

---

# IoT Erfassung und Darstellung Sensordaten

CAS Big Data

Semesterarbeit

Studiengang:

Autor:

Dozent:

Experte:

Datum:

CAS Big Data

Marc Habegger

Max Kleiner

2. Oktober 2019

# Inhaltsverzeichnis

<b>1 Einleitung</b>	2
<b>2 Sensor Plattform</b>	3
2.1 Mikrokontroller	3
2.2 Funkverbindung	3
2.3 Sensoren	4
2.4 Aufbau	4
2.5 Funkverbindung mit dem MQTT Gateway	5
2.5.1 Absicherung der Funkverbindung	6
2.6 Software	6
2.6.1 Verwendete Bibliotheken	6
2.6.2 Konfiguration eines Sensors	7
2.6.3 Datenübertragung	7
<b>3 MQTT Gateway</b>	9
3.1 Übersicht	9
3.2 Hardware	9
3.2.1 Aufbau	9
3.2.2 Zugriffsschutz	10
3.2.3 Warum kein TLS?	10
3.3 Software	10
3.3.1 Verwendete Bibliotheken	10
3.3.2 Empfang der Daten	11
3.3.3 Senden der Daten an den MQTT Broker	12
3.3.4 Konfiguration	12
<b>4 Speicherung und Visualisierung</b>	13
4.1 Übersicht	13
4.2 Hardware	13
4.3 MQTT Broker	14
4.3.1 Installation	14
4.3.2 Warum kein TLS?	14
4.4 Datenspeicherung	14
4.4.1 Telegraf	15
4.4.1.1 Installation / Konfiguration	15
4.4.2 InfluxDB	16
4.4.2.1 Installation	16
4.4.2.2 Datenbank einrichten	17
4.5 Visualisierung	17
4.5.1 Installation	17
4.5.2 Grafana Dashboard	18
4.6 Ersatz für Kafka	19

<b>5 Fazit</b>	20
<b>6 Verwendete Bauteile</b>	20
6.1 ESP32 Mikrokontroller	21
6.2 Arduino	22
6.3 Raspberry Pi	22
6.4 Sensoren	23
6.4.1 Temperatur und Luftfeuchtigkeit	23
6.4.1.1 DHT22	23
6.4.1.2 DHT11	23
6.4.2 Bodenfeuchtigkeit	23
6.4.3 Funk	24
<b>Index</b>	27

# Management Summary

Das Internet der Dinge (Internet of Things (IoT)) zeichnet sich, neben einer allgegenwärtigen Verfügbarkeit von Daten, auch durch eine grosse Anzahl der Datenquellen aus. Mit dieser Arbeit soll das Zusammenspiel der verschiedenen Komponenten eines Sensornetzwerkes mit einem Speichersystem und einer grafischen Analyse aufgezeigt werden.

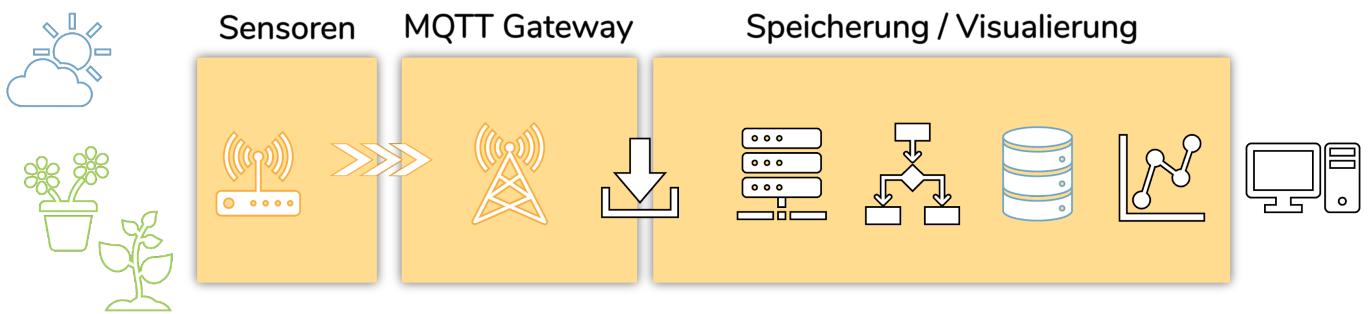


Abbildung 0.1: Systemaufbau Funknetzwerk

Das realisierte Netzwerk besteht aus mehreren Sensoren, die Messdaten über Funk an ein MQTT-Gateway senden, welches über ein Netzwerk mit einer Datenbank verbunden ist. Für die Speicherung der Daten wird eine Zeitreihendatenbank verwendet, die speziell für die Behandlung von Messwerten optimiert ist.

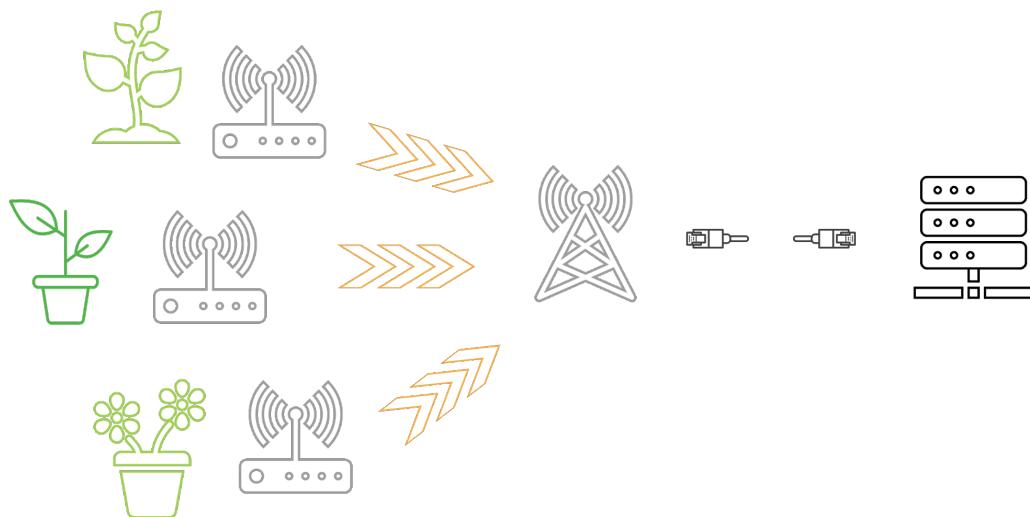


Abbildung 0.2: Systemaufbau

Schlussendlich werden die Daten visualisiert und können über einen Browser betrachtet werden. Beim erreichen oder unterschreiten selbst definierter Schwellwerte kann ein Alarm ausgelöst werden.

# 1 Einleitung

Der gesamte Code und die Dokumentation dieser Arbeit ist auf Github unter folgendem Link zu finden:

<https://github.com/habis-git/CAS-BGD-SemesterArbeit>

und wird auch nach Abschluss dieser Arbeit weiterentwickelt. Die Komponenten wurden durch das Vorhandensein von Bibliotheken und Treibern, sowie die leichte Beschaffung auch innerhalb der Schweiz, ausgewählt. Der Arduino Mega als MQTT Gateway und der Raspberry Pi 2 für die Linux Plattform waren bereits von früheren Projekten vorhanden und wurden deshalb in diesem Projekt eingesetzt.

Der Aufbau dieses Dokumentes folgt der Übersichtsgrafik mit den Sensoren auf der linken Seite und der Visualisierung auf der Rechten Seite und dementsprechend werden die Sensoren zuerst beschrieben und die Visualisierung findet sich gegen Ende des Dokumentes.

Diese Dokumentation hat nicht den Anspruch den Aufbau des Projektes in jedem Punkt zu erläutern, gewisse Arbeiten wie z. Bsp. Lötarbeiten oder auch Konfigurationen auf Betriebssystem Ebene fehlen. Die Übersicht der Webseiten [6.4.3] welche die nötigen Informationen für dieses Projektes lieferten, kann bei solchen Fragen weiterhelfen.

Das ursprüngliche LaTeX Template für dieses Dokument stammt aus diesem Projekt:

<https://gitlab.ti.bfh.ch/bfh-latex/bfh-latex>

## 2 Sensor Plattform

### 2.1 Mikrokontroller

Ein sehr beliebter Mikrokontroller ist der ESP32 von Espressif [9], welcher durch grossen Funktionsumfang und geringem Preis überzeugen kann. Ein Vorteil ist ebenfalls die Möglichkeit der Programmierung durch die Arduino[10] IDE, welche frei verfügbar ist und die Entwicklung stark vereinfacht. Es existieren viele verschiedene Mikrocontroller, welche den ESP32 Baustein verwenden. Für dieses Projekt kam der Firebeetle [11] von DF Robot zum Einsatz.

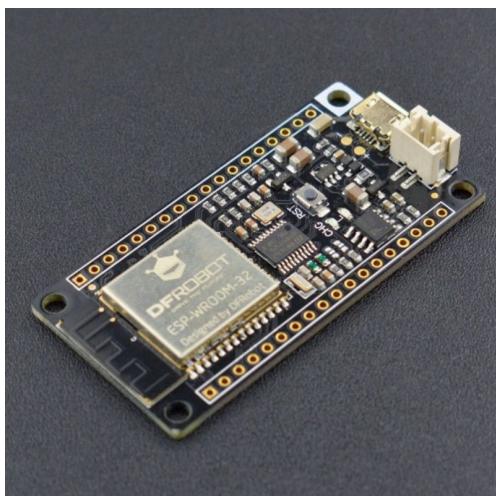


Abbildung 2.1: Firebeetle mit dem ESP32 Mikrokontroller

Der ESP32 besitzt sowohl WLAN als auch Bluetooth LE Schnittstellen. Diese wurden jedoch aus Gründen des Stromverbrauchs (WLAN) oder der Reichweite (Bluetooth LE) in diesem Projekt nicht verwendet.

Der Firebeetle[11] Baustein hat sowohl die Möglichkeit einer Stromversorgung mit 5 Volt, als auch mit 3.7 Volt, womit er direkt durch Lithium-Ionen Akkus betrieben werden kann.

### 2.2 Funkverbindung

Die Wahl der Funktechnologie wurde nach folgenden Punkten getroffen:

- Stromverbrauch (je kleiner desto besser)
- Reichweite (20-30 m für die Anwendung im Garten)
- Preis
- Verfügbarkeit von Bibliotheken für die Programmierung

Aus der Tabelle 2.1 geht hervor, dass die ZigBee Technologie ein attraktives Paket aus geringem Stromverbrauch und guter Reichweite bietet. Allerdings sind die Module für ZigBee relativ kostspielig (20-30 Fr.), womit sich jeder einzelne Sensor stark verteuert. Bluetooth LE ist zwar ebenfalls sehr stromsparend, dies allerdings nur in der kürzesten Reichweite (>10m), welche für dieses Projekt unzureichend ist.

	NRF24	ZigBee	Bluetooth LE	WLAN
Frequenz	2.4 GHz	868/915 MHz & 2.4 GHz	2.4 GHz	2.4–5 GHz
Stromverbrauch	18 mA	30–40 mA	30–15 mA	116–22 mA
Benötigte Leistung	60 mW	36.9 mW	215 mW–10 mW	835 mW–200 mW
Reichweite	10–50 m	10–300 m	10–30 m	100–500 m

Tabelle 2.1: Leistung der verschiedenen Funktechniken im Vergleich

Quelle: Design of Wireless Sensors for IoT with Energy Storage and Communication Channel Heterogeneity [24]

Schussendlich war es die Kombination aus guter Reichweite, sehr geringem Stromverbrauch und extrem günstigem Preis (< 2 Fr.) bei den NRF24 Modulen, die überzeugte.

## 2.3 Sensoren

Es existiert eine grosse Anzahl an verschiedenen Sensoren, welche auch für Privatanwender nutzbar sind. In diesem Projekt wurden Sensoren gewählt für welche frei verfügbare Bibliotheken in einer Version für den ESP32 vorhanden sind. Grundsätzlich wird bei den Sensoren unterschieden zwischen solchen die

- Digitale Werte liefern, die direkt einer Einheit (Grad Celsius) oder Prozentangabe (Luftfeuchtigkeit) entsprechen
- Analoge Messwerte, welche auf eine Einheit umgerechnet werden müssen

Für dieses Projekt wurden mehrere Sensoren ausgewählt und kombiniert:

- Lufttemperatur- und Luftfeuchtigkeitssensoren DHT11 [6.4.1.2] und DHT22 [6.4.1.1]
- Bodenfeuchtesensor V1.2 [6.4.2]

## 2.4 Aufbau

Die Komponenten werden alle mit den 3.7 Volt und Masse Anschlüssen verbunden. Das Funkmodul benötigt die SPI Kontakte SCK, MOSI und MISO und zusätzlich zwei digitale Pins für CE und CSN, die frei gewählt werden können. In meinem Code habe ich CE auf den Pin 25 gesetzt und CSN auf Pin 26. Wenn man andere Pins verwenden möchte, dann kann dies bei der Initialisierung des Funkmoduls angegeben werden:

```
1 radio = new RF24(25, 26); //CE, CSN
```

Der Daten-Input des Temperatur Sensors benötigt einen digitalen Eingang, welchen ich auf den Pin 27 gesetzt habe. Der Bodenfeuchtigkeitssensor hat einen analogen Ausgang, welchen ich mit Pin 4 verbunden habe. Für andere Pins können die folgenden Variablen angepasst werden:

```

1 // DHT Sensor
2 uint8_t DHTPin = 27;
3 // Moisture Sensor
4 uint8_t MOISTSENSOR_PIN = 4;
```

Die Schaltung kann über den Mikro-USB Anschluss oder einen Lithium-Ionen-Akku mit Strom versorgt werden. Wenn ein Akku direkt angeschlossen werden soll, hat es auf der Rückseite unterhalb des USB Anschlusses zwei Lötaugen. Ein so angeschlossener Akku kann danach über die USB Buchse geladen werden, der ESP32 enthält zu diesem Zweck eine Lade-Elektronik.

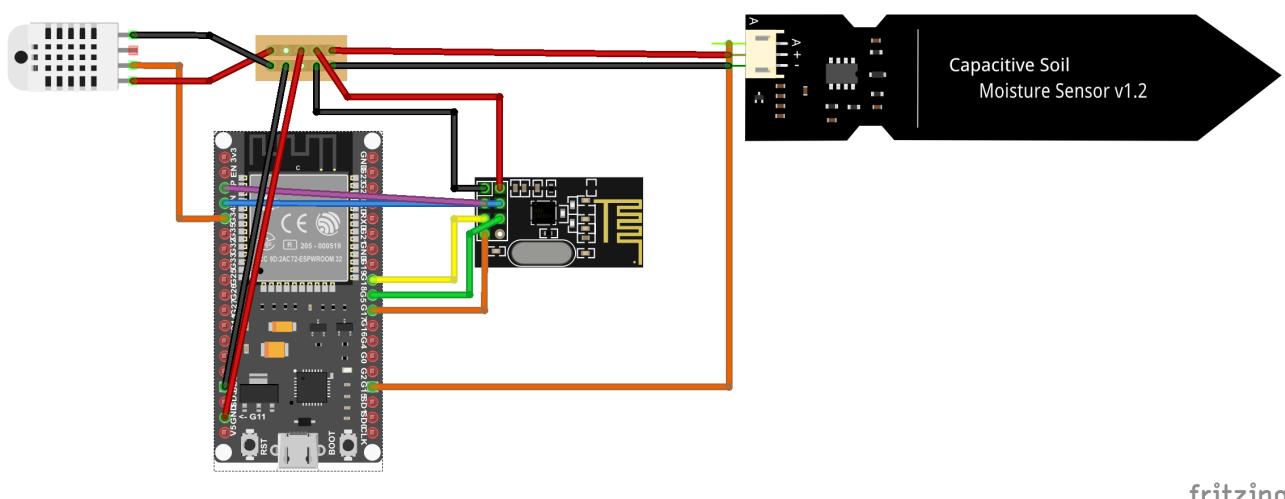


Abbildung 2.2: Verdrahtung der Komponenten

## 2.5 Funkverbindung mit dem MQTT Gateway

Da die Sensoren nicht WLAN als Funktechnik verwenden, können sie nicht direkt auf den MQTT Server verbinden und diesem Nachrichten senden. Diese Aufgabe übernimmt das MQTT-Gateway, welches auf der einen Seite ebenfalls mit einem NRF24 [2.5] Funkmodul ausgerüstet ist und andererseits eine Ethernet Netzwerkanbindung hat.

Jedes Funkmodul kann mit 5 anderen Modulen kommunizieren, dabei muss jedes Modul eine eigene Adresse verwenden unter der es senden und empfangen kann.

ESP32      Arduino Mega

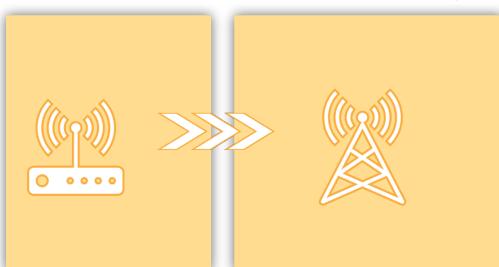


Abbildung 2.3: Funkverbindung

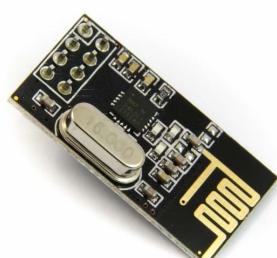


Abbildung 2.4: NRF24L01+ Funkmodul

Bei der Programmierung der Sensoren wurde darauf geachtet, dass die Identifikation der Sensoren über eine einzigartige Id unabhängig vom Programmcode vorgenommen werden kann [2.6.2].



Abbildung 2.5: Funkkanäle des NRF24L01

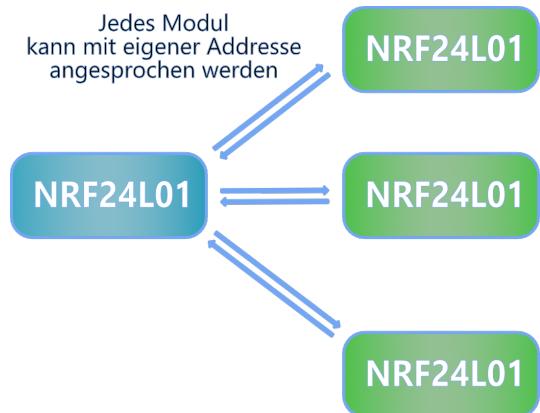


Abbildung 2.6: Addressierung Funkmodule

## 2.5.1 Absicherung der Funkverbindung

Die Identifikation eines Sensors ist aber nicht zu Verwechseln mit einer Prüfung ob der Sender überhaupt berechtigt ist Daten zu senden. Da das Funkprotokoll unverschlüsselt ist, können andere Sender ebenfalls Nachrichten absenden und so Daten verfälschen oder die Verarbeitung stören. Um dies zu verhindern, wird eine Technik Namens Keyed-Hash Message Authentication Code (HMAC) eingesetzt. Die eigentliche Nachricht

4;25.1;71.1;10.2

wird dabei um einen Hash-Wert erweitert

3A51266ABD432C1FB797DEDEE215FD16E42BAB627F104BCB6F4169EDE4544582:4;25.1;71.1;10.2

und die Nachricht mit HMAC und den Messwerten wird an das Message Queuing Telemetry Transport (MQTT) Gateway geschickt.

## 2.6 Software

### 2.6.1 Verwendete Bibliotheken

**SPI library** Ansteuerung der Sensoren [4]

**mbed TLS** Berechnung des HMAC [3]

**RF24** Funkmodul [7]

**DHT-sensor-library** Luft- und Feuchtigkeitssensor [8]

Einige der aufgeführten Bibliotheken können direkt über die Bibliotheksverwaltung der Arduino IDE installiert werden. Ansonsten können die Bibliotheken über den Link (führt meistens auf [github.com](https://github.com)) heruntergeladen und von Hand installiert werden. **Achtung: Es gibt viele Bibliotheken mit ähnlichem Namen und Beschreibung. Falls das kompilieren fehlschlägt bitte prüfen, ob die richtige Bibliothek eingebunden wurde.**

## 2.6.2 Konfiguration eines Sensors

Der Flash-Speicher des ESP32 kann in mehrere Bereiche unterteilt werden. So können Dateien unabhängig von dem kompilierten Programm auf den Mikrocontroller geladen werden. Während der Programm-Code während der Entwicklung ständig ändert und immer wieder via Sketch-Upload auf den ESP32 übertragen wird, werden die Konfigurationsparameter nur einmal festgelegt.

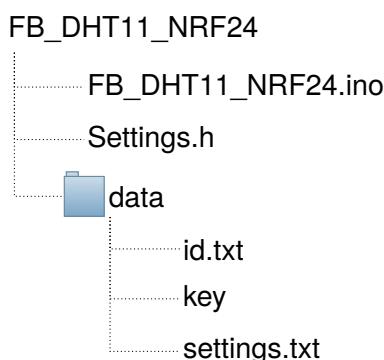


Abbildung 2.7: Verzeichnisstruktur Sensor

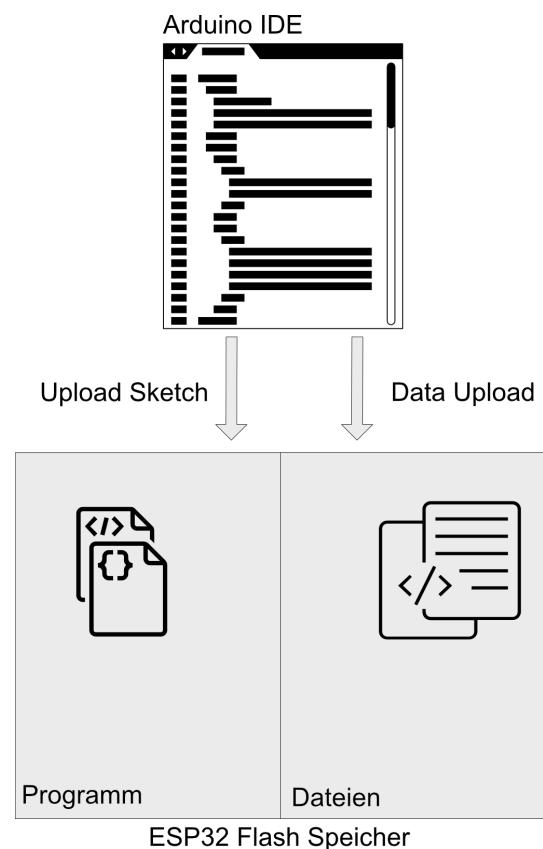


Abbildung 2.8: Speicher-Aufteilung ESP32

Jeder Sensor benötigt einige Parameter:

**Id** Identifiziert den Sensor, gespeichert in der Datei id.txt

**key** Der geheime Schlüssel um den HMAC zu berechnen, gespeichert in der Datei key. Dieser Schlüssel muss auf den Sensoren wie auf dem Gateway identisch sein.

Die folgende Parameter werden alle in der Datei settings.txt gespeichert:

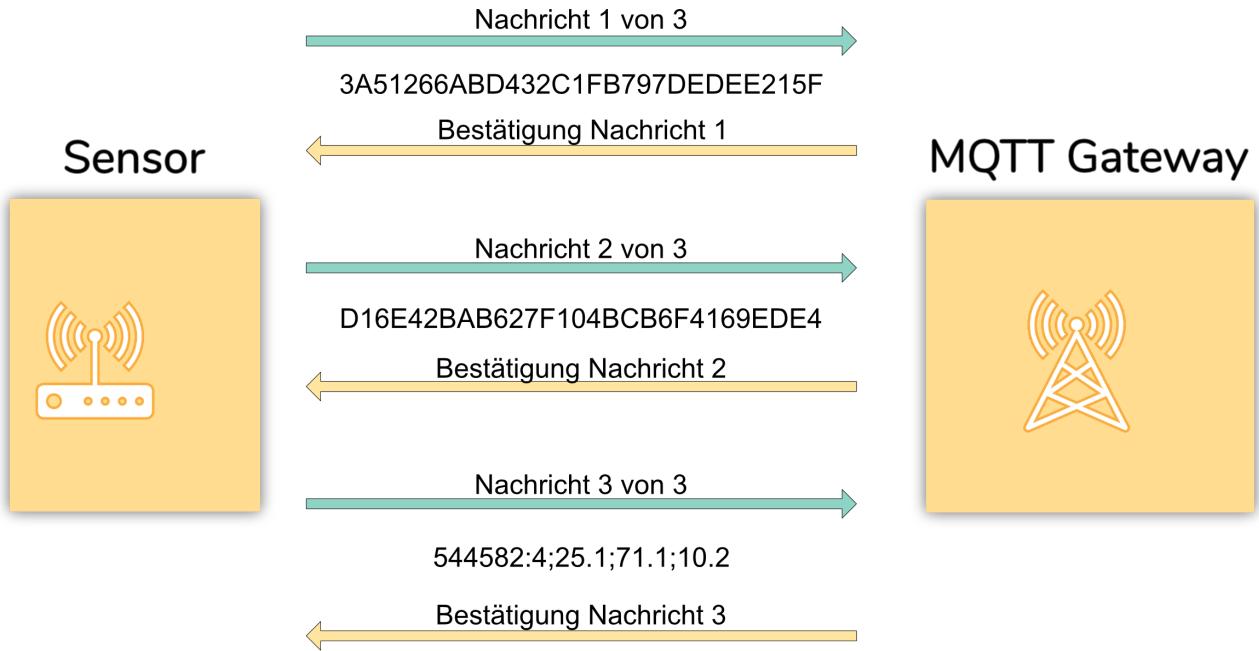
**sensorType** Typ des Temperatursensor, entweder DHT11 oder DHT22

**sleepTime** Wartezeit zwischen zwei Messungen in Millisekunden

**paLevel** Funkstärke des NRF24 Moduls, erlaubte Werte sind 'low', 'med' und 'high'

## 2.6.3 Datenübertragung

Durch die Verwendung des HMAC wird die Nachricht stark vergrössert und erreicht um die neunzig Zeichen. Da die maximale Grösse eines Datenpaketes des NRF24 Moduls nur 32 Zeichen beträgt, muss die gesamte Nachricht in mehreren Paketen übertragen werden.



3A51266ABD432C1FB797DEDEE215FD16E42BAB627F104BCB6F4169EDE4544582:4;25.1;71.1;10.2

Abbildung 2.9: Übertragen der Messdaten in mehreren Paketen

```

1  bool ok = radio->write( &sending, sizeof(sending));      //Sends first packet to radio
2
3  if (ok) {
4      byte packetReceived = 0; //packetReceived (feedback) = 0th packet
5      radio->startListening(); //start listening for a response
6      int retry = 0;
7      while (!confirmed && retry < 5) {
8          delay(50);
9          if (radio->available()) {
10              radio->read(&packetReceived, sizeof(packetReceived)); //save response
11              if (packetReceived == pkt) { //If packet number received == packet sent
12                  confirmed = true;
13                  Serial.println("Packet was confirmed, proceed to send next packet.");
14              }
15              else {
16                  confirmed = false;
17                  sendArray = false;
18                  Serial.println("Either nothing sent/no confirmation/Mismatch of packets
19                  → (resend??);");
20              }
21          }
22          retry++;
23      }
24      radio->stopListening();
}

```

# 3 MQTT Gateway

## 3.1 Übersicht



Die Aufgabe des MQTT Gateways ist es die Messdaten, welche über Funk gesendet werden, zu empfangen und an den MQTT Broker weiterzuleiten. Dazu ist es mit einem Funkmodul und einer Ethernet Schnittstelle ausgerüstet.

## 3.2 Hardware

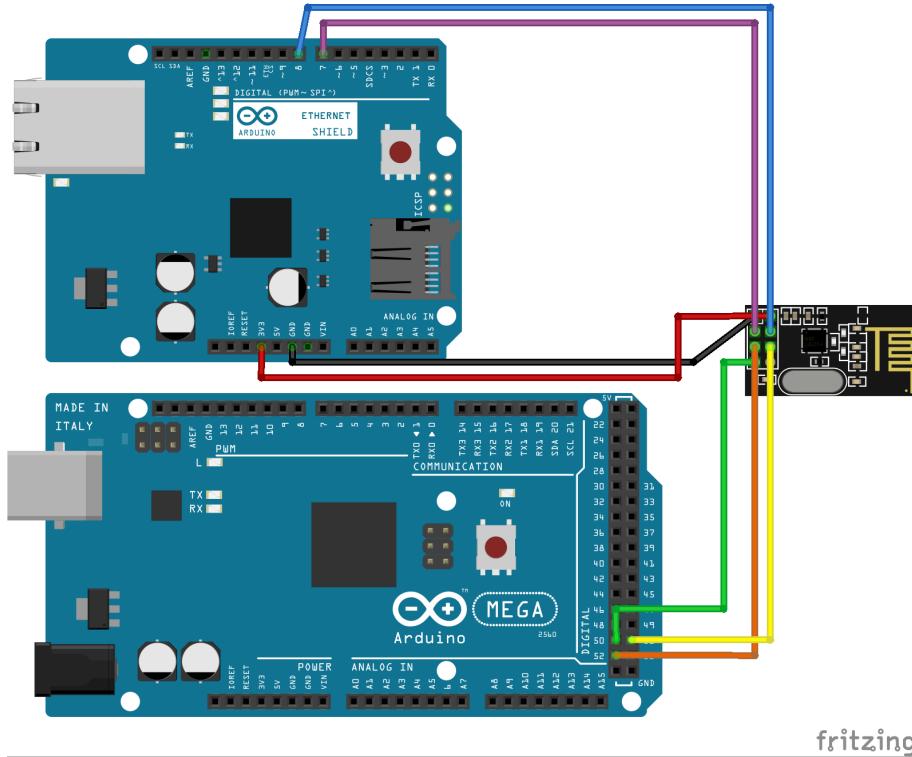
Die Basis des MQTT Gateway bildet ein Arduino Mega [6.2] welcher mit einem Ethernet Shield kombiniert wird. Die Funkverbindung wird wie bei den Sensoren durch ein NRF24 [2.5] Modul hergestellt.

### 3.2.1 Aufbau

Die Stromversorgung des Funkmoduls erfolgt über VCC (3.3V) und GND.

Die zusätzlichen Pins werden wie folgt mit dem Arduino verbunden (Links -> Pin Arduino, Rechts -> Anschluss Funkmodul):

- 8 - CSN
- 7 - CD
- 52 - SCK
- 51 - MOSI
- 50 - MISO



fritzing

Abbildung 3.1: Verdrahtung der Komponenten

### 3.2.2 Zugriffsschutz

Jede empfangene Nachricht wird durch die Valdierung des HMAC (HMAC) sowohl auf einen berechtigten Absender, als auch auf unverfälschte Daten überprüft [2.5.1].

Der Empfänger der Nachricht, welcher denselben geheimen Schlüssel wie der Sender verwendet, berechnet aus den übertragenen Daten ebenfalls den HMAC und vergleicht den selbst berechneten Wert mit dem empfangenen Wert. Wenn diese beiden Werte nicht übereinstimmen so wird der Datensatz verworfen.

### 3.2.3 Warum kein TLS?

Die etablierten Verschlüsselungstechniken wie Transport Layer Security (oder SSL) können mit IoT Geräten nicht verwendet werden. Ein Verbindungsaufbau mit TLS ist ein Zeit- und Rechenintensiver Prozess welcher ein vielfaches der Übertragungsdauer der Daten in Anspruch nehmen würde. Der Stromverbrauch wäre mit einer solchen Technik hoch und die Laufzeiten der Geräte zu gering.

## 3.3 Software

### 3.3.1 Verwendete Bibliotheken

**pubsubclient** Anbindung eines MQTT Brokers [13]

**ethernet** Benutzung einer Ethernet Verbindung [14]

**NRF24** Verwendung des NRF24 Funkmoduls [15]

**cryptosuite** SHA und HMAC Berechnungen [16]

### 3.3.2 Empfang der Daten

Um von allen Sensoren Daten empfangen zu können wird auf jeder Adresse eines Sensors ein Kanal geöffnet.

```
1  Serial.println(F("Open pipes"));
2  for (int i = 1; i < pipes_length; i++) {
3      radio.openReadingPipe(i, pipes[i]);
4      Serial.print(F("Listening to pipe: "));
5      char c_address[8];
6      sprintf(c_address, "-> %s", pipes[i]);
7      Serial.print(c_address);
8      Serial.print(" on channel ");
9      Serial.println(i);
10 }
```

Sobald ein Paket empfangen wird, extrahiert das Gateway die Anzahl Pakete pro Nachricht und fordert das nächste fehlende Paket vom Sensor an.

```
1  byte data[32];
2  if (!allReceived && radio.available()) {
3      radio.read( &data, sizeof(data) ); //place received data in byte structure
4      byte pkt = data[0]; //part of header, packet number received
5
6      for (int i = 2; i < 32; i++) {
7          received[(pkt - 1) * 30 + (i - 2)] = data[i];
8      }
9      delay(5);
10     radio.stopListening(); //stop listening to transmit response of packet
11     ↵ received.
12
13     byte* address = pipes[pipe_num];
14     radio.openWritingPipe(address);
15     bool ok = radio.write(&pkt, sizeof(pkt)); //send packet number back to confirm
```

Sobald die gesamte Nachricht empfangen wurde wird der HMAC mit dem durch das Gateway selbst berechneten Wert verglichen:

```
1  char* hmac = strtok(received, ":" );
2  char* data = strtok(NULL, ":" );
3
4  Sha256.initHmac(key, keyLength);
5  Sha256.print(data);
6  uint8_t* calculatedHmac = Sha256.resultHmac();
7
8  char calculatedHmac_char[hmacSize * 2];
9  for (int cnt = 0; cnt < hmacSize; cnt++)
10 {
11     // convert byte to its ascii representation
12     sprintf(&calculatedHmac_char[cnt * 2], "%02X", calculatedHmac[cnt]);
13 }
14
15 uint8_t authorizationCheck = strcmp(calculatedHmac_char, hmac);
16
17 if (authorizationCheck != 0) {
```

```

18     Serial.println("Hmac not matching, not authorized or tampered message! Message
19     ↪ rejected");
}

```

Falls der empfangene Wert und der berechnete Wert nicht übereinstimmen wird die Nachricht verworfen.

### 3.3.3 Senden der Daten an den MQTT Broker

Nachdem die Nachricht eines Sensors validiert wurde, wird das Format in JSON umgewandelt und an ein MQTT Topic verschickt.

```

{
    "ID": 0001,
    "Temp.Air": 22.6,
    "Hum.Air": 78.1,
    "Hum.Soil": 43.7
}

```

Abbildung 3.2: Sensor Datensatz in JSON Format

Alle Datensätze werden in ein gemeinsames Topic namens /weather/ gespeichert. In einer frühen Version hatte ich für jede Kategorie von Messwerten (Lufttemperatur, Luftfeuchtigkeit etc.) ein eigenes Topic verwendet. Dies war aber unpraktisch für die Weiterverarbeitung und wurde zu einem einzelnen Topic geändert.

```

1 char json[100];
2 sprintf(json, "{\"ID\": \"%s\", \"Temp.Air\": %s, \"Hum.Air\": %s, \"Hum.Soil\": %s}",
3   ↪ SensorId, Temperature, Humidity, SoilHumidity);
4 if (!client.connected()) {
5   reconnect();
6 }
7 if (client.connected()) {
8   Serial.print("Send data to topic '/weather/' ");
9   client.publish("/weather/", json);
10  Serial.println(json);
11 }

```

### 3.3.4 Konfiguration

Das Gateway benötigt für die Berechnung des HMAC den gleichen Schlüssel wie der Sensor. Leider bietet der Arduino nicht die Möglichkeit Dateien in einem eigenen Speicherbereich abzulegen. Der benötigte geheime Schlüssel wird deshalb in der Datei keyFile.h als String definiert. So kann der Code weiterhin in dem Versionskontrollsysteem hinterlegt werden ohne den geheimen Schlüssel preisgeben zu müssen.

Die Datei keyFile.h enthält nur eine Zeile mit der Deklaration des Schlüssels

```
String key_str = "geheimerSchlüssel";
```

Arduino\_NRF24\_Receiver

```
├── Arduino_NRF24_Receiver.ino
└── keyFile.h
```

# 4 Speicherung und Visualisierung

## 4.1 Übersicht

Für die Speicherung und Verarbeitung der gesammelten Sensordaten sind mehrere Softwarekomponenten zuständig.

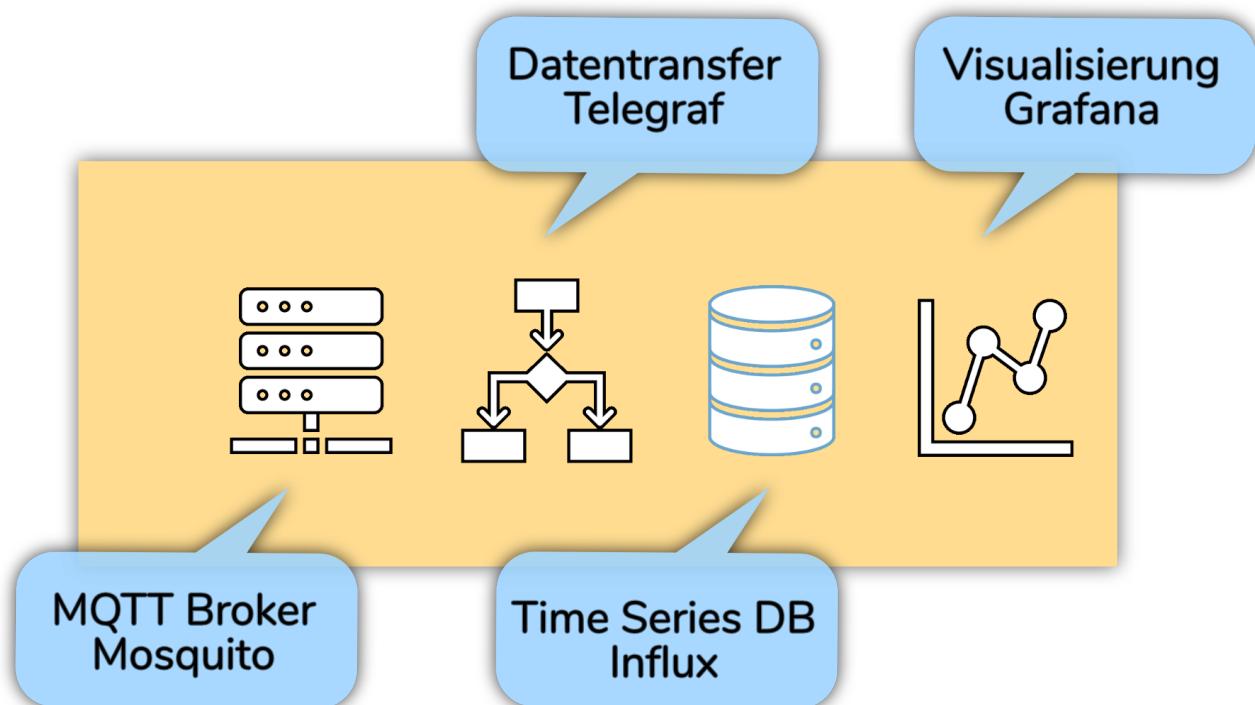


Abbildung 4.1: Verwendete Software

**MQTT Broker** Die von dem MQTT-Gateway geschickten Nachrichten werden durch Mosquitto [17] in einer Queue bis zu einer weiteren Verarbeitung gespeichert

**Datentransfer** Um die Daten aus einer (oder mehreren) MQTT Queue in eine Datenbank zu transferieren wird Telegraf [18] mit dem MQTT Plugin benutzt [19]

## 4.2 Hardware

Als Plattform für Mosquitto [17], Telegraf [18], InfluxDB [20] und Grafana [22] dient ein Raspberry Pi2 Model B [6.3].

## 4.3 MQTT Broker

Als MQTT Broker wird die Open Source Software Mosquitto [17] verwendet.

### 4.3.1 Installation

Die Installation von Mosquitto sollte ebenfalls den Mosquitto-Client umfassen, damit man nach der Installation diese direkt testen kann.

---

```
sudo apt-get install -y mosquitto mosquitto-clients
```

---

Um den Broker automatisch bei jedem Neustart zu starten, muss der Service noch aktiviert werden.

---

```
systemctl enable mosquitto.service
```

---

### 4.3.2 Warum kein TLS?

Eigentlich wollte ich SSL mit Client-Zertifikaten für die Verbindung zwischen dem MQTT Gateway und dem MQTT Broker verwenden. Beispiele dazu kann man im Internet finden, sogar eine direkte Verbindung zwischen ESP32 Geräten und einem MQTT Broker mit SSL Mutual Auth ist möglich [[26], [27]].

Allerdings stellte sich heraus, dass die Ethernet Bibliothek kein SSL unterstützt. Die WI-FI Bibliothek unterstützt dies hingegen und es gibt Tips wie man mit einigen Klassen aus der WI-FI Implementierung die Ethernet Bibliothek anpassen könnte. Ich habe allerdings aus Zeitgründen dieses Thema nicht weiter verfolgt.

## 4.4 Datenspeicherung

Die Messdaten werden in der Zeitreihendatenbank Influx [20] gespeichert. Eine Zeitreihendatenbank erlaubt das schnelle schreiben von Messwerten und kann automatisch ältere Datensätze entfernen um den Platzbedarf nicht zu gross werden zu lassen.

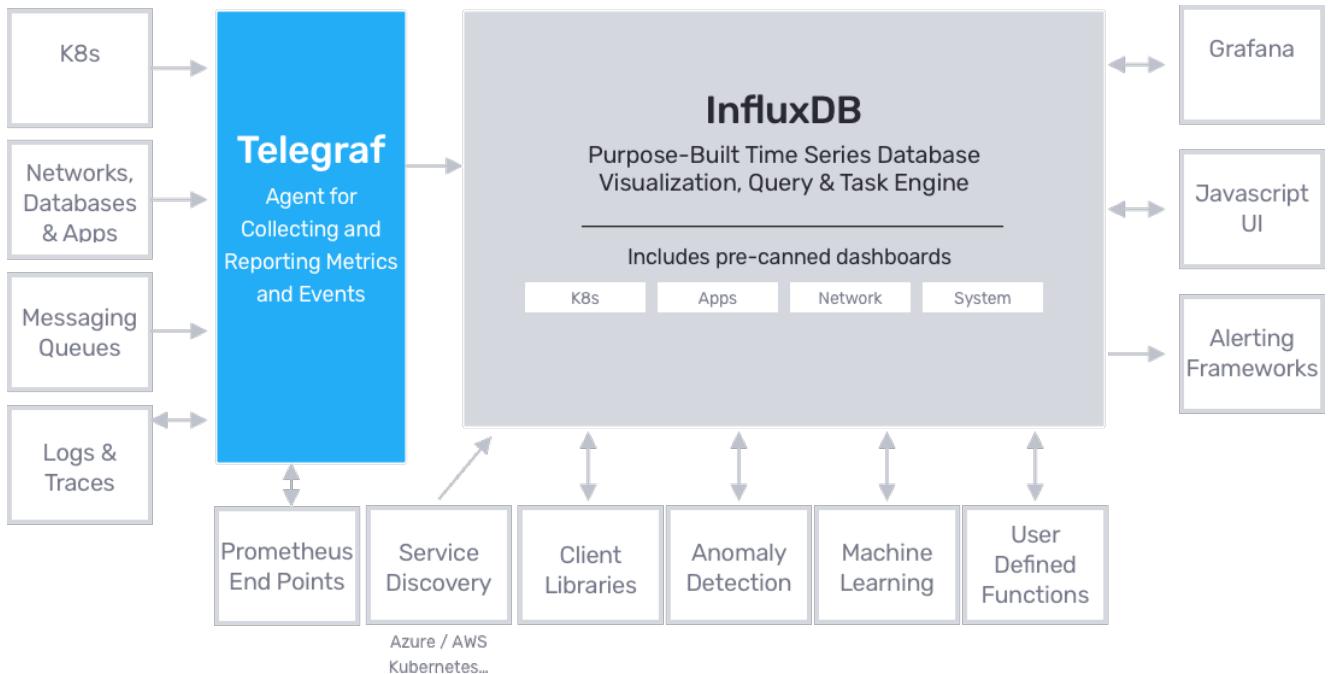


Abbildung 4.2: Übersicht Influx Software

Quelle: [www.influxdata.com](http://www.influxdata.com)

## 4.4.1 Telegraf

Um die Daten aus dem MQTT Topic zu lesen und in die Datenbank zu schreiben, wird die Software Telegraf [18] eingesetzt.

### 4.4.1.1 Installation / Konfiguration

Telegraf wird zusammen mit InfluxDB installiert und muss nur noch konfiguriert werden. Eine Konfigurationsdatei kann initial mit dem Kommando

---

```
telegraf -sample-config -input-filter mqtt_consumer -output-filter influxdb >
→ telegraf.conf
```

---

erzeugt werden.

In der erzeugten Datei muss das Einlesen der MQTT Queue im Abschnitt "[[inputs.mqtt\_consumer]]" angepasst werden.

---

```
[["inputs.mqtt_consumer"]]
  servers = ["tcp://localhost:1883"]
  topics = [
    "/weather/",
  ]
  topic_tag = "weather"
```

```
qos = 0
connection_timeout = "30s"
max_undelivered_messages = 1000
client_id = "telegraf"
data_format = "json"
tag_keys = ["ID"]
```

---

Auch das Schreiben der Daten in die Influx Datenbank muss im Abschnitt "[[outputs.influxdb]]" konfiguriert werden.

---

```
[[outputs.influxdb]]
urls = ["http://localhost:8086"]
database = "weather_db"
timeout = "5s"
username = "telegraf"
password = "*****"
user_agent = "telegraf"
```

---

Zuletzt wird Telegraf mitgeteilt welche Konfigurationsdatei verwendet werden soll:

---

```
telegraf --config telegraf.conf
```

---

## 4.4.2 InfluxDB

### 4.4.2.1 Installation

Die Installationsquelle ist unterschiedlich für die verschiedenen Linux Distributionen. Wenn, wie in meinem Fall, die Raspian Distribution in der Version 'buster' verwendet wird, kann das benötigte Repository mit folgenden Kommandos hinzugefügt werden:

---

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
sudo apt install apt-transport-https
echo "deb https://repos.influxdata.com/debian buster stable" | sudo tee
→ /etc/apt/sources.list.d/influxdb.list
```

---

Die Installation ist schnell gemacht:

---

```
sudo apt-get update && sudo apt-get install influxdb
sudo systemctl unmask influxdb.service
```

---

Anschliessend muss der Service noch gestartet werden:

---

```
sudo systemctl start influxdb
```

---

#### 4.4.2.2 Datenbank einrichten

Achtung: Standardmässig ist die Influx Datenbank ohne Authentifizierung eingerichtet. In einem offenen Umfeld sollte die standardmässige Authentifizierung eingeschaltet werden [21]

Um eine Datenbank und einen Benutzer einzurichten verwendet man das Kommandozeilen-Tool

---

```
influx
```

---

```
Connected to http://localhost:8086 version 1.7.7
InfluxDB shell version: 1.7.7
> CREATE DATABASE weather_db
> USE weather_db
>CREATE USER telegraf WITH PASSWORD '*****'
>GRANT [READ,WRITE,ALL] ON weather_db TO telegraf
```

---

Eine Spaltendefinition ist nicht nötig, die Spalten werden direkt aus den empfangenen Daten gebildet.

## 4.5 Visualisierung

Die Visualisierung der Daten wird über die Open Source Software Grafana [22] durchgeführt. Mit dieser Software können nicht nur Zeitreihen dargestellt werden, es ist auch möglich bei der Überschreitung von Schwellwerten Alarm auszulösen.

### 4.5.1 Installation

Grafana für den Raspberry Pi kann nicht über ein Repository installiert werden, weshalb die aktuelle Version zuerst mit

---

```
wget
```

---

heruntergeladen und danach entpackt wird.

```
wget
```

```
→ https://github.com/fg2it/grafana-on-raspberry/releases/download/v5.1.4/grafana_5.1.4_armhf.deb
sudo apt-get install -y adduser libfontconfig1
sudo dpkg -i grafana_5.1.4_armhf.deb
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
sudo systemctl enable grafana-server.service
```

## 4.5.2 Grafana Dashboard

Die Darstellung der Messwerte erfolgt in Grafana über Panels, welche in einem Dashboard [23] zusammengefasst werden.

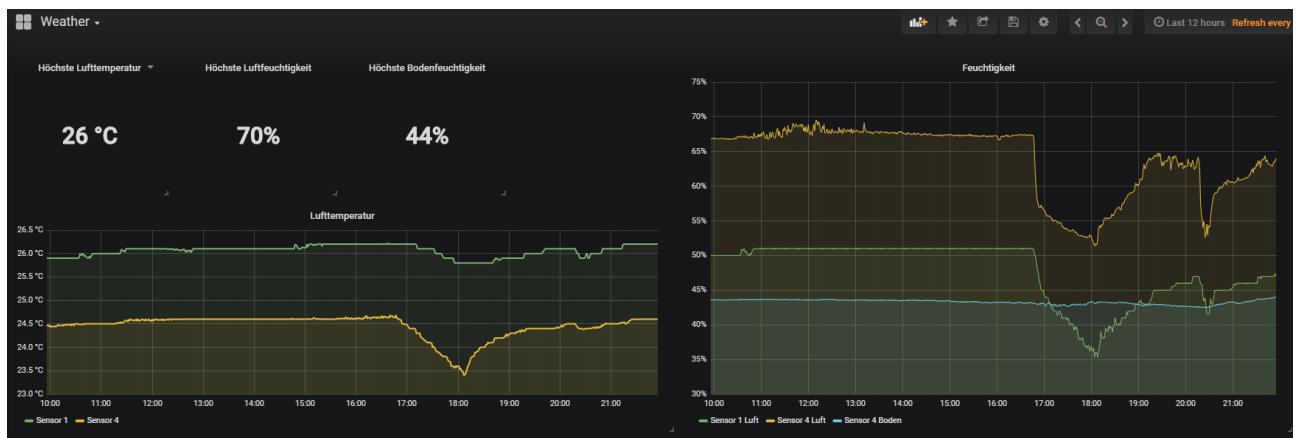
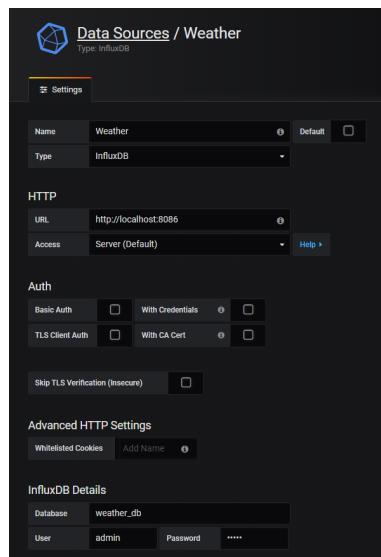


Abbildung 4.3: Grafana Dashboard



Bevor ein Dashboard erstellt wird, muss man die Datenquellen definieren. Hier ist die Konfiguration der Datenquelle, welche aus der lokal installierten InfluxDB und der Datenbank `weather_db` die Messwerte liest.

Abbildung 4.4: Grafana: Konfiguration Influx Datenquelle

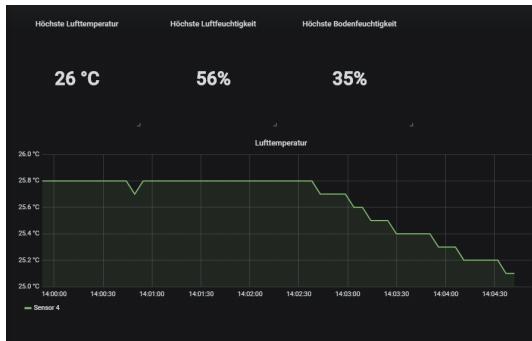


Abbildung 4.5: Grafana Dashboard: Panel Lufttemperatur

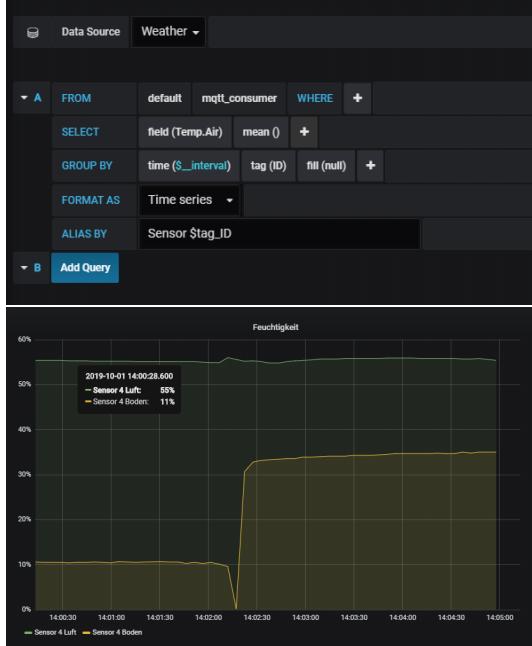


Abbildung 4.6: Grafana Dashboard: Panel Feuchtigkeit

Die Datenbankabfrage für jedes Panel kann direkt in der Benutzeroberfläche von Grafana erstellt werden. Dabei können die Mittelwerte über mehrere Messwerte gebildet werden oder auch nur der maximale Wert in einem bestimmten Zeitintervall bestimmt werden. In dem Weather Dashboard werden von allen drei Mess-Kategorien die höchsten Werte der letzten Tage dargestellt neben den Panels mit dem zeitlichen Verlauf.

Die Legende wird normalerweise aus den Werten der Datenquelle und der Zeitreihe gebildet, was sehr unverständlich sein kann. In der Konfiguration eines Panels kann deshalb im Feld "ALIAS BY" eine alternative Bezeichnung angegeben werden. In diesem Beispiel ist es

Sensor \$tag\_ID

was dazu führt, dass jeweils der Text "Sensor" und die Id des Sensors verwendet wird.

## 4.6 Ersatz für Kafka

Ursprünglich war geplant zwischen dem MQTT Broker und InfluxDB den Apache Kafka Message Bus mit Streams zu verwenden. Die Installation von Zookeeper und Kafka auf dem Raspberry Pi verlief problemlos und die Software lies sich starten und testen. Um die Nachrichten aus dem MQTT Topic zu lesen und in ein Kafka Topic zu schreiben kann man das Kafka Connect Framework mit dem Kafka Connect MQTT Connector verwenden [25]. Allerdings ist der MQTT Connector nicht frei verfügbar (im Gegensatz zu Kafka selber) und kann kostenlos nur mit einer 30 Tage Testlizenz betrieben werden. Zudem erwies sich Kafka Connect als sehr Ressourcen intensiv. In der Default Einstellung wird von 2GB Ram alleine für Kafka Connect ausgegangen. Da der Raspberry Pi nur 1GB Ram besitzt musste ich diesen Wert reduzieren. Der Connector lies sich danach starten, es dauert aber sehr lange bis der MQTT Adapter bereit war. Aufgrund dieser Erfahrungen habe ich nach alternativen gesucht und bin dabei bei Telegraf gelandet. Meine Erfahrungen damit sind ausgesprochen positiv, der Speicherbedarf ist viel geringer und die Geschwindigkeit höher.

Anstatt den MQTT Connector zu verwenden wäre es natürlich auch mit einem eigenen Programm oder Skript möglich gewesen die Nachrichten aus dem MQTT Broker zu lesen und in ein Kafka Topic zu

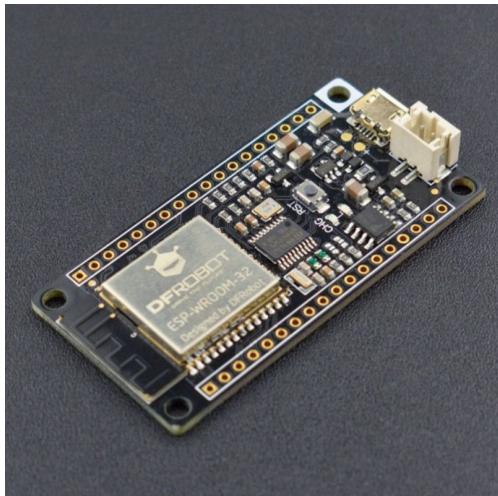
schreiben. Dies hätte jedoch Zeit des Projektes verbraucht welche ich lieber für andere Punkte aufwenden wollte.

## 5 Fazit

Die in diesem CAS behandelten Komponenten Mosquitto und Influx können sehr gut für IoT Projekte eingesetzt werden. Insbesondere die gute Performance auf der sehr schwachen Hardware in diesem Projekt (Raspberry Pi 2) war überraschend. Als negative Erfahrung gilt für mich Kafka Connect und insbesondere der MQTT Connector. Nicht nur die Installation war mühsam, auch die Geschwindigkeit lag nicht im akzeptablen Bereich. Der Unterschied zwischen dem in Go geschriebenen Telegraf und der Java Applikation Kafka Connect war massiv.

# 6 Verwendete Bauteile

## 6.1 ESP32 Mikrokontroller



Als Basis für alle Sensoren ist der Firebeetle<sup>1</sup>[11] sehr gut geeignet. Leider sind noch nicht alle Bibliotheken, welche für den Vorgänger ESP8266 existieren, für den ESP32 adaptiert worden. Vor einem Projekt sollte man also prüfen ob die benötigten Bibliotheken in der richtigen Version vorhanden sind.

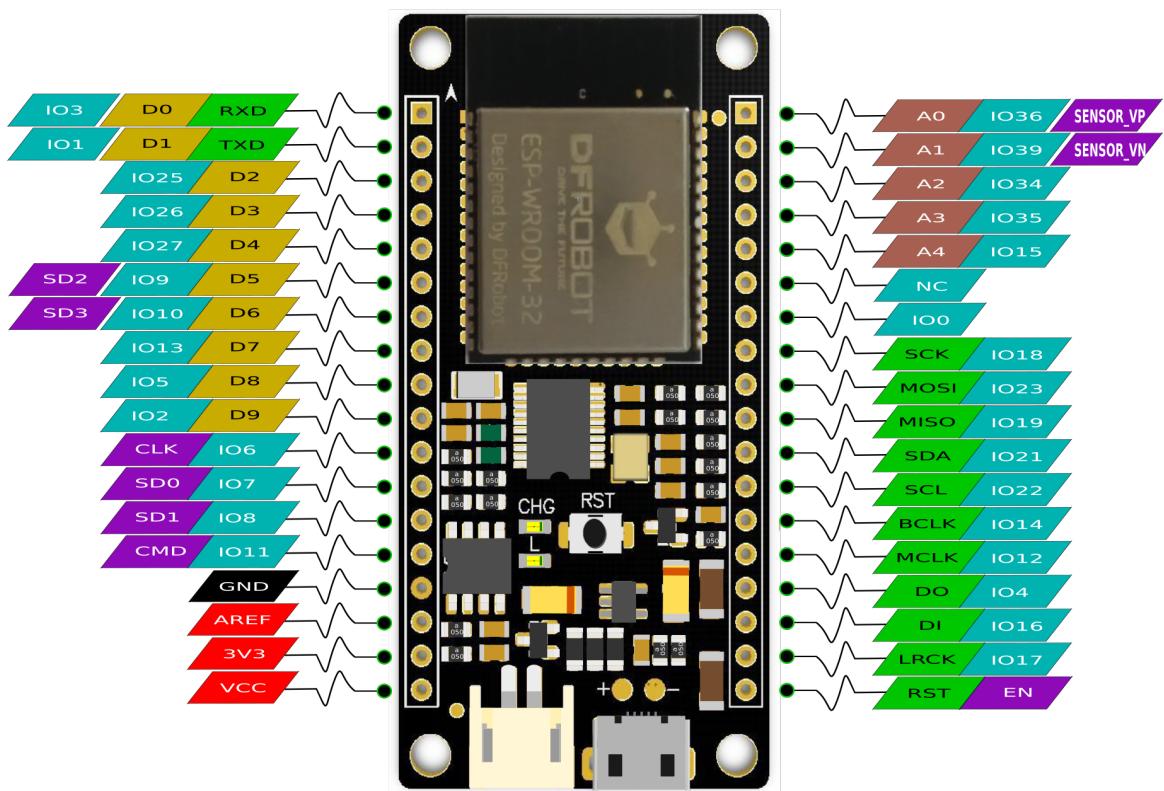


Abbildung 6.1: Anschlüsse des Firebeetle

<sup>1</sup><https://www.bastelgarage.ch/firebeetle-esp32-iot-mikrocontroller-mit-wifi>

## 6.2 Arduino



Abbildung 6.2: Arduino Mega

Ein Arduino Mega<sup>2</sup> zusammen mit dem Ethernet Shield (Shield = Erweiterung) W5100<sup>3</sup> bildet das MQTT Gateway. Für die Kommunikation mit den Sensoren ist ein NRF24 Funkmodul zuständig [6.4.3].

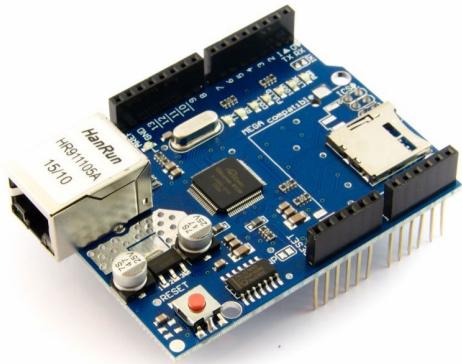


Abbildung 6.3: Ethernet Shield für den Arduino

## 6.3 Raspberry Pi



Abbildung 6.4: Raspberry Pi 2

Die Software für die Speicherung der Daten und die Visualisierung wurde auf einem Raspberry Pi 2 installiert<sup>4</sup>. Obwohl dieses Modell bereits älter ist und nur 1GB Ram besitzt ist die Leistung für dieses Projekt ausreichend.

<sup>3</sup><https://store.arduino.cc/arduino-mega-2560-rev3>

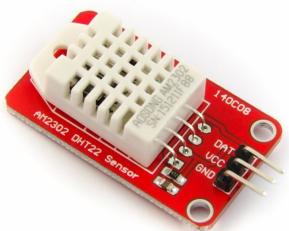
<sup>3</sup><https://www.bastelgarage.ch/ethernet-shield-spi-w5100-für-arduino>

<sup>4</sup><https://www.pi-shop.ch/raspberry-pi-2-model-b-v1-2-1-gb-ram>

## 6.4 Sensoren

### 6.4.1 Temperatur und Luftfeuchtigkeit

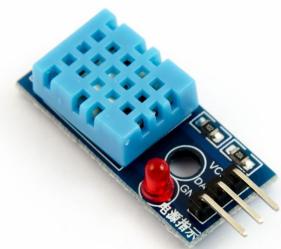
#### 6.4.1.1 DHT22



Der DHT22<sup>5</sup> ist eine Kombination aus Lufttemperatur und Luftfeuchtigkeitssensor. Gegenüber dem DHT11 ist er präziser, kostet aber auch mehr.

Abbildung 6.5: Temperatur und Luftfeuchtigkeitsmesser  
DHT22

#### 6.4.1.2 DHT11



Als Alternative zu dem oben genannten DHT22 gibt es eine billigere Variante mit dem Namen DHT11<sup>6</sup>. Der DHT11 Sensor hat eine geringere Auflösung als der DHT22, ist ansonsten aber gleich anwendbar.

Abbildung 6.6: DHT11 Luft- und Feuchtigkeitssensor

### 6.4.2 Bodenfeuchtigkeit



Der Bodenfeuchtesensor V1.2<sup>7</sup> nimmt die Messung kapazitiv vor, d.h. es werden keine leitenden Kontakte in das Erdreich benötigt. Dadurch ist dieser Sensor keinem Verschleiss unterworfen. Die Messung erfolgt analog und muss danach in die gewünschte Masseinheit umgerechnet werden.

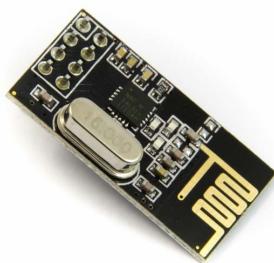
Abbildung 6.7: Bodenfeuchtesensor V1.2

<sup>5</sup><https://www.bastelgarage.ch/dht22-temperatur-und-luftfeuchtigkeitssensor-steckbar>

<sup>6</sup><https://www.bastelgarage.ch/dht11-temperatur-und-luftfeuchtigkeitssensor>

<sup>7</sup><https://www.bastelgarage.ch/bauteile/sensoren/kapazitiver-bodenfeuchtesensor-v1-2>

### 6.4.3 Funk



Das Funkmodul NRF24L01+<sup>8</sup>funkt im 2,4GHz Band und benötigt wenig Strom bei einer guten Reichweite. Der Preis ist sehr niedrig für diesen Baustein, allerdings ist die Anwendung aufwendiger als bei anderen Funkprotokollen.

Abbildung 6.8: NRF24L01+ Funkmodul

---

<sup>8</sup><https://www.bastelgarage.ch/bauteile/funk-wireless-lora/nrf24l01-wireless-funk-modul-2-4ghz>

# Abbildungsverzeichnis

0.1 Systemaufbau Funknetzwerk	1
0.2 Systemaufbau	1
2.1 Firebeetle mit dem ESP32 Mikrokontroller	3
2.2 Verdrahtung der Komponenten	5
2.3 Funkverbindung	5
2.4 NRF24L01+ Funkmodul	5
2.5 Funkkanäle des NRF24L01	6
2.6 Addressierung Funkmodule	6
2.7 Verzeichnisstruktur Sensor	7
2.8 Speicher-Aufteilung ESP32	7
2.9 Übertragen der Messdaten in mehreren Paketen	8
3.1 Verdrahtung der Komponenten	10
3.2 Sensor Datensatz in JSON Format	12
4.1 Verwendete Software	13
4.2 Übersicht Influx Software	15
4.3 Grafana Dashboard	18
4.4 Grafana: Konfiguration Influx Datenquelle	18
4.5 Grafana Dashboard: Panel Lufttemperatur	19
4.6 Grafana Dashboard: Panel Feuchtigkeit	19
6.1 Anschlüsse des Firebeetle	21
6.2 Arduino Mega	22
6.3 Ethernet Shield für den Arduino	22
6.4 Raspberry Pi 2	22
6.5 Temperatur und Luftfeuchtigkeitsmesser DHT22	23
6.6 DHT11 Luft- und Feuchtigkeitssensor	23
6.7 Bodenfeuchtesensor V1.2	23
6.8 NRF24L01+ Funkmodul	24

# Tabellenverzeichnis

2.1 Leistung der verschiedenen Funktechniken im Vergleich	4
---	---

# Glossar

**HMAC** Keyed-Hash Message Authentication Code. 6, 7, 10–12, *Glossary*: Keyed-Hash Message Authentication Code

**IoT** Internet of Things. 1, *Glossary*: Internet of Things

**MQTT** Message Queuing Telemetry Transport. 6, *Glossary*: Message Queuing Telemetry Transport

# Linkverzeichnis

- [1] What is HMAC Authentication and why is it useful?  
<https://www.wolfe.id.au/2012/10/20/what-is-hmac-authentication-and-why-is-it-useful>
- [2] ESP32 Arduino: Applying the HMAC SHA-256 mechanism  
<https://techtutorialsx.com/2018/01/25/esp32-arduino-applying-the-hmac-sha-256-mechanism/>
- [3] mbed TLS, Library für den ESP32 welche viele Kryptografische Funktionen implementiert  
<https://tls.mbed.org/>
- [4] Standard Bibliothek sowohl für den Arduino wie auch den ESP32 welche die Kommunikation mit anderen Komponenten, in diesem Fall die einzelnen Sensor Module, übernimmt  
<https://www.arduino.cc/en/Reference/SPI>
- [5] Bibliothek für den nRF24L01 Baustein welche sowohl auf dem Arduino als auch dem ESP32 verwendbar ist  
<https://github.com/nRF24/RF24>
- [6] Online HMAC Code Generator  
<https://www.freeformatter.com/hmac-generator.html#ad-output>
- [7] NRF24L01+ Funkmodul Treiber für Arduino und ESP32  
<http://tmrh20.github.io/RF24/index.html>
- [8] Bibliothek für die beiden Sensoren DHT11 und DHT22 sowohl für Arduino als auch den ESP32  
<https://github.com/adafruit/DHT-sensor-library>
- [9] Herstellerseite Espressif für den ESP32  
<https://www.espressif.com/en/products/hardware/esp32/overview>
- [10] Arduino Herstellerseite mit der gleichnamigen IDE und vielen Bibliotheken und Tutorials  
<https://www.arduino.cc/>
- [11] Herstellerseite des Firebeetle Mikrokontrollers  
<https://www.dfrobot.com/product-1590.html>

- [12] Erweiterung für die Arduino IDE um Dateien in den Flash Speicher des ESP32 zu laden  
<https://github.com/me-no-dev/arduino-esp32fs-plugin>
- [13] Arduino Client for MQTT  
<https://github.com/knolleary/pubsubclient>
- [14] Arduino Ethernet library  
<https://www.arduino.cc/en/Reference/Ethernet>
- [15] Arduino driver for nRF24L01 2.4GHz Wireless Transceiver  
<https://github.com/maniacbug/RF24>
- [16] SHA1 / SHA256 and HMAC-SHA1 / HMAC-SHA256 Hash library  
<http://spaniakos.github.io/Cryptosuite/>
- [17] Open Source MQTT Broker Mosquitto  
<https://mosquitto.org/>
- [18] Server Agent welcher Daten aus verschiedenen Quellen sammelt und weiterleitet  
<https://www.influxdata.com/time-series-platform/telegraf/>
- [19] MQTT Plugin für Telegraf  
[https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mqtt\\_consumer](https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mqtt_consumer)
- [20] InfluxDB, Open Source Zeitreihendatenbank  
<https://www.influxdata.com/products/influxdb-overview/>
- [21] Authentication and authorization in InfluxDB  
[https://docs.influxdata.com/influxdb/v1.7/administration/authentication\\_and\\_authorization/](https://docs.influxdata.com/influxdb/v1.7/administration/authentication_and_authorization/)
- [22] Datenvisualierungs-Software Grafana  
<https://grafana.com/>
- [23] Grafana Graph Panel  
<https://grafana.com/docs/features/panels/graph/>
- [24] Design of Wireless Sensors for IoT with Energy Storage and Communication Channel Heterogeneity  
<https://www.mdpi.com/1424-8220/19/15/3364/pdf>
- [25] Kafka Connect MQTT Connector  
<https://docs.confluent.io/current/connect/kafka-connect-mqtt/>
- [26] ESP32 MQTT Library  
<https://github.com/espressif/esp-mqtt>
- [27] Mosquitto SSL Configuration -MQTT TLS Security  
<http://www.steves-internet-guide.com/mosquitto-tls/>