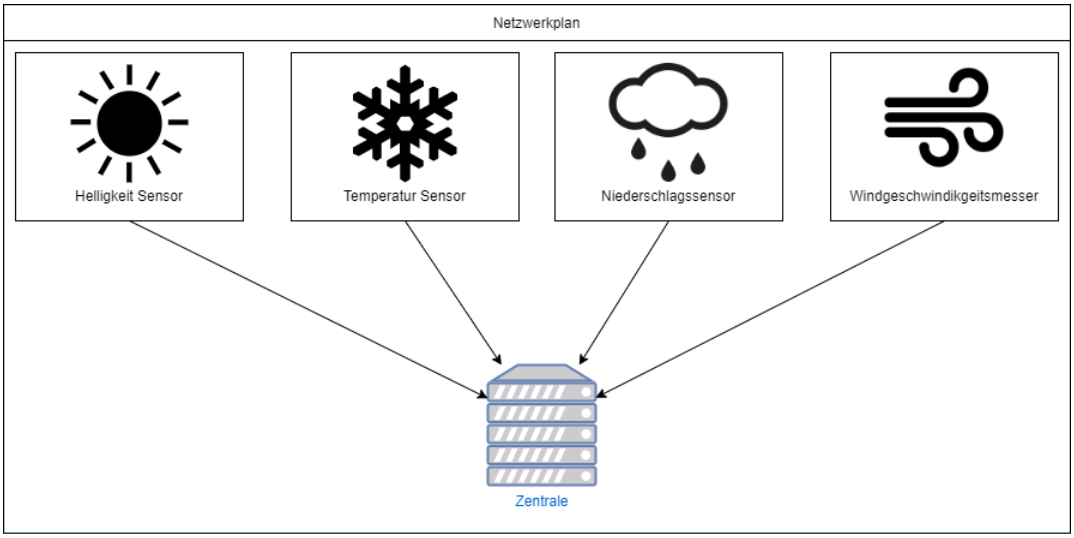
```
{
    "timestamp": "2020.05.09 22:30:00",
    "name": "Dachterasse",
    "sensor_type" :"Temperatur",
    "value" : 23
}
```

Vorlage für Testdokumentation:

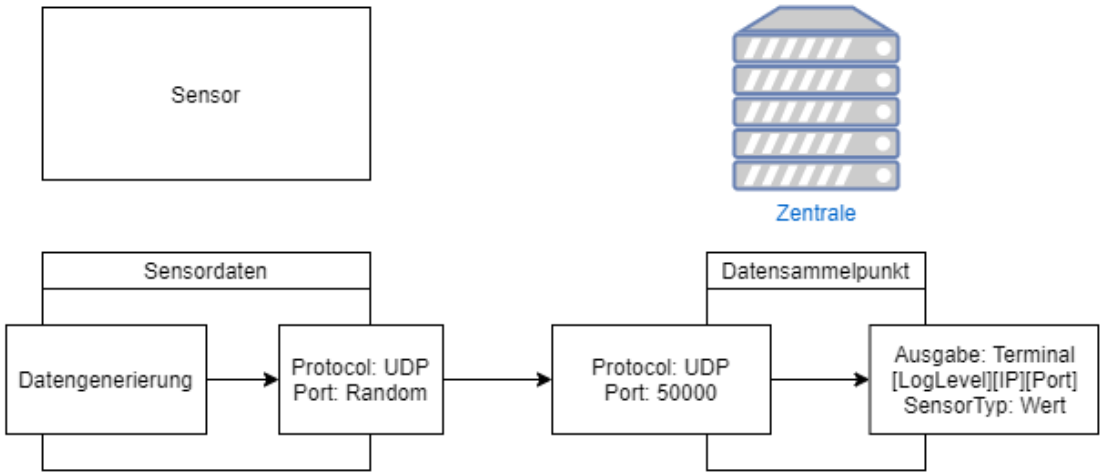
Name des Tests	Was wurde getestet?	Wie wurde es getestet?	Erwartetes Ergebnis	Tatsächliches Ergebnis
Test 1	...	...	...	...

A)

Systemdesign:



Es gibt vier verschiedene Sensoren für: Helligkeit, Temperatur, Niederschlag und Windgeschwindigkeit. Diese Sensoren befinden sich in jeweils eigenen Programmen und schicken kontinuierlich Sensordaten an die Zentrale.



Auf den jeweiligen Sensoren läuft jeweils ein Programm, welches aus zwei Komponenten besteht, der Datengenerierung und des Datentransfers. Bei der Datengenerierung werden alle 10 Sekunden neue Messdaten erzeugt und an den Datentransfer übergeben. Der Datentransfer ist dafür zuständig die Daten mittels UDP an die Zentrale zu schicken.

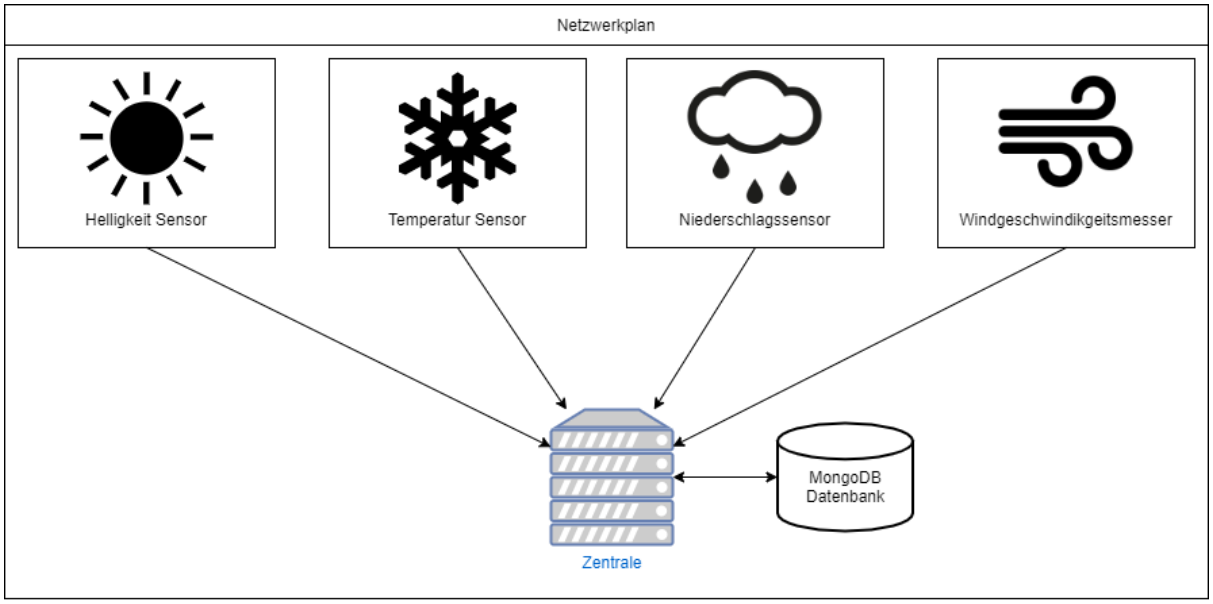
Übertragene Daten beinhalten dabei den Sensortyp, Name des Sensors (Standort), Zeitpunkt der Erfassung sowie den dazugehörigen Sensorwert. Die Daten werden im JSON Format übertragen.

Die Zentrale erhält die Sensordaten mittels UDP auf dem Port 50000. Die Daten werden nach dem Empfang an eine Konsolenausgabe weitergeleitet um diese auf der Konsole anzeigen zu können. Bei der Konsolenausgabe wird dafür das Muster “[LogLevel][IP][Port][SensorTyp] SensorName: Wert” verwendet.

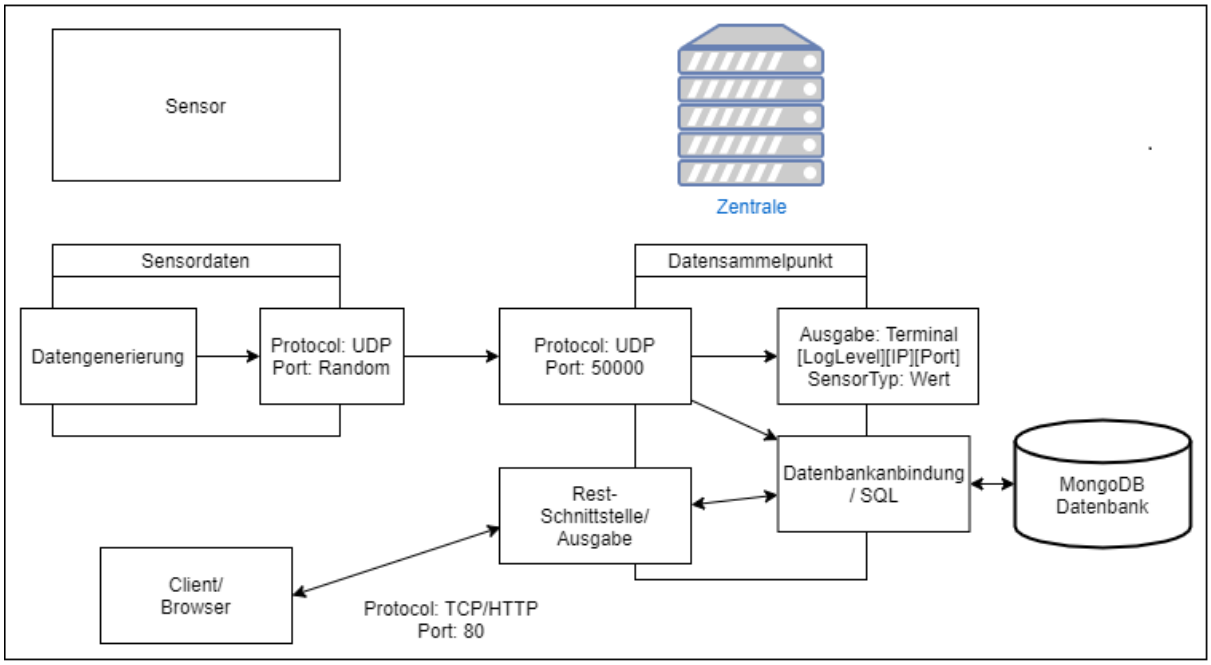
Funktionale Anforderungen	Nicht funktionale Anforderungen
- ständig ändernde Werte sollen an den Host gesendet werden per UDP	- Nachrichtenformat, um Daten auf dem Host korrekt einzulesen: z.B. CSV (comma separated value)
- Daten sollen beinhalten: Typ des Sensors, Wert	- Jeder Sensor soll in eigenen Speicherbereich laufen (d.h. keine Threads)
- UDP Pakete, die dem Übertragungsformat nicht genügen, soll der Server ignorieren	
- Daten sollen bei der Zentrale unter Angabe von IP, Port und Type des Sensors auf der Standardausgabe ausgegeben werden	

Funktionale Tests	Nicht funktionale Tests
- 10 Sensorpakete an die Zentrale senden, und überprüfen ob auch tatsächlich 10 Pakete angekommen sind	- In einem Zeitraum von x Sekunden, so viele UDP-Pakete wie möglich an die Zentrale senden, und zählen wie viele auf der Zentrale ankommen
- Das korrekte einlesen der Daten überprüfen, in dem die Daten aus den UDP-Paketen auf der Zentrale eingelesen, und auf der Konsole ausgegeben werden	- Messen der durchschnittlichen Latenz - 10 Pakete an die Zentrale senden, und die durchschnittliche Latenz berechnen
- Daten an die Zentrale senden, die nicht dem Übertragungsformat entsprechen	

1.3 Aufgabe 1b - TCP Sockets



Zusätzlich zu dem Netzplan aus Aufgabe A kommt nun ein MongoDB Server hinzu um die Sensordaten persistent speichern zu können.



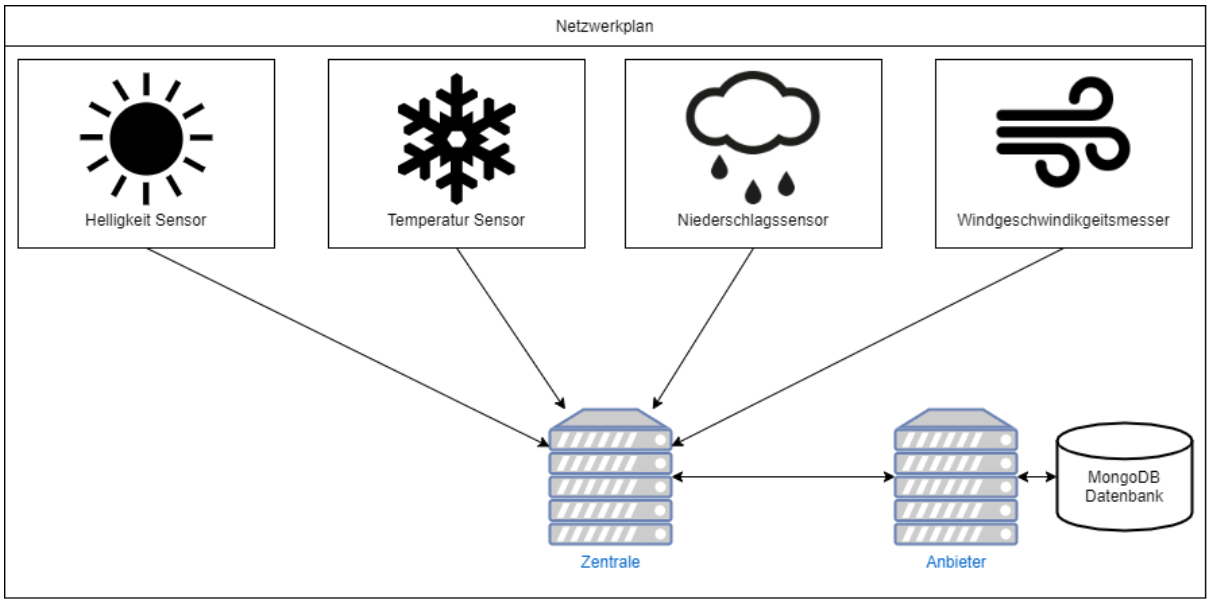
Die Übermittlung der Sensordaten erfolgt genauso wie die Ausgabe auf die Konsole gleich. Die Daten werden zusätzlich in eine NoSQL Datenbank geschrieben um zwischengespeichert zu werden. Die Informationen können zusätzlich mit einem Browser vom Client/Kunden abgefragt werden. Hierfür wird die Abfrage über eine HTTP/Rest Schnittstelle zur Verfügung gestellt.

Funktionale Anforderungen	Nicht funktionale Anforderungen
- HTTP Server aufsetzen, der Clients	- HTTP GET Anfragen sollen

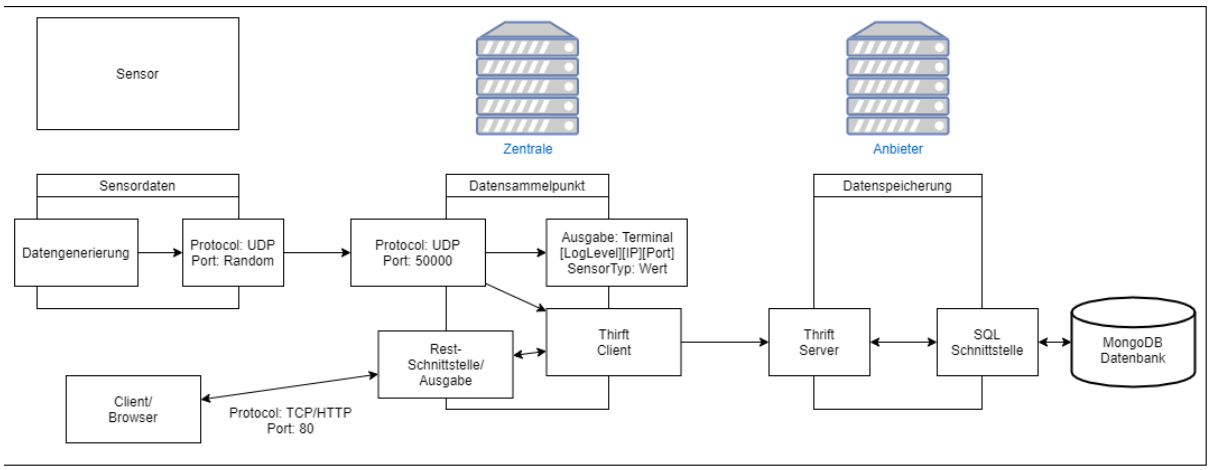
Sensordaten anzeigen kann	vollständig bis Zeilenende eingelesen werden
- Persistieren der Daten auf dem Host mittels (SQL)-Datenbank	
- Mittels URIs soll der Client einzelne Sensordaten, alle Sensordaten, sowie die Historie der Sensordaten angezeigt bekommen	
- Dies soll über eine REST-API erfolgen, d.h. bei 3 Sensoren wären 7 verschiedene URIs notwendig	
- Die Zentrale soll zeitgleich Clients bedienen, und Sensordaten einlesen können	

Funktionale Tests	Nicht funktionale Tests
- 10 Testdaten an die Zentrale senden, und mittels HTTP GET Request überprüfen, ob auch tatsächlich 10 Daten angezeigt werden	- Zwei oder mehr Clients stellen eine HTTP GET Request, und es wird überprüft ob die Zentrale mehrere Clients zeitgleich bedienen kann
- Eine nicht unterstützte GET-Anfrage stellen und schauen wie die Zentrale reagiert	

1.4 Aufgabe 2 - Remote Procedure Calls (RPC)



Es wird nun ein zweiter Server, der Anbieter hinzugefügt. Hierbei werden die Daten von der Zentrale an den Anbieter geschickt, welcher diese endgültig speichert. Die Daten werden somit bei der Zentrale temporär und beim Anbieter permanent gespeichert.



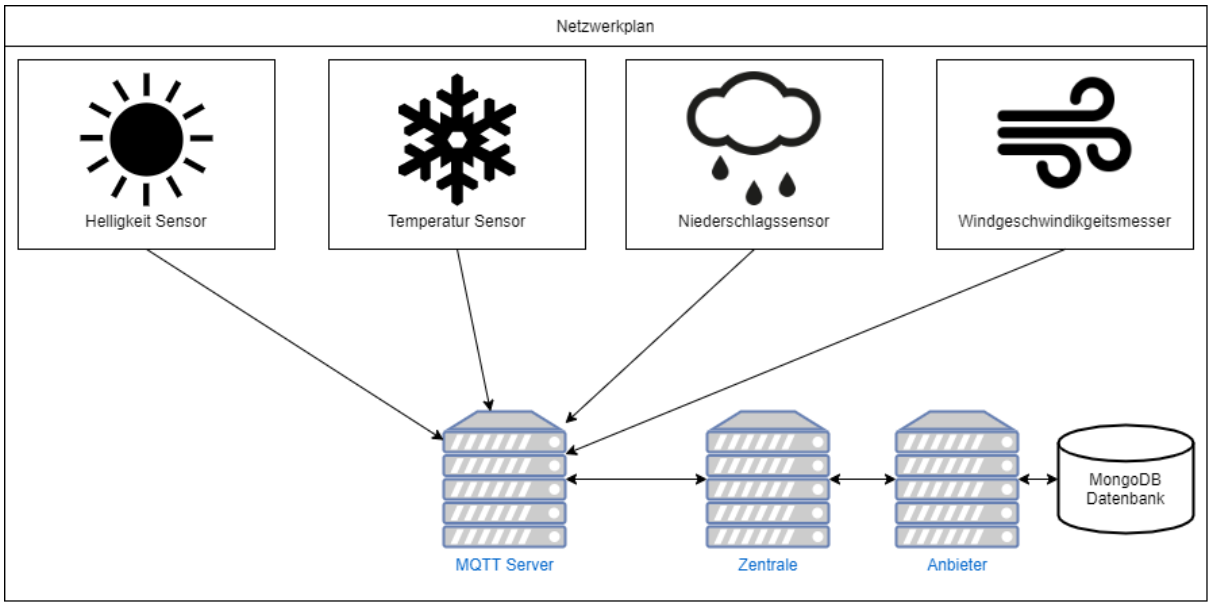
Die Daten werden nicht mehr auf der Zentrale sonder dem Anbieter gespeichert werden, da dieser über mehr Speicherkapazitäten verfügt. Die Sensordaten werden dabei von der Zentrale über einen Thrift Client mithilfe von RPC (Remote Procedure Call) an den Thrift Server des Anbieters geschickt. Auf der Seite des Anbieters werden die Daten in einer SQL (oder MongoDB) Datenbank abgespeichert.

Funktionale Anforderungen	Nicht funktionale Anforderungen

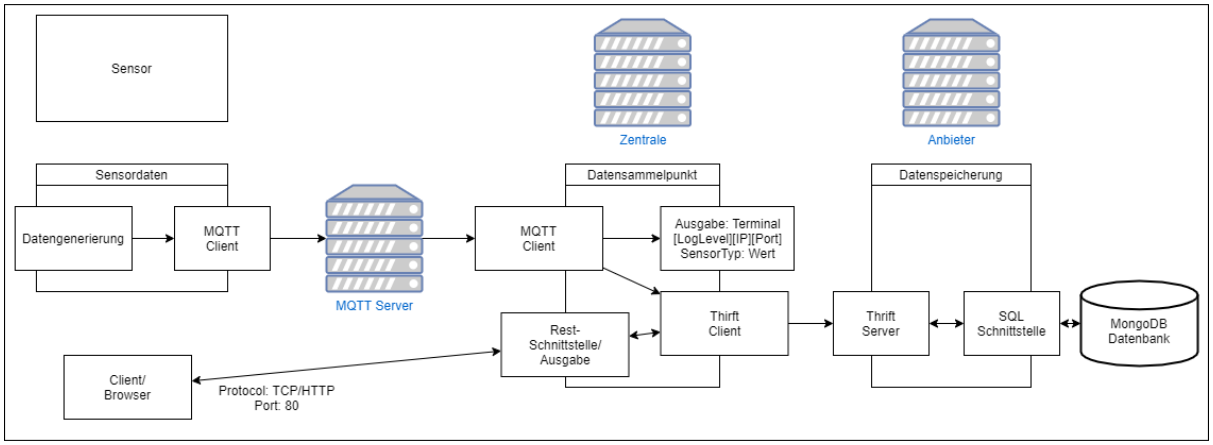
- Die Zentrale übermittelt die Sensordaten per Thrift an Server des Anbieters	- Festlegen, wie viele Daten auf einmal an den Anbieter gesendet werden (z.B. jeweils die 10 aktuellsten Sensordaten)
- Thrift API festlegen (d.h. Typ der Variablen und Funktionen)	- Festlegung einer sinnvollen Datenbank
- Der Anbieter soll die Daten persistieren (-> Datenbank)	- Da als Nachrichtenformat JSON benutzt wird, wäre eine MongoDB sinnvoll und effektiv

Funktionale Tests	Nicht funktionale Tests
- Konsistenz der Zentrale und des Anbieters überprüfen, indem die Ausgaben verglichen werden	- Latenz zwischen Zentrale und Anbieter Server messen (z.B. 10 Thrift Daten senden und daraus die durchschnittliche Latenz berechnen)
- 10 Daten versenden, 10 persistieren und Datenbestand lokal und auf Datenbank des Anbieters auf Konsistenz überprüfen	- Performance des Servers testen, indem massenhaft RPC Pakete an den Anbieter geschickt werden, dann geschaut wird wie viele er innerhalb einer bestimmtes Zeitraumes tatsächlich persistieren kann
-	

1.5 Aufgabe 3 - MoM mittels MQTT



Es wird nun ein MQTT Server als Zentralen Dienst für die Kommunikation der Sensoren verwendet. Die Sensoren melden sich am MQTT Server an und schicken dort alle Sensordaten hin. Die Zentrale registriert sich um über neue Daten informiert zu werden.



Die Sensordaten können nun sowohl über UDP als auch TCP an den MQTT broker übermittelt werden. Das Protokoll hängt dabei von dem eingestellten Quality of Service ab. Die Zentrale meldet sich auch an dem MQTT Server an um neue Sensordaten empfangen zu können.

Funktionale Anforderungen	Nicht funktionale Anforderungen
- Die Daten sollen per MQTT statt UDP an die Zentrale übermittelt werden	- Jeder Sensor soll in eigenen Speicherbereich laufen (d.h. keine Threads)

- Sensoren publishen Daten an den MQTT-Broker	- Festlegung des MQTT QoS Levels (0, 1 oder 2)
- Zentrale ist Subscriber von dem MQTT-Broker und erhält von ihm die Sensordaten	-

Funktionale Tests	Nicht funktionale Tests
- 10 Pakete an die Zentrale publishen, und überprüfen ob auch tatsächlich 10 Pakete angekommen sind	- In einem Zeitraum von x Sekunden, so viele MQTT-Pakete wie möglich an den Broker publishen, und zählen wie viele bei der Zentrale ankommen
- Das korrekte einlesen der Daten überprüfen, in dem die Daten aus den MQTT-Pakete auf der Zentrale eingelesen, und auf der Konsole ausgegeben werden	- Messen der durchschnittlichen Latenz - 10 Pakete an die Zentrale publishen, und die durchschnittliche Latenz berechnen
	- Vergleich der Latenz zwischen UDP und MQTT ziehen

1.6 Bonusaufgabe - Hochverfügbarkeit und Konsistenz

Vorläufige Überlegung