Deckblatt noch machen

Inhalt

[Theoretischer Teil: Aufgabe 1 Domain Driven Design 3](#_Toc121878346)

[Theoretischer Teil: Aufgabe 2 Datenmodel 4](#_Toc121878347)

[Theoretischer Teil: Aufgabe 3 Eventliste 5](#_Toc121878348)

[Theoretischer Teil: Aufgabe 4 Deployment Modell der Microservices 6](#_Toc121878349)

[Praktischer Teil: Beschreibung des MVPs 7](#_Toc121878350)

[Sim: 8](#_Toc121878351)

[MongoDB 8](#_Toc121878352)

[MQTT 9](#_Toc121878353)

[Rectemp 10](#_Toc121878354)

[Config.js 11](#_Toc121878355)

[Wertebereich/ Gültigkeitsbereich der Sensoren: 11](#_Toc121878356)

[MQTT-CLIENTID: 12](#_Toc121878357)

[Schreiben in MongoDB: 12](#_Toc121878358)

[Sende Warnung bei Überschreiten von Schwellwert: 13](#_Toc121878359)

[Zusätzliche Features: 14](#_Toc121878360)

[Healthz-Schnittstelle: 16](#_Toc121878361)

[Temp-alert 16](#_Toc121878362)

[Schaubild MQTT Messages: 17](#_Toc121878363)

[Praktischer Teil: Kubernetes 17](#_Toc121878364)

[Netzwerk 17](#_Toc121878365)

[Konfigurierbare Verbindungsparameter für den Zugriff auf MongoDB: 17](#_Toc121878366)

[Resilienz gegen Ausfall 18](#_Toc121878367)

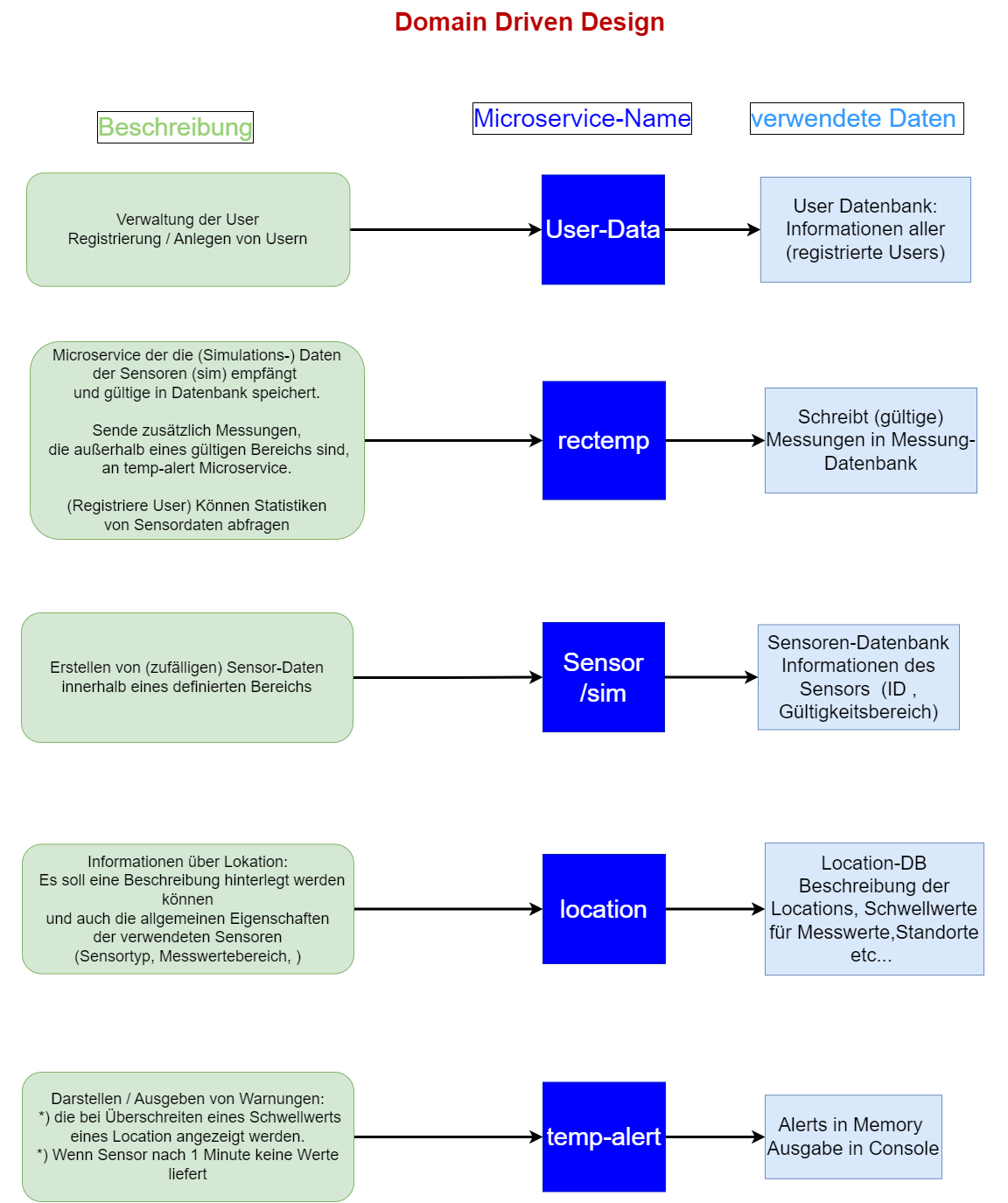
[Quota Limit für rectemp 18](#_Toc121878368)

[Integrationstest / (Installations-Anleitung) 19](#_Toc121878369)

[Lokaler Bau der Container mit docker-compose 19](#_Toc121878370)

[Deployment in Kubernetes 22](#_Toc121878371)

# Theoretischer Teil: Aufgabe 1 Domain Driven Design



|  |  |
| --- | --- |
| Sensor/SIM | |
| Name | Type |
| Sensorid | Integer |
| Messung | double |
| Datum | DateTime |
| Messungstyp | Char |
| Gueltig\_ab | Integer |
| Gueltig\_bis | Integer |
| *Locationid* | *Integer  (zeigt auf location)* |

# Theoretischer Teil: Aufgabe 2 Datenmodel

|  |  |
| --- | --- |
| rectemp | |
| Name | Type |
| *Sensorid* | Integer  (zeigt auf Sensor/SIM) |
| *Messung* | double (zeigt auf Sensor/SIM) |
| *Datum* | DateTime  (zeigt auf Sensor/SIM) |
| *Messungstyp* | Char  (zeigt auf Sensor/SIM) |
| *Sensorid* | Integer  (zeigt auf Sensor/SIM) |
| *Locationid* | String  *(zeigt auf location)* |
| *gpslatitude* | String  *(zeigt auf location)* |
| *gpslongitude* | String  *(zeigt auf location)* |
| *Beschreibung* | String  *(zeigt auf location)* |
| *Schwellwert\_unten* | Array/Map *(zeigt auf location)* |
| *Schwellwert\_oben* | Array/Map *(zeigt auf location)* |

|  |  |
| --- | --- |
| location | |
| Name | Type |
| Locationid | Integer |
| gpslatitude | String |
| gpslongitude | String |
| Beschreibung | String |
| Schwellwert\_unten\* | Array/Map definiert unteren Schwellwert pro Messungstyp |
| Schwellwert\_oben\* | Array/Map definiert oberen Schwellwert pro Messungstyp |

|  |  |
| --- | --- |
| User | |
| Name | Type |
| Username | String |
| Email | String (E-Mail) |
| Password | String (Hashed) |

|  |  |
| --- | --- |
| temp-alert | |
| Name | Type |
| *Sensorid* | Integer  (zeigt auf Sensor/SIM) |
| *Messung* | double (zeigt auf Sensor/SIM) |
| *Datum* | DateTime  (zeigt auf Sensor/SIM) |
| *Messungstyp* | Char  (zeigt auf Sensor/SIM) |
| *Locationid* | String  (zeigt auf rectemp) |
| *gpslatitude* | String  (zeigt auf rectemp) |
| *gpslongitude* | String  (zeigt auf rectemp) |
| *Beschreibung* | String  (zeigt auf rectemp) |

Anmerkung\*:  
Variablen bzw. Werte für *Schwellwert\_unten* und *Schwellwert\_oben* werden pro Location gesetzt, da angenommen wird, dass für jeden Standort unterschiedliche Schwellwerte (pro Sensortyp) definiert werden können sollen. Bspw. Soll in einem Raum mehr Personen erlaubt sein als in einem anderem.

# Theoretischer Teil: Aufgabe 3 Eventliste

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entity type | Event type | Event Topic | Event Data | Comments |
| User | User\_created | User/created | Username,Email, Passwort, |  |
| User | User\_updated | User/updated | Username,Email, Passwort, |  |
| User | User\_deleted | User/deleted | Username |  |
| location | Location\_created | Location/created | Locationid,gpslatitude,gpslongitude,Beschreibung,Schwellwert\_unten,Schwellwert\_oben | Eine neue Location wurde erstellt |
| location | Location\_updated | Location/updated | Locationid,gpslatitude,gpslongitude,Beschreibung,Schwellwert\_unten,Schwellwert\_oben | Eine Location wurde geändert |
| location | Location\_deleted | Location-deleted | Locationid | Eine Location wurde geöscht |
| Sensor/SIM | sende\_gültige\_messung | rec/messungen | JSON:{messungsdaten} | Gültige Messung wird gesendet |
| Sensor/SIM | sende\_ungültige\_messung | Alert/ungueltig | JSON:{messungsdaten} | Ungültige Messung außerhalb des Gültigkeitsbereichs des Sensors wird an *Alert- ungültig* gesendet |
| rectemp | sende\_messung\_warnung | Alert/Schwellwert |  | Messung außerhalb des Schwellwerts wird als Warnung gesendet |

Anmerkung:

temp-alert abonniert die Topics: Alert/ungueltig und Alert/Schwellwert  
dadurch kann unterschieden werden, ob eine Messung bereits bei der Erstellung am Sensor ungültig ist, oder die (gültige Messung) im nicht akzeptieren Bereich liegt (durch rectemp gesendet)

Sensor/SIM abonniert die Topics: Location/created und Location/updated und Location/delete,  
sodass beim Senden der Messungen die aktuellen Daten/Informationen der Locations verwendet werden (sind auch lokal zwischengespeichert)

rectemp abonniert das Topic: : rec/messungen. Hier kommen die güligen Messungen des Sensors/SIM an

fwe

weffwe

# Theoretischer Teil: Aufgabe 4 Deployment Modell der Microservices

Deployment Model für theoretischen Teil wie er so in Docker/Kubernetes Deployt werden kann:

Ein Bild, das Text enthält.

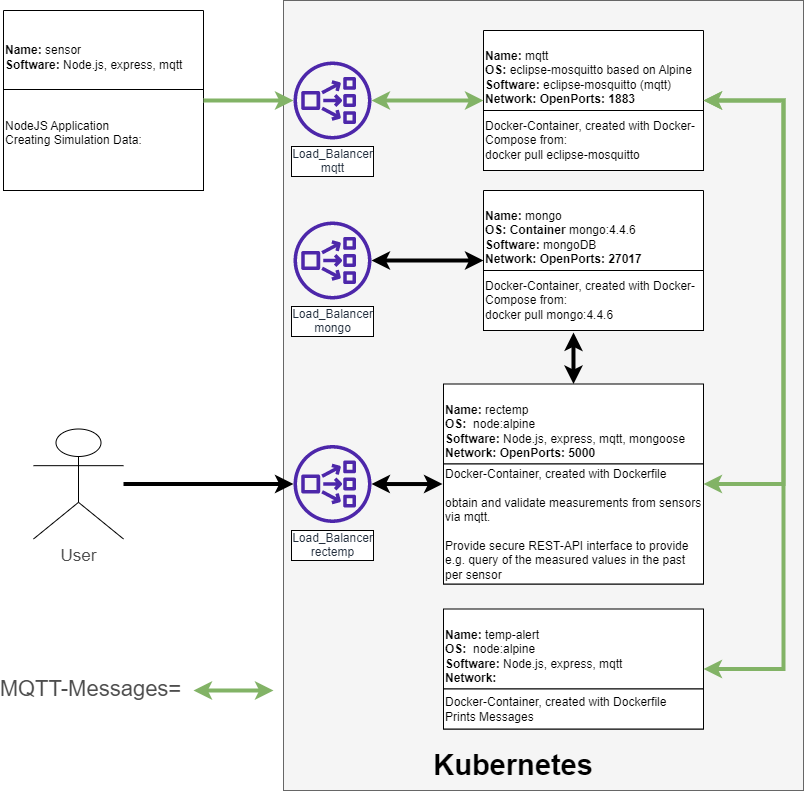
Automatisch generierte Beschreibung

# Praktischer Teil: Beschreibung des MVPs

Der MVP bildet eine abgespeckte Funktionalität unseres Messdatenerfassungssystems ab.

Anders als beispielsweiße im Theorieteil gibt es in diesem MVP erstmal kein eigenen Microservice zur Userverwaltung. Auch ist die Verwaltung der Locations durch einen eigenen Microservice so auch noch nicht abgebildet:

Das Deployment-Model des MVPs kann mit folgendem Schaubild gezeigt werden:



Der MVP-Besteht aus folgenden **Komponenten**:

## Sim:

Das Sim-Programm ist ein NodeJS Programm das Messungen eines Sensors von einem angegeben Standort simuliert.

Dabei müssen beim Start des Programms bestimmte Parameter definiert werden:

* + arg1 = timeinterval in dem die Daten gesendet werden in sec
  + arg1 = unique id of the sensor station - 4 stellig
  + arg2 = Anzahl der Daten in Zyklen - 2stellig'
  + arg3 = Sensortyp - T=temperature, X=co2, P=people in a room, H=luftfeuch
  + arg4 = start value
  + arg5 = end value

npm start 10 1234 5 T 20 25  
 🡪 alle 10 Sekunden, werden vom Standort 1234, 5 mal, Temperaturdaten, **zwischen** 20 und 25 generiert

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**Der generierte Messwert wird im Bereich der letzten zwei Argumente zufällig pro Iteration generiert

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDiese Messungen werden über MQTT jeweils an das Topic .  
mqttopic="4934001/messungen" gesendet.  
Dabei ist wichtig die korrekte ULR des MQTT-Host bestimmen. Diese ist in Zeile 12 des NodeJS Scripts zusetzen:  
(Bei Verwendung von Docker-Desktop mit aktiviertem Kubernetes ist diese angezeigte URL so korrekt

Aufgabe-5/sim/index-skeleton.js

Aufgabe-5/sim/index-skeleton.js

## Ein Bild, das Text enthält. Automatisch generierte BeschreibungMongoDB

Innerhalb von Kubernetes wird ein einfache MongoDB Deployed werden.   
Der Username und das Passwort werden entsprechend in der docker-compose Datei hinterlegt.

Aufgabe-8\mongodb\mongodb-deployment.yaml"

Ein Bild, das Text, Anzeigetafel enthält.

Automatisch generierte BeschreibungDamit die Daten innerhalb von Kubernetes persistiert werden (also bei Neustart des Containers nicht gelöscht werden), ist die Datenbank in einem Volumen zu finden, dass über ein PeristentVolumeClaim durch Kubernetes bereitgestellt wird.

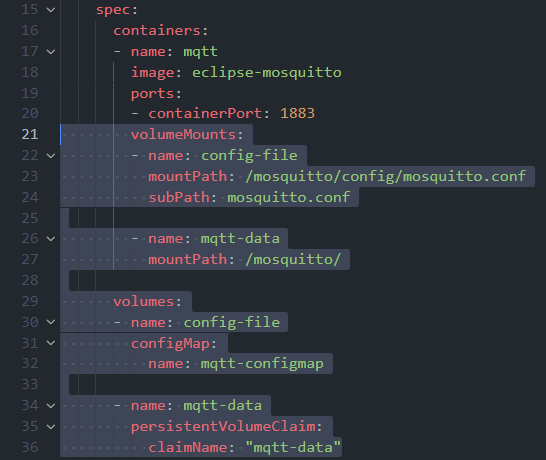
Diese Anforderungen, dass die Daten beim Neustart der Container   
von MongoDB nicht gelöscht werden, wurde entsprechen auch in Aufgabe  
7 konfiguriert. Hier wird bei der Erstellung mittels der docker-compose.yaml Datei, dem service mongo ein seperates Volume erstellt in diesem die MongoDB Datenbank gespeichert ist und bei Neustart des Containers beibehalten wird.

Aufgabe-8\mongodb\mongodb-deployment.yaml

Aufgabe-6-7\docker-compose.yaml

## MQTT

Als Message Broker soll innerhalb von Kubernetes Mosquitto-mqtt laufen.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungFür die passende Konfiguration von Mosquitto mqtt wird im Kubernetes eine Configmap verwendet:  
In dieser ist unter anderem definiert, dass mqqt auf Port 1883 lauschen soll und dass anonyme Anfragen erlaubt sind.

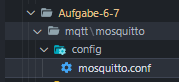
Aufgabe-8\mqtt\mqtt-configmap.yaml

Auch die Persistenz der Daten wird in dieser Configmap eingestellt. Dazu ist, wie bei MongoDB auch, wieder ein persistentes Volumen innerhalb von Kubernetes notwendig. Dieses wird auch wieder mit einem PeristentVolumeClaim dem Pod zugewiesen, sodass die Daten/vorgehaltenen MQTT-Nachrichten bei einem Neustart des Pods nicht verloren gehen

Aufgabe-8\mqtt\mqtt-deployment.yaml

Auch für Aufgabe 7 die Bereitstellung über Docker-Compose ist die Konfigdatei entsprechend eingebunden worden. Dabei muss diese Konfigurationsdatei physisch auf dem Hostsystem liegen. Diese wird über die Docker Volume Mapping Funktion entsprechend dem Container bereitgestellt

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung 

*Aufgabe-6-7\mqtt\mosquitto\config\mosquitto.conf*

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\docker-compose.yaml

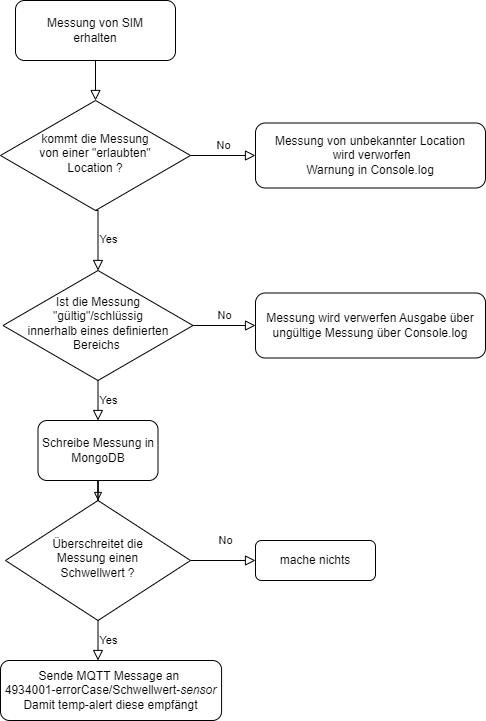
## Rectemp

Ein NodeJS Programm, dass die MQTT-Messages des SIM-Programms entgegennimmt und diese Überprüft,

* Ob eine Messung von einer erlaubten Location kommt
* Ob die Werte schlüssig sind (liegen die Messungen innerhalb eines Gültigkeitsbereichs eines Sensors)
* und ob, die Messungen einen bestimmten Schwellwert überschreiten.

Zusätzlich werden Messungen in eine MongoDB geschrieben und beim Überschreiten eines Schwellwerts einer Messung wird zusätzlich eine mqqt Nachricht an den temp-alert Service geschickt:

Der genaue Ablauf ist mit folgendem **Ablaufdiagramm** beschrieben:



### Config.js

Bei dem Node-JS Programm gibt es eine extra Java-Script Datei in der Konfigurationen unter anderem für die Verbindungs-Parameter der MongoDB, den MQTT-Service gespeichert sind. **(Config.js)**  
Auch sind hier global die erlaubten Locations in einem Array gespeichert und die Schwellwerte innerhalb diesen eine „normale“ Messung pro Sensortyp definiert ist:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\configs.js"

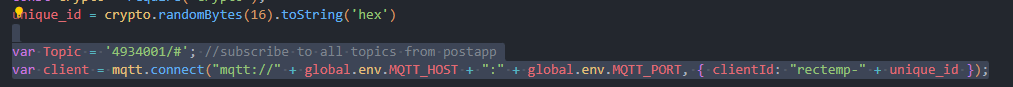
Wertebereich/ Gültigkeitsbereich der Sensoren:Die Wertebereiche, innerhalb diesem eine erhaltene Messung als gültig/schlüssig gilt, wurde auf Grund der Einfachheit des Programms hard-gecoded:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\mqtt.js

### MQTT-CLIENTID:

Beim Start der Node-JS Applikation wird zunächst die Verbindung zum MQTT-Broker initiiert und das Passende Topic wird abonniert:  


Aufgabe-6-7\rectemp\mqtt.js

Dabei ist sicherzustellen, dass bei jedem Start der Anwendung eine (neue) einzigartige ID als Client ID mitgeben wird. Dies ist notwendig, da in Kubernetes diese Anwendung mehrfach laufen soll und daher ist eine unterschiedliche Client ID notwendig, da sonst der MQTT-Broker die Verbindungen verweigern wird.  
Diese wird mit der Funktion *crypto.randomBytes(16).toString('hex')* sichergestellt die einen zufällige 16 Stellige zahl generiert die die Client-id mitgebenden wird.

*(dieses Vorgehen wird auch genauso beim Simulationsprogram SIM gehandhabt, da auch dies theoretisch mehrfach zur Ausführen kommen muss)*

### Schreiben in MongoDB:

Empfängt der rectemp Service eine „gülige“ Messung wird diese in die MongoDB Datenbank geschrieben. Dazu muss zunächst das Daten Schema für mongoDB definiert werden:

Ein Bild, das Text, Bildschirm, Screenshot enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\model\sensor\_werte.js

Anschließend wird die MonogDB URL entsprechend den Parametern der config.js Datei definiert:  
Es wird die mongoosurl mit Authentifizierung genommen   
*mongodb://mongoadmin:secret@mongo:27017/Temperatur?authSource=admin*

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\db.js

Dann kann auch die Messung in die richtige DB geschrieben werden:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\db.js

### Sende Warnung bei Überschreiten von Schwellwert:

Eine Messung wird auf Überschreiten von Schwellwerten überprüft.  
(Dabei kommen die Schwellwerte aus der Config.js Datei).  
Geschieht dies wird die *sende\_schwellwert\_mqqt\_message* Funktion ausgeführt, die wie der Name bereits vermuten lässt, eine mqtt Message an das topic *4934001-errorCase/Schwellwert* sendet.   
Dieses Topic wird vom temp-alert Service überwacht.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\mqtt.js

Ein Bild, das Text enthält.

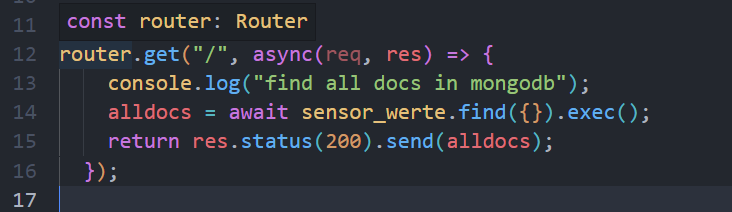
Automatisch generierte Beschreibung

Ausgabe in rectemp bei Überschreiten von Schwellwert

### Zusätzliche Features:

**(Rest-)API-Schnittstelle:**Es wurde ein (Rest-)API-Schnittstelle implementiert die **anonym**:

* Alle Messungen in der Datenbank ausgeben kann:  
  Dazu muss ein GET-Reqest an rectemp Service über localhost:5000/api gesendet werden:

****Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\Routes\api.js"

* Ein Bild, das Text enthält.

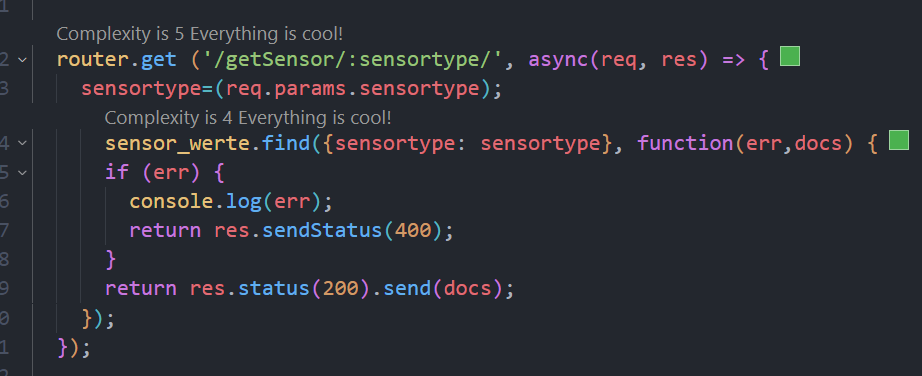
  Automatisch generierte BeschreibungAlle Messungen zu einer bestimmten Location in der Datenbank ausgeben kann:  
  Dazu muss ein GET-Reqest an rectemp Service über localhost:5000/api/getLocid/*1234* gesendet werden: (wobei *1234* eine gewünschte location ist)  
  Ein Bild, das Text enthält.

  Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\Routes\api.js"

* Alle Messungen zu eines bestimmten Sensors in der Datenbank ausgeben kann:  
  Dazu muss ein GET-Reqest an rectemp Service über localhost:5000/api/getSensor/*T*

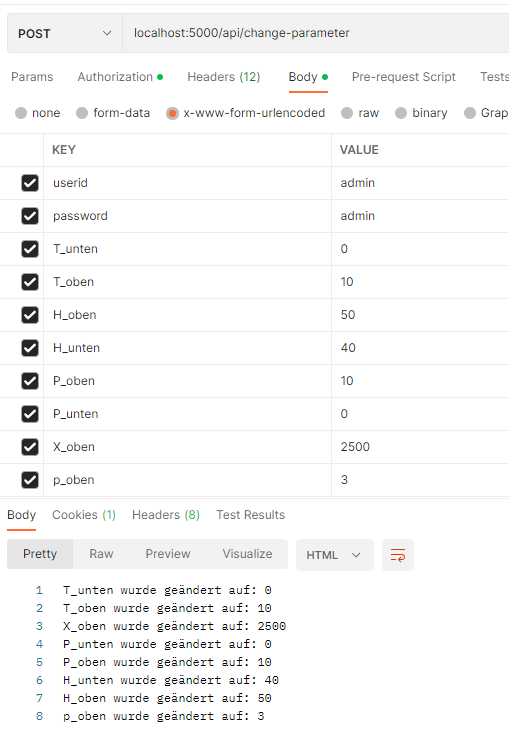
gesendet werden: (wobei *T* eine gewünschter Sensor ist)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-6-7\rectemp\Routes\api.js"

* **Ein Bild, das Text enthält.

  Automatisch generierte Beschreibung**Des Weiteren wurde ein POST-Schnittstelle implementiert die **authentifizierten** Personen es möglich macht die Schwellwerte (aus der Config.js) zu ändern. Dadurch ist da anpassen der Schwellwerten zur Laufzeit des Services möglich und das Auslösen der Warnungen bei unter/überschreiten eines Schwellwerts kann beeinflusst werden.  
    
  Dazu muss ein POST an localhost:5000/api/change-parameter ausgeführt werden und im Body müssen verpflichtend *userid* und *password* (richtig) gesetzt werden. Zusätzlich können die Variablen aus der Config.js im Body angegeben werde, auf welchen Wert diese geändert werden sollen.

Aufgabe-6-7\rectemp\Routes\api.js"



Healthz-Schnittstelle:Für dieLiveness**-**Probe für Kubernetes, also die Probe, ob der Service noch läuft und reagiert ist eine einfach Healthcheck Schnittstelle mittels eines GET-Requests implementiert worden der bei Aufruf von localhost:5000/healthz Status 200 und „OK“ zurück gibt.Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Temp-alert

Einfaches Node-Js Programm, dass das Topic *„4934001-errorCase“* abonniert, in welches rectemp die Messungen, die außerhalb des Schwellwerts liegen sendet.  
Auch hier muss, wie bei dem SIM-Programm ein MQTT-Host definiert werden.   
Da der Service mit der MQTT Service allerdings innerhalb von Kubernetes läuft muss dieser Host (mqtt:1883) so bestehen bleiben. Dieser verweist durch einen Loadbalancer innerhalb von Kubernetes auf den mosquitto MQTT-Broker

Ein Bild, das Text enthält.

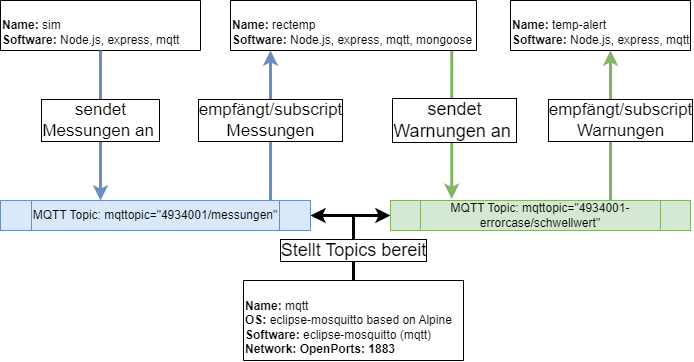
Automatisch generierte Beschreibung

*Aufgabe-6-7\temp-alert\index-skeleton.js*

Ausgabe in Temp-alert:  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Schaubild MQTT Messages:

Folgendes Schaubild hilft des Verständnis wie die einzelnen Services des MVPs sich gegenseitig Messages senden: 

# Praktischer Teil: Kubernetes

Netzwerk:  
Wie aus dem Schaubild auf Seite 7 zusehen sind die Services (rectemp mongo,mqtt und temp-alert) alle in Kubernetes (im selben Namespace), damit diese sich untereinander und auch von außen erreichbar sind sind die Services jeweils (alle außer temp-alert) mit einem internen Kubernetes Loadbalancer Verbunden.

Diese Loadbalancer ermöglichen es das innerhalb von Kubernetes anhand eines Namens des Loadbalancers auf die Pods zugegriffen werden kann. Auch die **Lastverteilung für den Service rectremp, der mit 3 Pods aufgesetzt** wird, kann damit gewährleistet werden.

## Konfigurierbare Verbindungsparameter für den Zugriff auf MongoDB:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungIm MongoDB Deployment von Kubernetes wird auf den Username und das Passwort (was in einem erstellten Secret steht) referenziert. Damit MongoDB innerhalb von Kubernetes mit Username und Passwort initialisiert wird, sind diese Values über die sogenannten Kubernetes-Secrets bei Erstellung des Deployment mitzugeben.

Aufgabe-8\mongoDB\mongodb-deployment.yaml

Aufgabe-8\mongodb\mongodb-secrets.yaml

## Resilienz gegen Ausfall

Um die Resilienz gegen Ausfall zu sichern, wurde der rectemp bereits mit einer Health-check ausgestatte. (siehe Seite 17)

Mit der Kubernetes Funktion *livenessProbe* wird in einem definierten Abstand an GET-Request auf eine definierte URL innerhalb des Pods durchgeführt. Schlägt dies nach (standartmäßig) 3 versuchen fehl ( kein http Status Code 200 oder 301) , wird der Pod neugestartet.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-8\rectemp\rectemp-deployment.yaml

Ein Bild, das Text, drinnen, Bildschirm, Screenshot enthält.

Automatisch generierte Beschreibung

livenessProbe schlägt fehl, da bspw. NodeJs abgestürzt ist  Pods wird neugestartet

## Quota Limit für rectemp

Um ein Quotalimit festzulegen ist folgendes zu Konfigurieren

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Aufgabe-8\rectemp\rectemp-deployment.yaml

# Integrationstest / (Installations-Anleitung)

Vorrausetzung für reibungslosen Test / Deployment ist ein Windows Hostsystem mit Docker-Desktop und Installiertem Kubernetes

Bevor der MVP in Kubernetes deployt werden kann, müssen die Docker-Container zuerst gebaut werden. Dazu ist es hilfreich zuerst in das Verzeichnis „Aufgabe-6-7“ zu Wechsel und anschließend über die docker-compose.yaml file die Container zu bauen:

Lokaler Bau der Container mit docker-compose  
  
 PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe build

Anschließend stehen die Images zur Verfügung:  
Ein Bild, das Text, Screenshot, Spieler, schwarz enthält.

Automatisch generierte Beschreibung

Dann können die Docker-Container testweise lokal gestartet werden. Dazu muss zuerst das Netzwerk mqtt erstellt werden:

PS C:\Users\Thomas\Aufgabe-6-7> docker network create mqtt

Dann können die Container mit docker-compose gestartet werden.

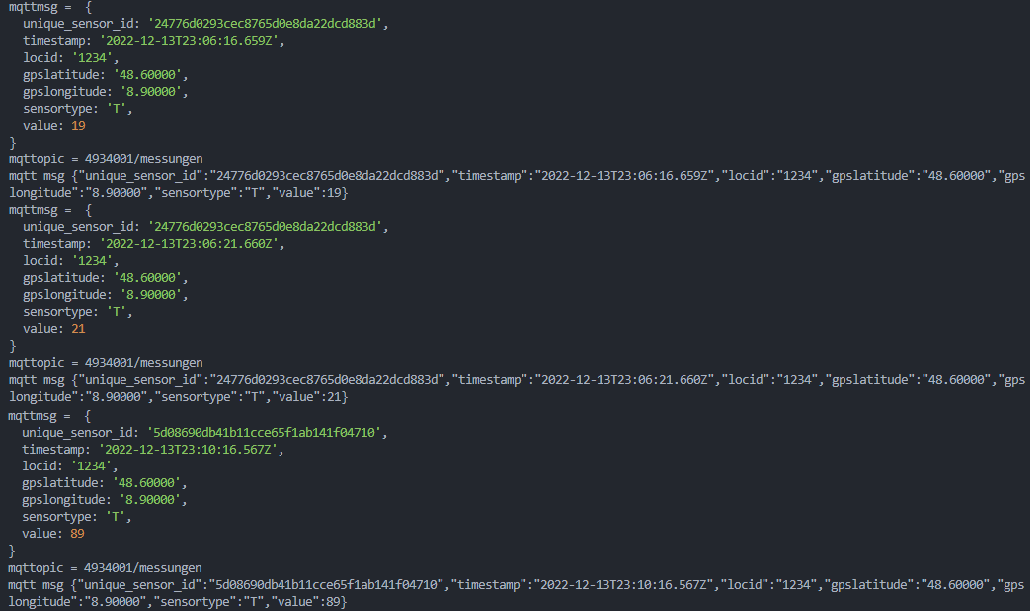
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe up

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Die Container sollten nun lokal in Docker laufen.

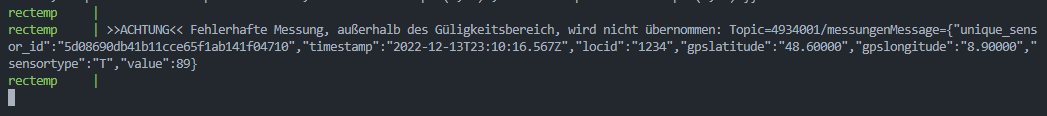
Startet man das Sim- Programm auf einer weiteren Shell sieht man auch schon diesen MQTT-Messages senden und von rectemp empfangen

PS C:\Users\Thomas\Aufgabe-5> npm start 5 1234 500 T 16 88  


Simulationsprogram sendet 3 Messwerte der Temperaturen. Einmal 19 Grad 21 Grad und 89 Grad

Die drei Messungen kommen an rectemp an:

* Messung 19 Grad ist in Ordnung und wird in die MongoDB gespeichert
* Messung 21 Grad ist nicht In Ordnung, da der Schwellwert überschritten (Temperatur darf nur zwischen 16 und 19 Grad sein) wird dennoch in MongoDB geschrieben
* Messung 89 Grad ist offensichtlich ein Fehler und liegt außerhalb des Gültigkeitsbereichs des Sensors, daher wird die Messung verworfen

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ausgabe der docker-container temp-alert und rectemp

Diese Daten sind dann auch in der MongoDB zu finden:

Connection: String: *mongodb://mongoadmin:secret@localhost:27017/Temperatur?authSource=admin*  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Und auch die Abfrage über die REST-Schnittstelle: http://localhost:5000/api/getSensor/T  
liefert die Ergebnisse:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
  
Somit ist sichergestellt, dass kann mit dem Kubernetes Deployment fortgefahren werden. Dazu müssen zunächst Container gestoppt und gelöscht werden. Mit CTRL + C werden die Container gestoppt. Anschließend müssen sie gelöscht werden.

PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe down  
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe rm

Deployment in Kubernetes

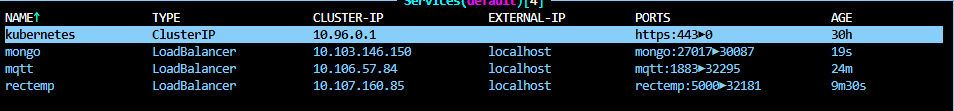
Damit die Anwendungen in Kubernetes laufen, müssen die Deployment. Yaml Files aus den Ordner Aufgabe-8 gestartet werden:

PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\mongodb\  
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\mqtt\  
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\rectemp\  
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\alerttemp\  
  
  
Pods:

Ein Bild, das Text, Monitor, Bildschirm, Screenshot enthält.

Automatisch generierte Beschreibung

Services:



**Beispiele:**

Anschließend kann das Sim- Programm weiter wieder Messungen generieren, diese landen dann im rectemp Service innerhalb von Kubernetes

recttemp Ausgabe:  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibungtemp-alert Ausgabe:  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
monogdb: