BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Fachbereich VI · Informatik und Medien

Studiengang Medieninformatik Master

Masterprojekt

Dokumentation - IoT-Backend

Konzipierung und Implementierung eines Prototypen für ein cloud-basiertes IoT Datenportal

Betreuer: Prof. Dr.-Ing. Edzard Höfig

Tom Wieschalla: s821990
Ray Gläske: s825259
Steven Maasch: s822166
Stefan Boitschuk: s825894
Elias Lerch: s823488
Lukas Runge: s828343

Inhaltsverzeichnis

Seite

1 Einführung **3**

2 Spezifikationen **4**

3 Aufbau des Gesamtsystems **6**

3.1 Infrastruktur 6

3.1.1 Gesamtsystem 7

3.1.2 Java-Backend 10

3.1.2.1 Schnittstelle zur Datenverarbeitung 10

3.1.2.2 DynamoDB-Key Schema 12

3.1.2.3 REST-Schnittstelle 12

3.1.3 NodeJS-Frontend 12

3.1.4 Daten-Generator 13

3.1.5 Provisioner 13

3.1.5.1 Docker 13

3.1.5.2 Puppet 13

4 Qualitätssicherung **14**

4.1 REST-Schnittstelle 14

4.2 NodeJS-Frontend 14

4.3 Performance 14

5 Installationsbeschreibung **15**

6 Anhang **16**

Kapitel 1

Einführung

Das Internet der Dinge ist im heutigen Umfeld ein realer und oft verwendeter Begriff. Er beschreibt den Umstand, die Realität, wie wir sie kennen, mit der Digitalität zu verknüpfen, beziehungsweise sie zu vereinen. Dies bedeutet auf der einen Seite die Integration vernetzter Rechner mit unserer Umwelt und auf der anderen Seite, die Spiegelung realer Dinge als eine Art Datenschatten im Netz.

Das in dieser Dokumentation beschriebene System befasst sich mit der kollektiven Sammlung, Anzeige und Speicherung von Daten aus unserer Umwelt. Mögliche Daten werden in diesem Szenario von Sensoren erzeugt, die ihre Beschleunigung, Position, Orientierung oder auch die Temperatur ihrer Umgebung versenden können. Verknüpft mit der ersten Komponente, dem RabbitMQ Message Broker, können sie ihre Daten via verschiedener Kanäle/Topics an jeweilige Interessenten übermitteln. Die Idee dahinter kann man als eine Art Publish-Subscribe-Konzept beschreiben. Ein einzelner Sensor verschickt seine Daten über einen einzigartigen Kanal und gibt so mehreren Interessenten die Möglichkeit, seine Daten abzufangen und zu benutzen.

Das Ziel des hier beschriebenen Prototypen ist es, ein cloud-basiertes Datenportal bereit zu stellen, welches Sensordaten entgegenimmt, diese Daten menschenlesbar aufbereitet, für Nutzer mittels einer Weboberfläche zugreifbar und live verfolgbar macht und alle Daten in einer Big Data Umgebung abspeichert. Des weiteren muss das System skalierbar und cloudfähig sein.

Ein Anwendungsfall wäre zum Beispiel das Thema Smart-Home. Ein Smart-Home verfügt über viele Sensoren, die innerhalb oder außerhalb eines Hauses installiert sind und Daten zur Verfügung stellen. Diese liefern den Bewohnern oder Eigentümern wichtige Informationen, zum Beispiel über die Temperaturen oder Energiewerte des Gebäudes.
Eine weitere Möglichkeit und auch als Usecase bei diesem Prototypen eingesetzt, können spezielle Anzüge mit Sensoren präperiert werden, welche Beschleunigungs-, Positions- und Orientierungsdaten übermitteln. Möglicher Einsatz sehen wir hier in der Videospiel- und Filmindustrie, sowie auch in der Wissenschaft und Medizin.

Kapitel 2

Spezifikationen

Java-Backend

- Entgegennahme von granularen Sensordaten mit unterschiedlichen Inhalten
- Aufbereitung der Sensordaten für spätere Speicherung und Nutzung aus der Datenbank
- Sinnvolle Speicherung der Daten in einer cloudfähigen Datenbank
- Bereitstellung einer REST-Schnittstelle[1] für den Zugriff auf Sensordaten
- Bereitstellung einer REST-Schnittstelle für eine Nutzerverwaltung
- Initialisierungsmechanismus für die Betriebnahme neuer Datenbanken

NodeJS-Frontend

- Bereitstellung einer Webschnittstelle mit grafischer Benutzeroberfläche
- Bereitstellung einer Nutzerverwaltung über die Webschnittstelle
- Bereitstellung einer Sensor-, Gateway-[2] und Clusterverwaltung[3] über die Webschnittstelle
- Bereitstellung einer Live-Übertragung von Daten zugeteilter Sensoren
- Visualisierung der verschiedenen Sensordatentypen im Live-Mitschnitt
- Bereitstellung eines Zugriffs auf gespeicherte Daten vergangener Daten-Mitschnitte

[1] Representational State Transfer
[2] Ein Gateway sammelt mehrere Daten eines Sensors in einem Bulk/Packet und sendet diese weiter
[3] Ein Cluster ist eine Gruppe von mehreren Sensoren

Messkriterien

- Skalierbarkeit

  Das System muss für eine unbegranzte Anzahl an Sensoren erweiterbar sein

  Der Datenverwaltung (Speicherung) dürfen keine Grenzen gesetzt sein

- Stabilität

  Das System muss alle gesendeten Daten ohne Verluste und Ausfälle verarbeiten können

  Granulare Datenmengen von mindestens 10000 Samples/sec müssen kontrollierbar sein

- Cloudfähigkeit

  Das System muss kompakt, vollständig und portabel auf weitere Cloud-Umgebungen installiert werden können

Out of Scope

- Keine direkte Interaktion mit den Sensoren
- Keine Server - Gateway Kommunikation
- Keine Webapplikation für mobile Geräte

Kapitel 3

Aufbau des Gesamtsystems

3.1 Infrastruktur

Die finale Infrastruktur besitzt in der Theorie genau Drei große Komponenten:

- Die Sensor-Umgebung mit den Gateways, welche die Daten liefern (links)
- Das Datenportal, was die Daten zur Verfügung stellt (rechts)
- Der Nutzer, der die Daten nutzt (unten)

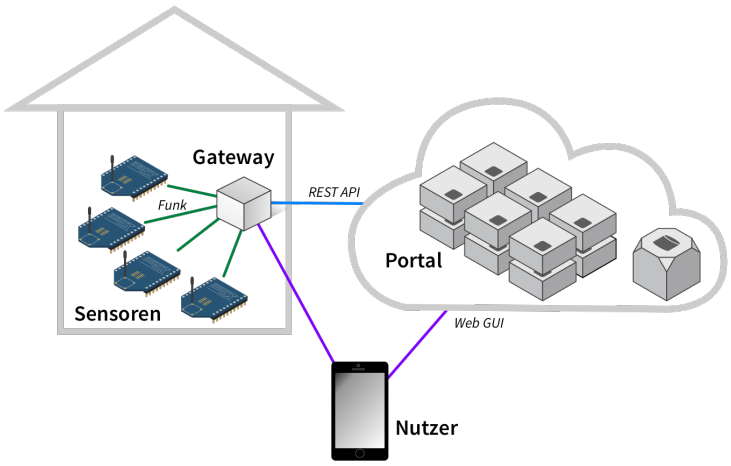


Abbildung 3.1:

Die nachfolgenden Kapitel werden zunächst alle Komponenten des Gesamtsystems beschreiben und dann detaillierter auf die einzelnen Komponenten des Datenportals eingehen.

3.1.1 Gesamtsystem

Das Gesamtsystem lässt sich, wie zuvor schon beschrieben, zusammenfassend in Drei Komponenten gliedern: Sensor-Umgebung, Datenportal und Nutzer.

Beginnend beim Aggregieren von Daten, stehen die **Sensoren** an erster Stelle des Gesamtsystems. Unterscheidbar in Sensoren, welche nur eine einzige Art von Daten messen können, wie zum Beispiel die Temperatur oder Sensoren, die verschiedene Arten von Daten unterscheiden. In diesem Projekt werden Vier verschiedene Typen von Daten differenziert:

- Temperatur (einzelner Integer Wert)
- Lokalität (String)
- Orientierung (orientation quaternions[1] - Liste mit Dezimalzahlen)
- Beschleunigung (Dreier-Vektor - Liste mit Dezimalzahlen)

An einem Anzug oder an Gegenständen befestigt und mit einem sogenannten **Gateway**[2] verbunden, senden die Sensoren ihre Daten an den Message-Broker, in diesem Fall ein RabbitMQ-Server. Die Kommunikation zwischen Sensor und Gateway geschieht über ZigBee und Wifi.

Die Gateways aggregieren mehrere Daten eines Sensors zu einem Bulk[3] zusammen und kommunzieren mit dem **Message-Broker** über das AMQP[4]. Pro Sensor wird vom RabbitMQ-Server ein einzigartiges Topic angelegt. Jedes Topic wird veröffentlicht und kann beliebig von jedem Nutzer im Netzwerk abbonniert werden und alle Daten dieses Topics erhalten. Für den Einsatz im Internet of Things Umfeld eignet sich ein derartiges Publish-Subscribe-Konzept ideal.

Neben der Funktion als Motor für die Datenverteilung zu fungieren, kann man den Message-Broker als Bindeglied zwischen *Sensor-Umgebung* und *Daten-Portal* betrachten. Das sequentiell nächste Bestandteil ist die **Java-Applikation**, welche in einem Tomcat-Applikationsserver gestartet wird. Sie abbonniert jedes Topic und speichert alle Sensordaten aufbereitet in die **DynamoDB NoSQL-Cloud-Datenbank**. Des weiteren stellt sie **REST-Schnittstellen** zur Verfügung, um auf gespeicherte Daten, über eine Webapplikation oder anderer externer Systeme, zuzugreifen. Neben Schnittstellen für den Datenzugriff werden ebenfalls Funktionen für Nutzer- und Entitäsverwaltung bereitgestellt. Somit hat der Nutzer, je nachdem welche Rechte er besitzt, Einfluss auf alle Entitäten und deren Informationen. Mehr dazu im Kapitel *3.1.2.2 Schnittstelle zur Datenverarbeitung*.

[1] Vierer-Vektor für die Beschreibung von Orientierung
[2] Kleinrechner wie zum Beispiel Raspberry Pi oder Handy
[3] Gruppierung von Daten - wie ein Array
[4] Advanced Message Queuing Protocol

Zu guter letzt finden wir in der Daten-Portal Komponente eine NodeJS-Applikation, welche den Webauftritt und die letztendliche Interaktion mit dem System zur Verfügung stelllt. Genau wie auch die Java-Applikation abbonniert sie die Topics des Message-Brokers und ermöglicht dem Nutzer so einen Live-Mittschnitt der Sensordaten. Ebenfalls werden die REST-Schnittstellen benutzt um mittels einer GUI die Entitäten zu verwalten.

Um den Messkriterien genüge zu werden, trifft das Daten-Portal Vorbereitungen, mit modernen Technologien, um eventuellen Skalierungen des Systems gewappnet zu sein. Die Software **Docker** verpackt alle benötigten Applikationen, wie die Java-Applikation, den Message-Broker und die NodeJS-Applikation mit allen benötigten Abhängigkeiten in sogenannte Container, welche praktikabel transportiert werden können. Mittels der Automatisierungs-Software **Puppet** können mit diesen Containern und programmierten Skripten schließlich ohne große Mühe alle benötigten Module für das Daten-Portal auf verschiedene, unabhängige Server-Umgebungen installiert werden.

Folgende Technologien wurden für die einzelnen Komponenten verwendet:

Komponente	**Technologien**
Java-Applikation	Spring Gradle HTTP/REST Hibernate
Frontend-Applikation	NodeJS D3
Datenbank	AWS DynamoDB
Message-Broker	RabbitMQ AMQP
Provisioning	Docker Puppet

Um einen verständlicheren Überblick zu schaffen, zeigt das folgende Big Picture detailliert die gesamte Infrastruktur, mit allen Elementen und deren Kommunikationstechnologien. Darauf folgende Unterkapitel gehen näher auf die einzelnen Komponenten ein.

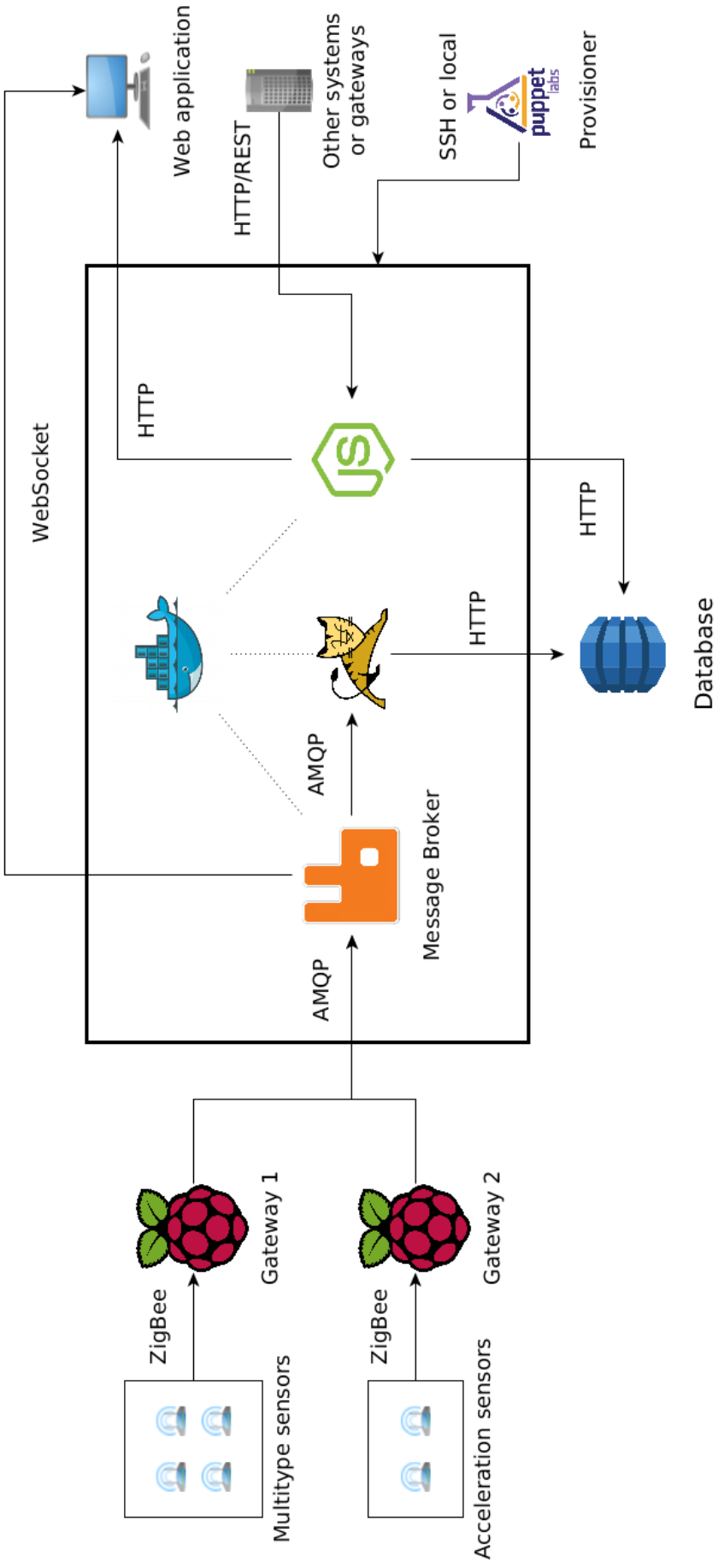


Abbildung 3.2: Gesamtarchitektur

3.1.2 Java-Backend

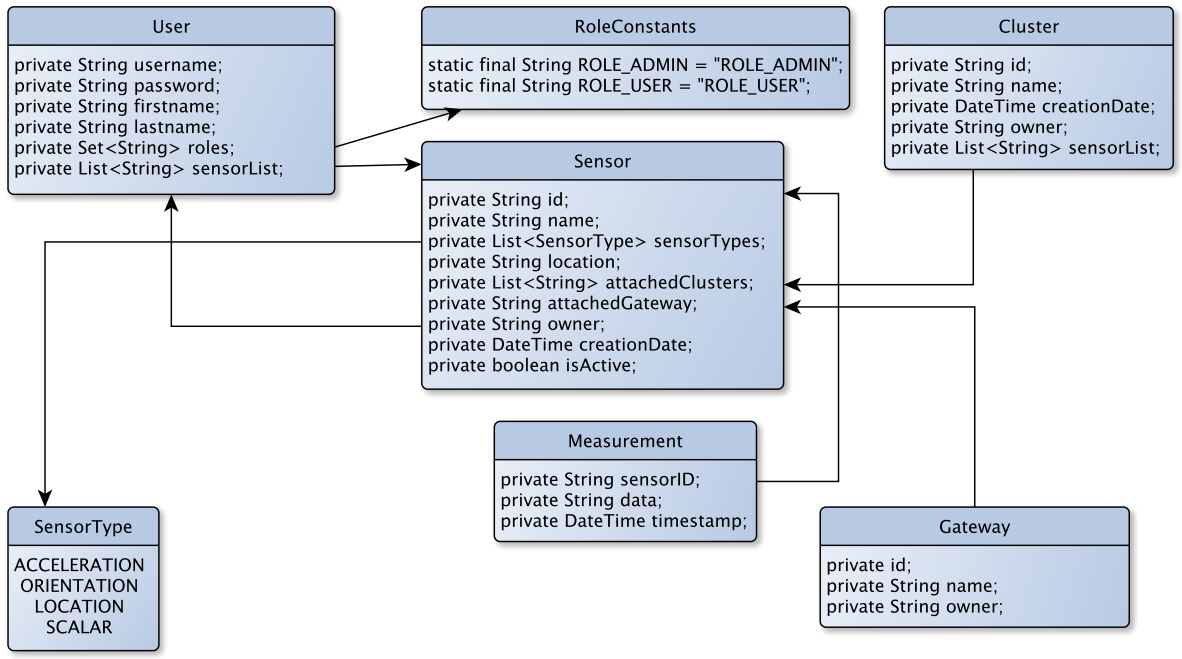
Die in einem Tomcat-Applikationsserver laufende Java-Applikation hat die folgenden Zwei Schnittstellen zu liefern:

3.1.2.1 Schnittstelle zur Datenverarbeitung

Die Schnittstelle zur Datenverarbeitung nimmt die Daten der Sensoren entgegen, bereitet sie zur Nutzung auf und speichert sie in der DynamoDB-Datenbank ab. Die Entgegennahme der Daten wird über das Topic-Abbonnement ermöglicht. Dabei treten folgende Schwierigkeiten auf:

- Sensoren/Topics müssen bekannt und in der Datenbank hinterlegt sein
- Datenmodell muss klar definiert sein
- Eingehende Bulks beinhalten nur Daten von einem einzigen Sensor

Das folgende Datenmodell veranschaulicht die Struktur und die gegeben Entitäten im Java-Backend.



Neben den oben genannten Aufgaben, wird ebenfalls ein Initialisierungs-Mechanismus zur Verfügung gestellt, der die Tabellen, falls noch nicht vorhanden, in der verknüpften Datenbank erstellt und sie mit Beispieldaten füllt.

User

Bildet die **Nutzer** des Daten-Portals ab und ermöglicht eine rollenspezifische Nutzer- und Entitätsverwaltung. (Erzeugung, Bearbeiten, Löschen von Clustern, Sensoren und Gateways) Die *sensorList* mit den zugeteilten Sensornamen wird benutzt, um einen einfachen Zugriff auf sensorspezifische Daten zu haben. Der *username* einer Nutzers ist im gesamten jeweiligen Datenbank-Umfeld einzigartig.

RoleConstants

Bildet die **Rollen** im Java-Backend ab. Sie ermöglichen es, verschiedene Nutzer zu unterscheiden und deren Rechte einzugrenzen oder zu erweitern. Wobei der *ADMIN* alle Rechte besitzt und der *User* nur Rechte für die von ihm selbst erzeugten Entitäten.

Sensor

Die Sensor-Entität bildet die datenerzeugende Hardware in der Java-Applikation ab. Eine Liste mit den Sensortypen vereinfacht die Aufbereitung der Daten. Einem Sensor können mehrere Cluster und genau ein Gateway zugeteilt werden. Der *owner* besitzt den *username* des Nutzers, welcher ihn erzeugt hat. Der boolean Wert *isActive* ermöglicht es, Sensoren zu de- oder aktivieren.

SensorType

Bildet alle bekannten **Typen** der Sensordaten ab, die über das NodeJS-Frontend visualisiert werden können. Wenn ein Typ nicht bekannt ist, wird es ohne Bearbeitung in der DynamoDB abgespeichert.

Cluster

Bilden eine Gruppe oder auch Verbund von mehreren Sensoren ab. Der *owner* ist dabei der Erzeuger des Clusters und die *sensorList* beinhaltet alle zugeteilten Sensoren. Es gibt keine maximale Obergrenze an zugeteilten Sensoren pro Cluster.

Gateway

Ein Gateway bildet die Hardware ab, welches die Sensordaten eines einzelnen Sensors zunächst in einem Bulk aggregiert und dann an den Message-Broker übermittelt. Jeder Sensor muss über ein Gateway die Daten versenden. Ein Gateway besitzt eine ID, einen Namen und einen Ersteller(*owner*). Es gibt keine maximale Obergrenze an zugeteilten Sensoren pro Gateway.

Measurement

Das Measurement repräsentiert die vom Sensor versendeten **Daten** an sich. Sie werden pro Bulk mit einem timestamp und der dazugehörigen Sensor-ID gespeichert.

3.1.2.2 DynamoDB-Key Schema

Die nachfolgende Tabelle beinhaltet alle Keys der DynamoDB-Entiäten. Zur Information: die HASH-Keys sind vergleichbar mit den Primary-Keys einer relationalen Datenbank und werden benutzt, um die jeweiligen Entitäten aus der Datenbank zu lesen. Mit den RANGE-Keys können die Daten in der Datenbank sortiert gespeichert werden.

Entität	HASH KEY	RANGE KEY	TYPE
User	username	-	String
Sensor	id	-	String
Gateway	id	-	String
Cluster	id	-	String
Bulk	sensorId	bulkReceived	String

3.1.2.3 REST-Schnittstelle

Für die Kommunikation nach außen wird mittels REST[5]-Konventionen eine Schnittstelle bereit gestellt, die neben Nutzerverwaltungs-Services, ebenfalls CRUD[6]-Methoden für alle weiteren Entitäten bereitstellt und über HTTP Aufrufe genutzt werden kann. Somit ist gegeben, Sensordaten mittels weiterer Software zu schöpfen, wie zum Beispiel die NodeJS-Applikation, oder ein direkter Zugriff über ein Terminal.

Welche Webservices für die jeweiligen Entitäten zur Verfügung stehen, wie sie aufgerufen und benutzt werden, wird in der API-Dokumentation im Anhang genau beschrieben. Genutzt wurde hier das Google Chrome Tool Postman.

3.1.3 NodeJS-Frontend

Das Frontend bietet dem User eine grafische Benutzeroberfläche, um mit dem Java-Backend zu interagieren. Dabei wird hauptsächlich mit der bereitgestellten REST-API interagiert, etwa um die Daten von Sensoren anzuzeigen. Um Sensordaten in Echtzeit anzuzeigen, wird mittels dem WebSocket-Protokoll eine direkte Verbindung mit dem Message Broker Rabbit aufgebaut, über der die Sensordaten mit geringst möglicher Latenz empfangen werden können.

[5]Representational State Transfer
[6]Create,Read,Update,Delete

3.1.4 Daten-Generator

Der Daten-Generator wurde entwickelt, um die noch nicht fertig gestellte Sensor-Umgebung (Sensoren und Gateways) und die damit einhergehende Generierung von Sensordaten zu simulieren. Dabei wurden mit 7 Sensoren 3 Minuten Daten in eine .DAT Datei geschrieben und diese mit einem Python Skript weiterverarbeitet. Während dieses Schrittes wurden die Daten in Bulks geschrieben und mit Hilfe des Message-Brokers(RabbitMQ-Servers) auf diverse Topics verteilt, welche wiederum vom Java-Backend abbonniert werden können. Im Anhang zeigt sich ein kleiner Auszug aus diesen generierten Daten in JSON-Form.

3.1.5 Provisioner

Um eine schnelle und mit geringem Aufwand verbundene Installation des Komponenten-Kollektivs und all ihren Abhängigkeiten zu gewährleisten, werden Zwei Technologien eingesetzt die dafür prädestiniert sind: *Docker* und *Puppet*

3.1.5.1 Docker

Docker verbindet die Eigenschaft, Software mit all ihren Abhängigkeiten kompakt in einen sogenannten Container zu packen, mit dem Vorteil, sie ebenfalls leicht transportierbar und installierbar zu handhaben.

3.1.5.2 Puppet

Puppet[7] ist ein Open-Source-Tool zum Konfigurationsmanagement von Computern via Netzwerk und wird innerhalb unseres Projektes zur automatischen Konfiguration des Produktivsystems genutzt. Dazu würde ein sog. Puppet-Modul[8] erstellt, das folgende Aufgaben übernimmt:

1. Installation der benötigen Software
2. Erstellung eines Systembenutzers
3. Klonen des gesamten Projektes von Github

[7] https://puppetlabs.com/
[8] https://forge.puppetlabs.com/maasch/iot_provisioning/

Kapitel 4

Qualitätssicherung

4.1 REST-Schnittstelle

Mittels des Google Chrome Tools *Postman* wurden alle Aufrufe der Schnittstelle auf ihre Funktionsfähigkeit getestet. Test-Aufrufe wurden im Kollektiv gespeichert und für Testzwecke im Repository hinterlegt:
`https://github.com/TomWieschalla/Masterprojekt-WS15-16-MMI-IoT/tree/master/iot-backend/docs/Postman-collections`

4.2 NodeJS-Frontend

Für das Frontend wurden Usability-Tests durchgeführt, die auf der einen Seite die Benutzerfreundlichkeit und auf der anderen Seite alle Funktionen auf ihre Funktionstüchtigkeit testen sollten.

4.3 Performance

Um die Stabilität und die Performance zu testen, wurden Lasttests durchgeführt. Das Ziel, 10000 Samples/Sec zu verarbeiten und gleichzeitig live auf der Weboberfläche Sensordaten anzuzeigen, wurde erfolgreich getestet.

Kapitel 5

Installationsbeschreibung

Die Anleitungen zur Installation bzw. zur Inbetriebnahme des Gesamtsystems befinden sich in den README-Dateien des Projekt-Repositories (Github) und sind unter folgenden URL's verfügbar:

Entwicklung

`https://github.com/TomWieschalla/Masterprojekt-WS15-16-MMI-IoT/tree/master/iot-backend`

Produktion:

`https://github.com/TomWieschalla/Masterprojekt-WS15-16-MMI-IoT`

Kapitel 6

Anhang

REST-API-DOKUMENTATION

IOT USER CRUD

GET USER

GET `http://localhost:8080/iot-friss/user/max`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/user/max';
```

UPDATE USER

PUT `http://localhost:8080/iot-friss/user/max`

Headers `Content-Type: application/json Authorization: Basic bWF4OnRlc3QxMjM=`

Raw data
```
{ "username": "max", "password": "test123", "firstname": "klaus", "lastname": "mustermann", "roles": [ "ROLE_USER" ], "releasedForSensors": [], "releasedForGateways": [], "releasedForClusters": [] }
```

Curl
```
curl -X PUT -H -H "Content-Type: application/json" -H "Authorization: Basic bWF4OnRlc3QxMjM=" -d ' { "username": "max", "password": "test123", "firstname": "klaus", "lastname": "mustermann", "roles": [ "ROLE_USER" ], "releasedForSensors": [], "releasedForGateways": [], "releasedForClusters": [] }' 'http://localhost:8080/iot-friss/user/max';
```

GET ALL

GET `http://localhost:8080/iot-friss/user`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/user';
```

DELETE USER

DELETE `http://localhost:8080/iot-friss/user/max`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl

```
curl -X DELETE -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/user/max';
```

CREATE USER

POST `http://localhost:8080/iot-friss/user`

Headers

```
Content-Type: application/json Authorization: Basic YWRtaW46YWRtaW4=
```

Raw data

```
{ "username":"max1", "password":"test123", "firstname":"max",
"lastname":"mustermann", "roles": ["ROLE_USER"] }
```

Curl

```
curl -X POST -H -H "Content-Type: application/json" -H "Authorization: Basic
YWRtaW46YWRtaW4=" -d '{ "username":"max1", "password":"test123",
"firstname":"max", "lastname":"mustermann", "roles": ["ROLE_USER"] }'
'http://localhost:8080/iot-friss/user';
```

IOT SENSOR CRUD

CREATE SENSOR

POST `http://localhost:8080/iot-friss/sensor`

Headers `Content-Type: application/json Authorization: Basic bWF4OnRlc3QxMjM=`

Raw data

```
{ "name": "Rücken7", "types": [ "acceleration", "orientation" ], "location": "Berlin, Germany", "attachedCluster": "d65776c7-3514-4c3e-a320-00ba5b0b72cd", "attachedGateway": "914e6e00-8cdd-428e-b418-bee0b248a34c", "owner": "max", "creationDate": "2016-01-31T20:13:00.000Z", "isActive": true }
```

Curl

```
curl -X POST -H -H "Content-Type: application/json" -H "Authorization: Basic bWF4OnRlc3QxMjM=" -d ' { "name": "Rücken7", "types": [ "acceleration", "orientation" ], "location": "Berlin, Germany", "attachedCluster": "d65776c7-3514-4c3e-a320-00ba5b0b72cd", "attachedGateway": "914e6e00-8cdd-428e-b418-bee0b248a34c", "owner": "max", "creationDate": "2016-01-31T20:13:00.000Z", "isActive": true }' 'http://localhost:8080/iot-friss/sensor';
```

UPDATE SENSOR

PUT `http://localhost:8080/iot-friss/sensor/1fd29eab-a01a-4d0b-82b8-46fe0b5126f7`

Headers `Content-Type: application/json Authorization: Basic YWRtaW46YWRtaW4=`

Raw data

```
{ "name": "Rücken5", "types": [ "acceleration", "orientation" ], "location": "Berlin, Germany", "attachedCluster": "d65776c7-3514-4c3e-a320-00ba5b0b72cd", "attachedGateway": "914e6e00-8cdd-428e-b418-bee0b248a34c", "owner": "max", "creationDate": "2016-01-31T20:13:00.000Z", "isActive": true }
```

Curl

```
curl -X PUT -H -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ' { "name": "Rücken5", "types": [ "acceleration", "orientation" ], "location": "Berlin, Germany", "attachedCluster": "d65776c7-3514-4c3e-a320-00ba5b0b72cd", "attachedGateway": "914e6e00-8cdd-428e-b418-bee0b248a34c", "owner": "max", "creationDate": "2016-01-31T20:13:00.000Z", "isActive": true }' 'http://localhost:8080/iot-friss/sensor/1fd29eab-a01a-4d0b-82b8-46fe0b5126f7';
```

GET ALL

GET `http://localhost:8080/iot-friss/sensor?owner=max`

Param "owner" not required

Headers `Authorization: Basic bWF4OnRlc3QxMjM=`

Curl
```
curl -X GET -H "Authorization: Basic bWF4OnRlc3QxMjM=" -d '' 'http://localhost:8080/iot-friss/sensor?owner=max';
```

DELETE SENSOR

DELETE `http://localhost:8080/iot-friss/sensor/1085d1c6-18e5-4345-b135-3d5785ae8a95`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X DELETE -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/sensor/1085d1c6-18e5-4345-b135-3d5785ae8a95';
```

GET SENSOR

GET `http://localhost:8080/iot-friss/sensor/ed7c7559-0cf1-488e-8023-695b9d8d9729`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/sensor/ed7c7559-0cf1-488e-8023-695b9d8d9729';
```

IOT CLUSTER CRUD

GET SENSORS FROM CLUSTER

GET `http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd/sensor`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl

```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd/sensor';
```

GET ALL

GET `http://localhost:8080/iot-friss/cluster?owner=max`

Param "owner" not required

Headers `Authorization: Basic bWF4OnRlc3QxMjM=`

Curl

```
curl -X GET -H "Authorization: Basic bWF4OnRlc3QxMjM=" -d '' 'http://localhost:8080/iot-friss/cluster?owner=max';
```

GET CLUSTER

GET `http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl

```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd';
```

UPDATE CLUSTER

PUT `http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd`

Headers `Content-Type: application/json Authorization: Basic YWRtaW46YWRtaW4=`

Raw data

```
{ "name":"cluster1", "owner": "max", "creationDate": "2016-01-31T20:00:40.000Z" }
```

Curl
```
curl -X PUT -H -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '{ "name":"cluster1", "owner": "max", "creationDate": "2016-01-31T20:00:40.000Z" }' 'http://localhost:8080/iot-friss/cluster/d65776c7-3514-4c3e-a320-00ba5b0b72cd';
```

CREATE CLUSTER

POST `http://localhost:8080/iot-friss/cluster`

Headers `Content-Type: application/json Authorization: Basic YWRtaW46YWRtaW4=`

Raw data
```
{ "name":"cluster2", "owner":"max", "creationDate": "2016-01-31T20:00:40.000Z" }
```

Curl
```
curl -X POST -H -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '{ "name":"cluster2", "owner":"max", "creationDate": "2016-01-31T20:00:40.000Z" } ' 'http://localhost:8080/iot-friss/cluster';
```

DELETE CLUSTER

DELET `http://localhost:8080/iot-friss/cluster/92b73cb2-2c04-4a4c-9eb0-7ac6fc9217ae`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X DELETE -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/cluster/92b73cb2-2c04-4a4c-9eb0-7ac6fc9217ae';
```

IOT GATEWAY CRUD

UPDATE GATEWAY

PUT `http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c`

Headers `Content-Type: application/json`

Raw data `{ "name":"TestNameFürGateway", "owner": "admin" }`

Curl
```
curl -X PUT -H -H "Content-Type: application/json" -d '{ "name":"TestNameFürGateway", "owner": "admin" }' 'http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c';
```

GET SENSORS FROM GATEWAY

GET `http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c/sensor`

Headers `Content-Type: application/json Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c/sensor';
```

GET GATEWAY

GET `http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d '' 'http://localhost:8080/iot-friss/gateway/914e6e00-8cdd-428e-b418-bee0b248a34c';
```

DELETE GATEWAY

DELETE `http://localhost:8080/iot-friss/gateway/a14a7b43-6f9a-4524-9603-256d695ae6f8`

Curl
```
curl -X DELETE -d '' 'http://localhost:8080/iot-friss/gateway/a14a7b43-
```

```
6f9a-4524-9603-256d695ae6f8';
```

CREATE GATEWAY

POST `http://localhost:8080/iot-friss/gateway`

Headers
```
Content-Type: application/json
```

Raw data
```
{ "name":"Gateway2", "owner":"max" }
```

Curl
```
curl -X POST -H -H "Content-Type: application/json" -d '{
"name":"Gateway2", "owner":"max" } ' 'http://localhost:8080/iot-
friss/gateway';
```

GET ALL

GET `http://localhost:8080/iot-friss/gateway?owner=max`

Param "owner" not required

Headers
```
Authorization: Basic YWRtaW46YWRtaW4=
```

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/gateway?owner=max';
```

IOT CACHE

CLEAR ALL CACHES

DELETE `http://localhost:8080/iot-friss/cache`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X DELETE -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/cache';
```

CLEAR CACHE

DELETE `http://localhost:8080/iot-friss/cache/sensor-active-cache`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X DELETE -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/cache/sensor-active-cache';
```

GET ALL CACHE NAMES

GET `http://localhost:8080/iot-friss/cache`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/cache';
```

IOT ROLE

GET AVAILABLE ROLES

GET `http://localhost:8080/iot-friss/role`

Headers `Authorization: Basic YWRtaW46YWRtaW4=`

Curl
```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -d ''
'http://localhost:8080/iot-friss/role';
```

IOT BULK

GET BULK

GET `http://localhost:8080/iot-friss/bulk?sensor_id=a833519d-3dfb-4746-8ec0-8bb66d80817f&start=2016-01-25T18:30:48.444Z&end=2016-01-25T18:30:48.444Z&page=0&page_size=50&sort_order=asc`

Parameters 'start' and 'end' are required in UTC

Headers

```
Authorization: Basic bWF4MTp0ZXN0MTIz
```

Curl

```
curl -X GET -H "Authorization: Basic bWF4MTp0ZXN0MTIz" -d '' 'http://localhost:8080/iot-friss/bulk?sensor_id=a833519d-3dfb-4746-8ec0-8bb66d80817f&start=2016-01-25T18:30:48.444Z&end=2016-01-25T18:30:48.444Z&page=0&page_size=50&sort_order=asc';
```

Beispiel Sensordaten

Tabelle 6.1: Schema eines Samples innerhalb des JSON Files

Bezeichner	Beschreibung des Inhalts
time	ISO timestamp
id	Eindeutige Sensor uuid
acceleration	[float, float, float]
location	Textuelle Beschreibung oder GPS Koordinaten
orientation	[float, float, float, float]

Tabelle 6.2: Sample Beispiel Schulter

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	26518222-ea4d-4318-8b97-19e23bf1d090
acceleration	[0.03, 0.02, -0.09]
location	Europe/Berlin
orientation	[0.276001, -0.959656, 0.053711, -0.003357]

Tabelle 6.3: Sample Beispiel BeinLinks

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	ed7c7559-0cf1-488e-8023-695b9d8d9729
acceleration	[0.0, -0.01, 0.02]
location	Europe/Berlin
orientation	[0.279358, 0.799377, -0.531921, 0.000488]

Tabelle 6.4: Sample Beispiel BeinRechts

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	aeeae543-af7e-43af-8329-9910d087c6a8
acceleration	[-0.01, -0.02, -0.05]
location	Europe/Berlin
orientation	[0.43457, 0.847351, 0.305176, 0.000427]

Tabelle 6.5: Sample Beispiel Ruecken

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	a833519d-3dfb-4746-8ec0-8bb66d80817f
acceleration	[-0.02, 0.02, 0.0]
location	Europe/Berlin
orientation	[0.777954, -0.627869, -0.022217, 0.007751]

Tabelle 6.6: Sample Beispiel Kopf

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	1085d1c6-18e5-4345-b135-3d5785ae8a95
acceleration	[-0.03, -0.01, 0.0]
location	Europe/Berlin
orientation	[0.98645, 0.112427, 0.119324, -0.003479]

Tabelle 6.7: Sample Beispiel ArmRechts

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	146acbde-e555-48f1-83a3-e23b1ea618ce
acceleration	[-0.25, -0.13, 0.18]
location	Europe/Berlin
orientation	[0.126343, -0.979004, -0.04657, 0.153137]

Tabelle 6.8: Sample Beispiel ArmLinks

Bezeichner	Inhalt des JSON Feldes
time	2016-01-18 18:00:04.288000 CET
id	64b8429c-d27e-4b60-84f8-fb62c0c41232
acceleration	[0.01, 0.0, 0.06]
location	Europe/Berlin
orientation	[0.846252, -0.164917, -0.505005, -0.040405]