



IoT Erfassung von und Darstellung Sensordaten

CAS Big Data

Semesterarbeit

Studienrichtung:

Autor:

Dozent:

Experte:

Datum:

CAS Big Data

Marc Habegger

Max Kleiner

1. Oktober 2019

Inhaltsverzeichnis

1 Einleitung	2
1.1 Übersicht	2
1.2 Hardware	2
2 Sensor Plattform	3
2.1 Mikrokontroller	3
2.2 Sensoren	3
2.3 Aufbau	4
2.4 Funkverbindung mit dem MQTT Gateway	4
2.4.1 Absicherung der Funkverbindung	5
2.5 Software	6
2.5.1 Verwendete Bibliotheken	6
2.5.2 Konfiguration eines Sensors	6
2.5.3 Datenübertragung	7
3 IoT Gateway	9
3.1 Übersicht	9
3.2 Hardware	9
3.2.1 Aufbau	9
3.2.2 Zugriffsschutz	10
3.3 Software	10
3.3.1 Konfiguration	10
4 Speicherung und Visualisierung	12
4.1 Übersicht	12
4.2 Hardware	12
4.3 MQTT Broker	12
4.3.1 Installation	13
4.4 Datenspeicherung	13
4.4.1 Telegraf	13
4.4.1.1 Installation	14
4.4.2 InfluxDB	14
4.4.2.1 Installation	14
4.4.2.2 Datenbank einrichten	14
4.5 Visualisierung	15
4.5.1 Installation	15
4.5.2 Grafana Dashboard	15
5 Verwendete Bauteile	18
5.1 ESP32 Mikrokontroller	18

5.2 Sensoren	19
5.2.1 Temperatur und Luftfeuchtigkeit	19
5.2.1.1 DHT22	19
5.2.1.2 DHT11	19
5.2.2 Bodenfeuchtigkeit	19
5.2.3 Funk	20
Index	25

Management Summary

Das Internet der Dinge (Internet of Things) 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.

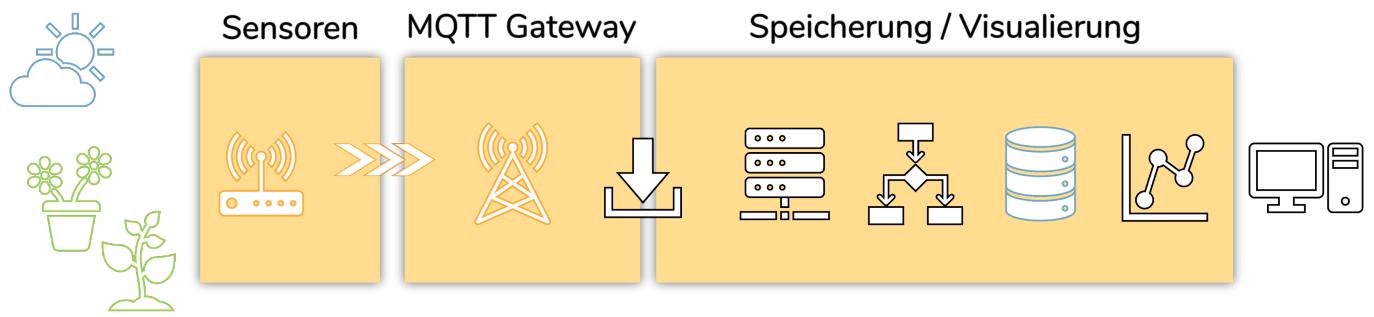


Abbildung 0.1: Systemaufbau Funknetzwerk

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

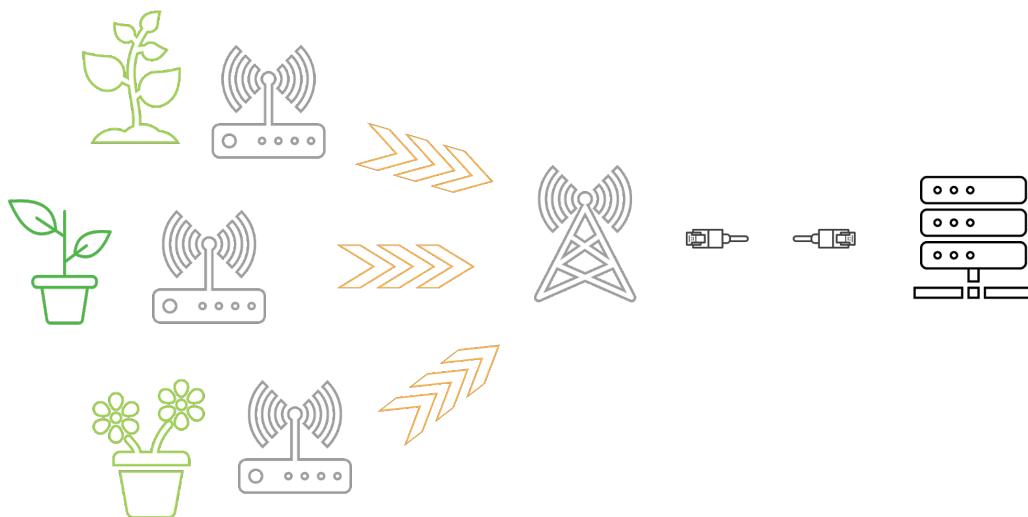


Abbildung 0.2: Systemaufbau

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 gesammte Code und die Dokumentation dieser Arbeit ist auf Github unter folgendem Link zu finden: <https://github.com/habis-git/CAS-BGD-SemesterArbeit>

1.1 Übersicht

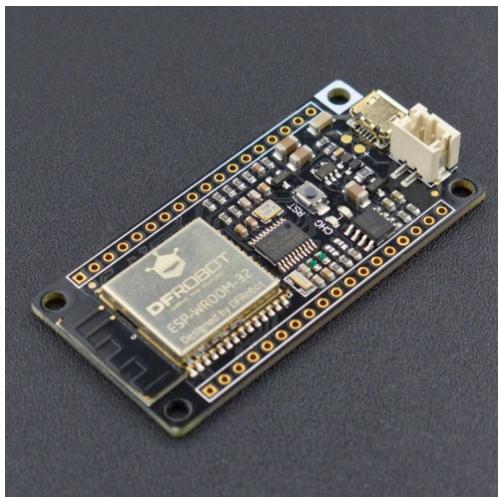
Die Komponenten wurden durch das Vorhandensein von Bibliotheken und Treibern sowie die leichte Beschaffung auch innerhalb der Schweiz bestimmt. Der Verzicht auf kommerzielle Systeme erlaubt eine grosse Flexibilität in der Kombination von Funktionalitäten.

1.2 Hardware

2 Sensor Plattform

2.1 Mikrokontroller

Ein sehr beliebter Mikrokontroller ist der ESP32 von Espressif [9] welcher durch grossen Funktionsumfang und einen recht geringen 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.¹



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.

Abbildung 2.1: Firebeetle mit dem ESP32 Mikrokontroller

2.2 Sensoren

Es existiert eine grosse Anzahl an verschiedenen Sensoren welche auch für Privatanwendern 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 [5.2.1.2] und DHT22 [5.2.1.1]
- Kapazitiver Bodenfeuchtesensor V1.2 [5.2.2]

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

2.3 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 welche 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 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 hab. Falls andere Pins verwendet werden so können die Variablen am Anfang des Programms 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.

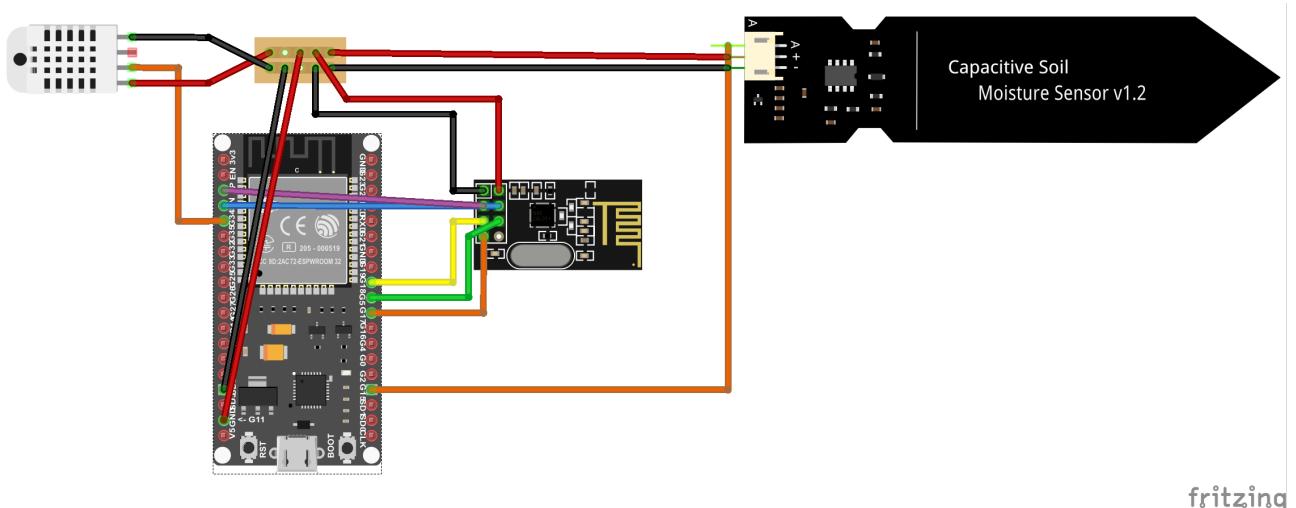


Abbildung 2.2: Verdrahtung der Komponenten

2.4 Funkverbindung mit dem MQTT Gateway

Da die Sensoren nicht WLAN als Funktechnik verwenden können sie nicht direkt auf den MQTT Server verbinden und Nachrichten senden. Diese Aufgabe übernimmt das MQTT-Gateway welches auf der einen Seite ebenfalls mit einem NRF24 [2.4] 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

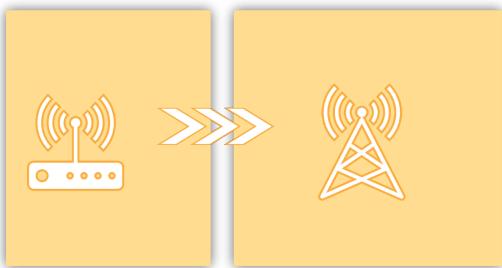


Abbildung 2.3: Funkverbindung



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 [2.5.2].



Abbildung 2.5: Funkkanäle des NRF24L01

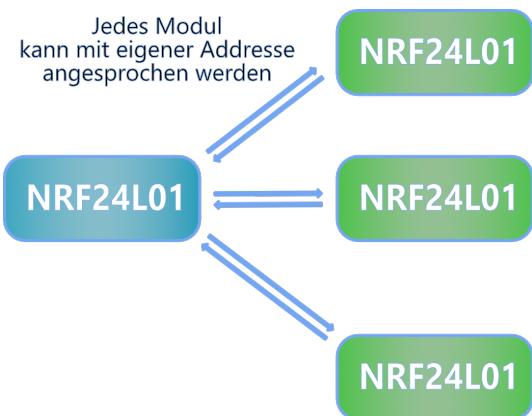


Abbildung 2.6: Addressierung Funkmodule

2.4.1 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

3A51266ABD432C1FB797DEEE215FD16E42BAB627F104BCB6F4169EDE4544582:4;25.1;71.1;10.2

und die Nachricht mit HMAC und den Messwerten wird an das Internet of Things (IoT) Gateway geschickt.

2.5 Software

2.5.1 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) 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.**

2.5.2 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 werden so müssen die Konfigurationsparameter nur einmal festgelegt werden und ändern sich, wenn überhaupt, nur selten.

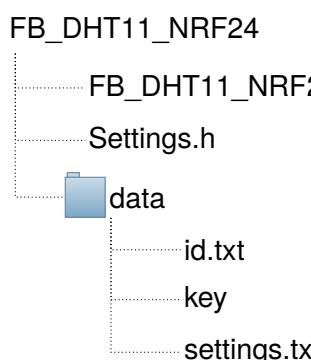


Abbildung 2.7: Verzeichnisstruktur Sensor

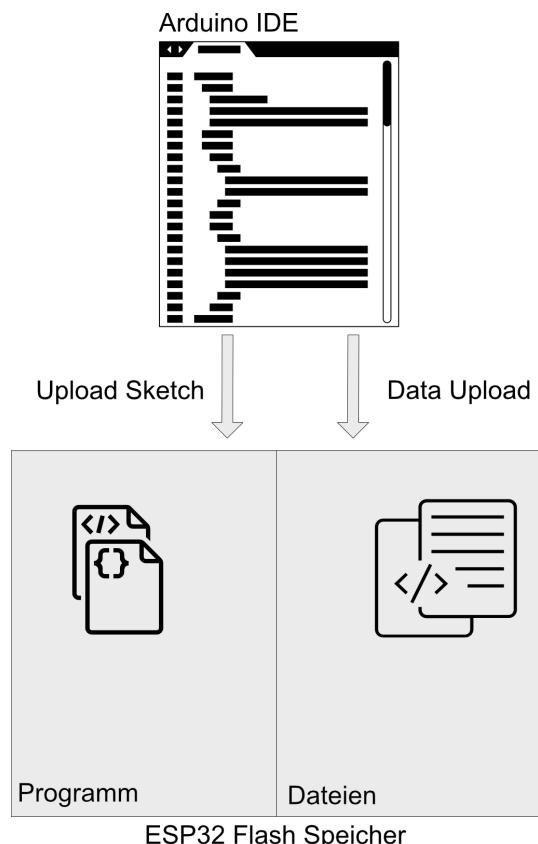


Abbildung 2.8: Speicher-Aufteilung ESP32

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'

2.5.3 Datenübertragung

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

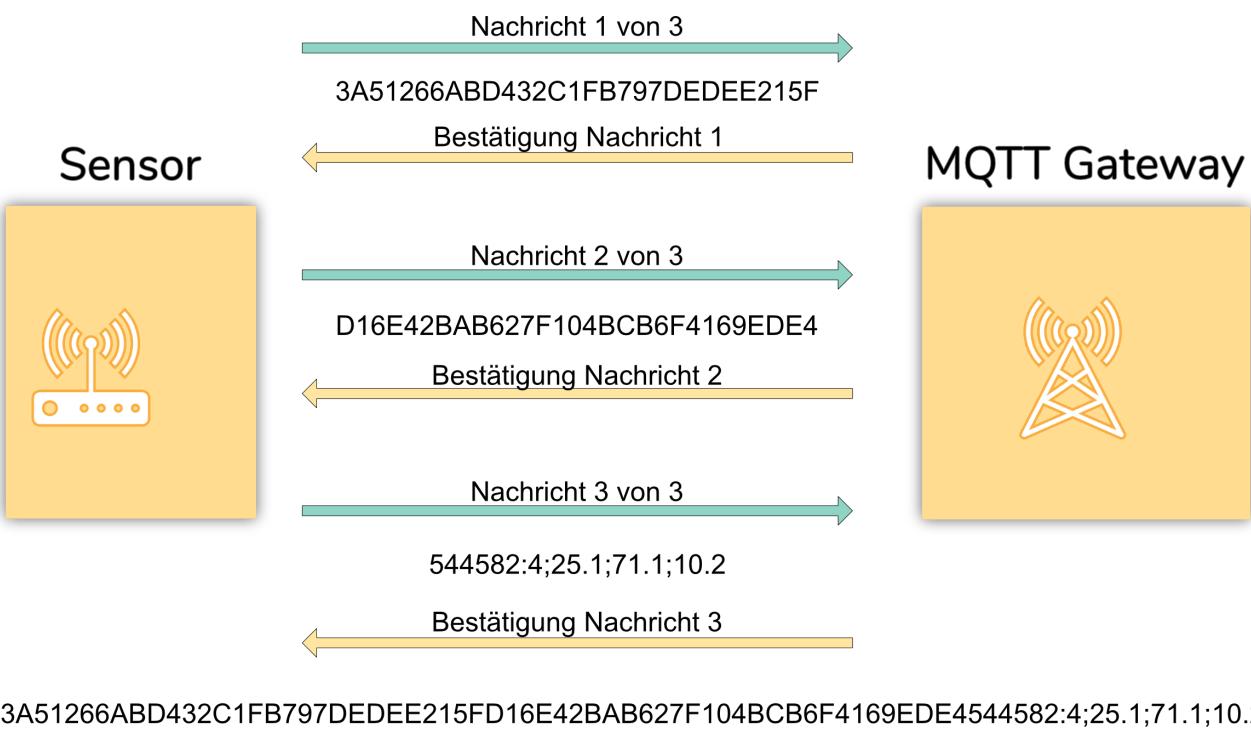


Abbildung 2.9: Übertragen der Messdaten in mehreren Paketen

```
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 retry++;
22 }
23 radio->stopListening();
24 }
```

3 IoT Gateway

3.1 Übersicht

Die Aufgabe des IoT 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 Mega¹ welcher mit einem Ethernet Shield² kombiniert wird. Die Funkverbindung wird wie bei den Sensoren durch ein NRF24 [2.4] Modul hergestellt.

3.2.1 Aufbau

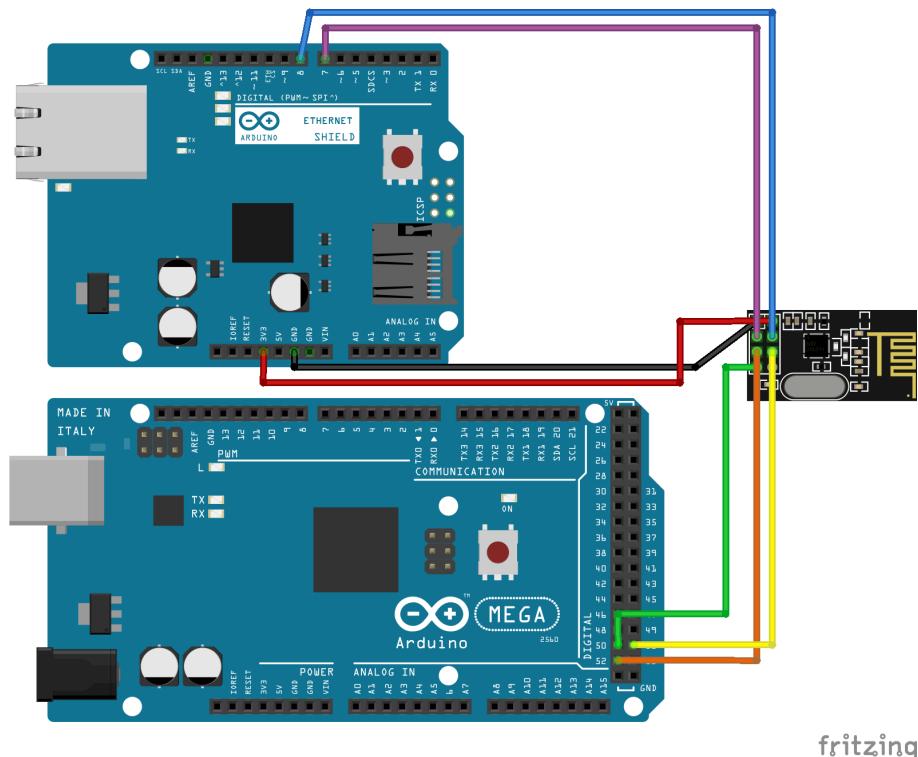


Abbildung 3.1: Verdrahtung der Komponenten

¹<https://store.arduino.cc/arduino-mega-2560-rev3>

²[https://www.bastelgarage.ch/ethernet-shield-spi-w5100-für-arduino](https://www.bastelgarage.ch/ethernet-shield-spi-w5100-fur-arduino)

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 [2.4.1].

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.

3.3 Software

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}",  
         SensorId, Temperature, Humidity, SoilHumidity);  
3 if (!client.connected()) {  
4     reconnect();  
5 }  
6 if (client.connected()) {  
7     Serial.print("Send data to topic '/weather/' ");  
8     client.publish("/weather/", json);  
9     Serial.println(json);  
10 }
```

3.3.1 Konfiguration

Das Gateway benötigt für die Berechnung des HMAC den gleichen Schlüssel wie der Sensor. Leider bietet der Arduino nicht die Möglichkeit Dateien hochzuladen. Der benötigte geheime Schlüssel wird deshalb in der Datei keyFile.h als String definiert. So kann der Code weiterhin in dem Versionskontrollsystem 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.

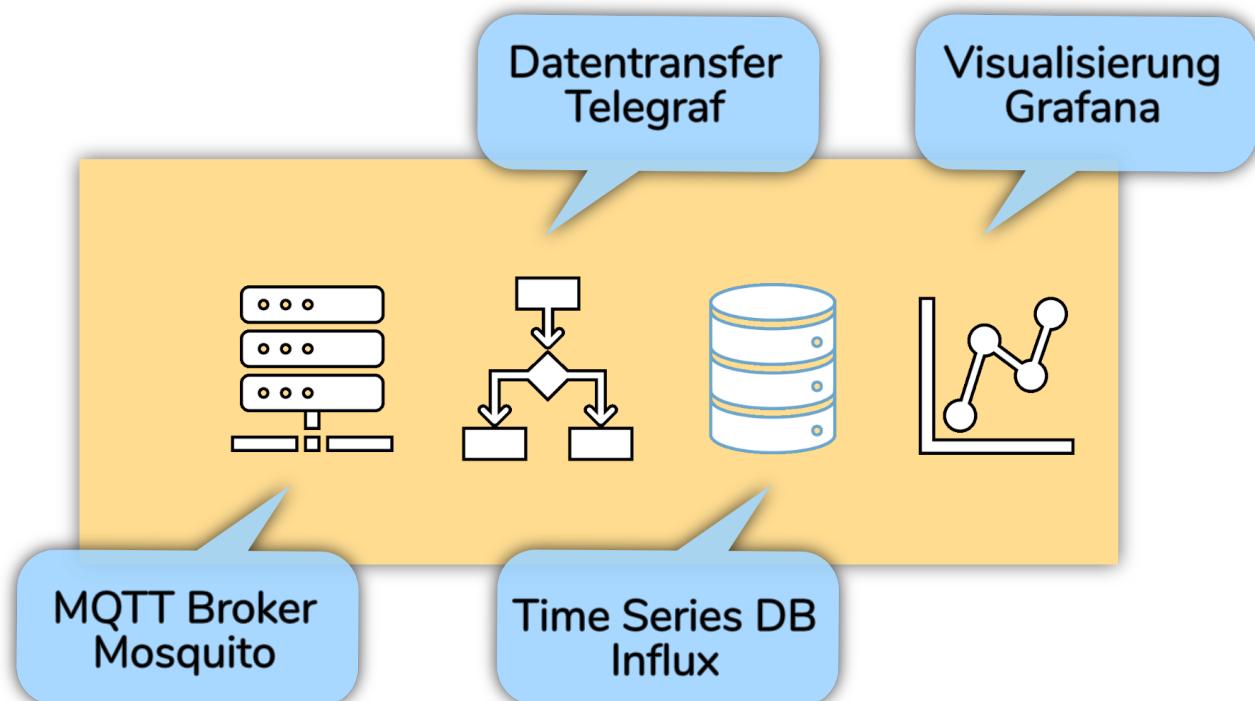


Abbildung 4.1: Verwendete Software

MQTT Broker Die von dem MQTT-Gateway geschickten Nachrichten werden durch Mosquitto [13] 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 [14] mit dem MQTT Plugin benutzt [15]

4.2 Hardware

4.3 MQTT Broker

Als MQTT Broker wird die Open Source Software Mosquitto [13] 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
```

4.4 Datenspeicherung

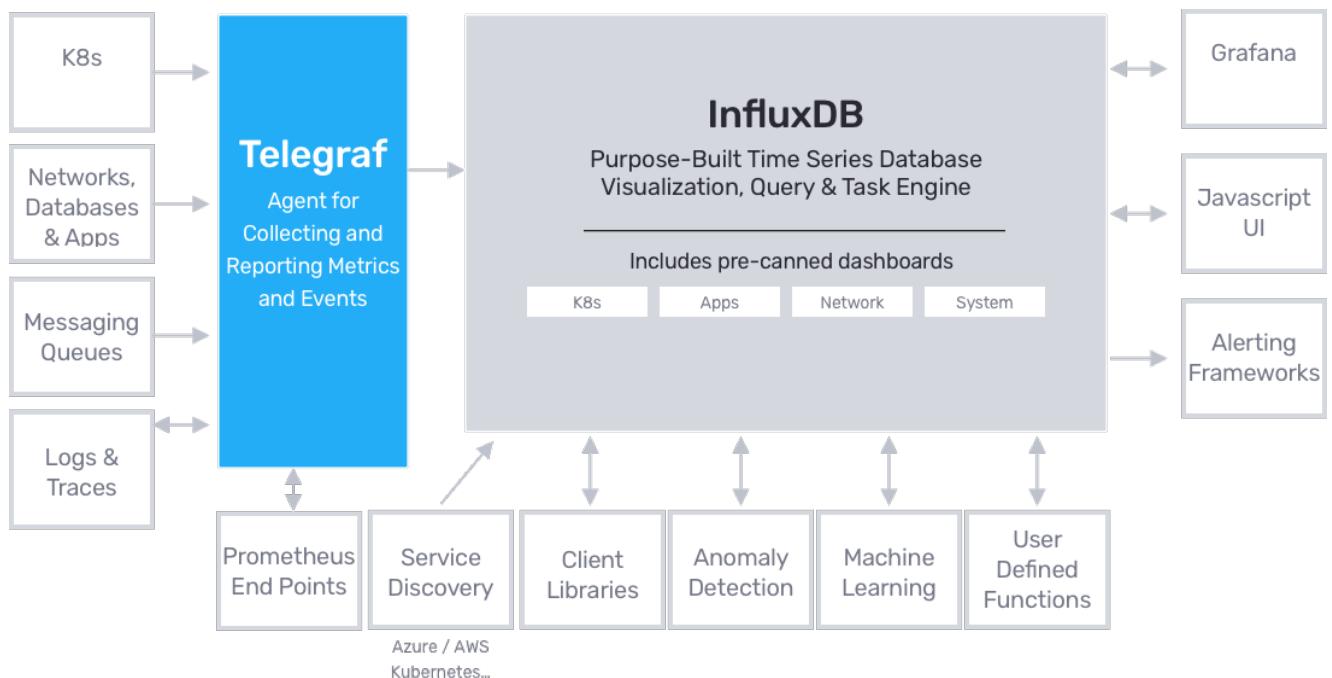


Abbildung 4.2: Übersicht Influx Software

Quelle: www.influxdata.com

4.4.1 Telegraf

Um die Daten aus dem Mosquitto Topic zu lesen und in die Datenbank zu schreiben wird die Software Telegraf [14] eingesetzt.

4.4.1.1 Installation

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 danach schnell gemacht:

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

Danach 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 [17]

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 [18] durchgeführt.

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
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 [19] zusammengefasst werden.



Abbildung 4.3: Grafana Dashboard

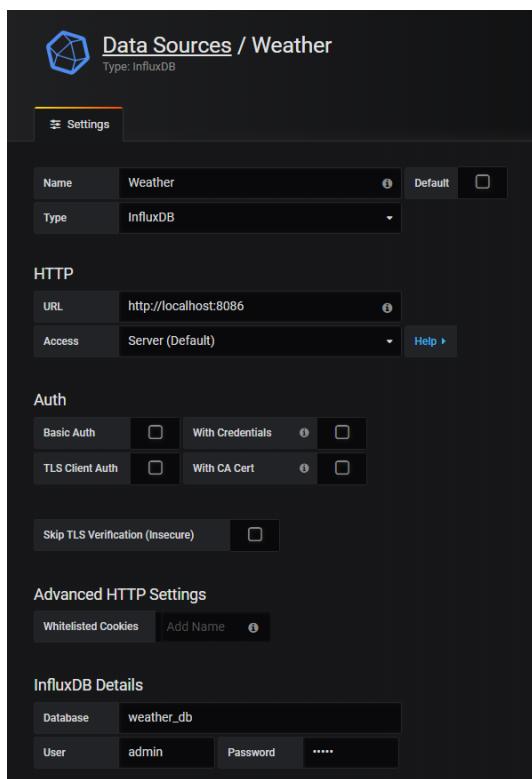


Abbildung 4.4: Grafana: Konfiguration Influx Datenquelle

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.

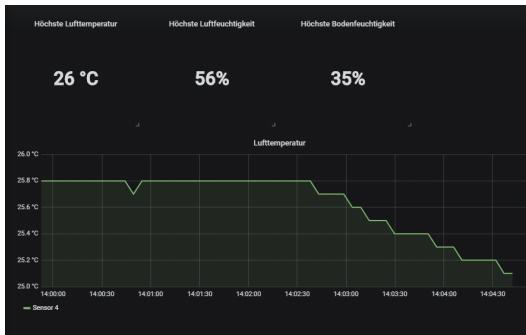
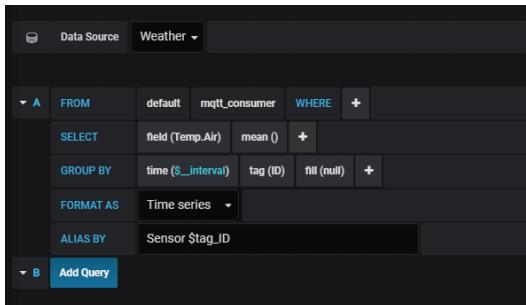


Abbildung 4.5: Grafana Dashboard: Panel Lufttemperatur



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.

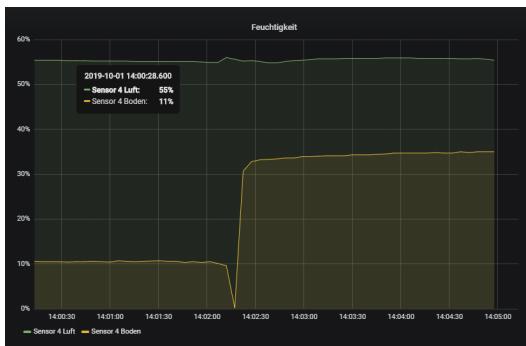


Abbildung 4.6: Grafana Dashboard: Panel Feuchtigkeit

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 ÄLIAS BY eine alternative Bezeichnung angegeben werden. In diesem Beispiel ist es

Sensor \$tag_ID

was dazu führt dass jeweils der Text SSensor und die ID des Sensors verwendet wird.

5 Verwendete Bauteile

5.1 ESP32 Mikrokontroller

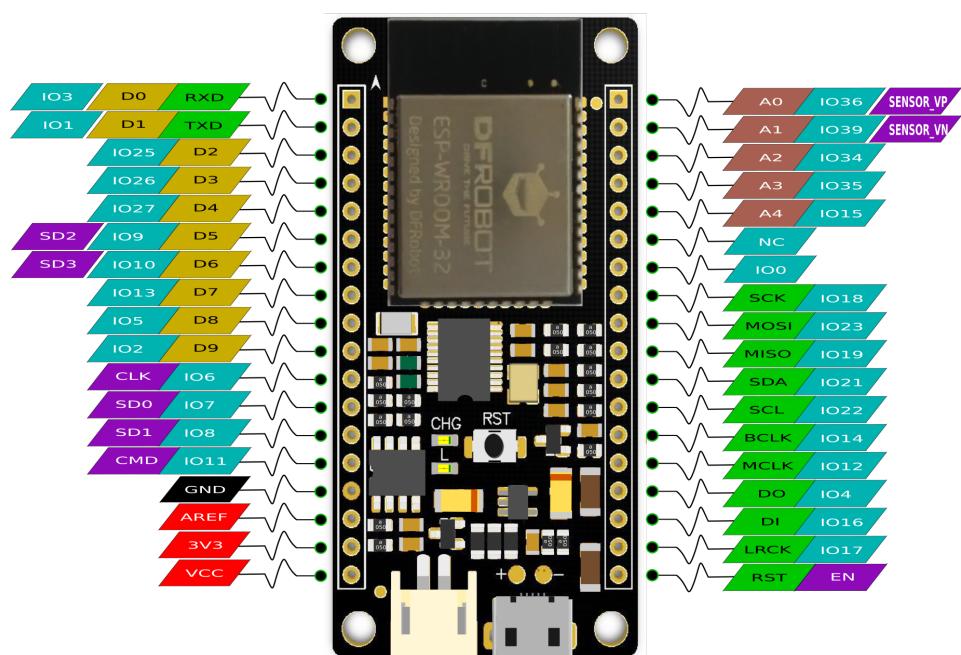
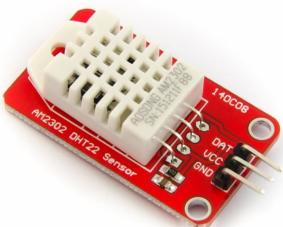


Abbildung 5.1: Anschlüsse des Firebeetle

5.2 Sensoren

5.2.1 Temperatur und Luftfeuchtigkeit

5.2.1.1 DHT22



DHT22 im Onlineshop ¹

Abbildung 5.2: Temperatur und Luftfeuchtigkeitsmesser
DHT22

5.2.1.2 DHT11

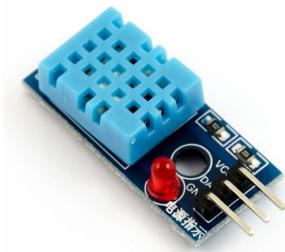


Abbildung 5.3: DHT11 Luft- und Feuchtigkeitssensor

Als Alternative zu dem oben genannten DHT22 gibt es eine billigere Variante mit dem Namen DHT11². Der DHT11 Sensor hat eine geringere Auflösung als der DHT22, ist ansonsten aber gleich verwendbar.

5.2.2 Bodenfeuchtigkeit



Abbildung 5.4: Kapazitiver Bodenfeuchtesensor V1.2

¹<https://www.bastelgarage.ch/dht22-temperatur-und-luftfeuchtigkeitssensor-steckbar>

²<https://www.bastelgarage.ch/dht11-temperatur-und-luftfeuchtigkeitssensor>

Bodensensor im Onlineshop³

5.2.3 Funk

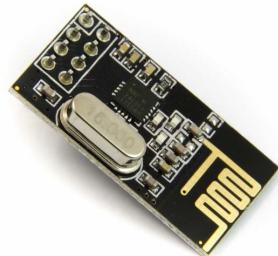


Abbildung 5.5: NRF24L01+ Funkmodul

NRF24L01+⁴

³<https://www.bastelgarage.ch/bauteile/sensoren/kapazitiver-bodenfeuchtesensor-v1-2>

⁴<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	4
2.3 Funkverbindung	5
2.4 NRF24L01+ Funkmodul	5
2.5 Funkkanäle des NRF24L01	5
2.6 Addressierung Funkmodule	5
2.7 Verzeichnisstruktur Sensor	6
2.8 Speicher-Aufteilung ESP32	6
2.9 Übertragen der Messdaten in mehreren Paketen	7
3.1 Verdrahtung der Komponenten	9
3.2 Sensor Datensatz in JSON Format	10
4.1 Verwendete Software	12
4.2 Übersicht Influx Software	13
4.3 Grafana Dashboard	16
4.4 Grafana: Konfiguration Influx Datenquelle	16
4.5 Grafana Dashboard: Panel Lufttemperatur	17
4.6 Grafana Dashboard: Panel Feuchtigkeit	17
5.1 Anschlüsse des Firebeetle	18
5.2 Temperatur und Luftfeuchtigkeitsmesser DHT22	19
5.3 DHT11 Luft- und Feuchtigkeitssensor	19
5.4 Kapazitiver Bodenfeuchtesensor V1.2	19
5.5 NRF24L01+ Funkmodul	20

Tabellenverzeichnis

Glossar

HMAC Keyed-Hash Message Authentication Code. 5–7, 10, *Glossary: Keyed-Hash Message Authentication Code*

Internet of Things deutsch Internet der Dinge, Bezeichnet ein loses Netzwerk in welcher beliebige elektronische Geräte untereinander vernetzt werden. Wir häufig im Zusammenhang mit Sensornetzwerken verwendet.
https://de.wikipedia.org/wiki/Internet_der_Dinge. 1

IoT Internet of Things. 5, *Glossary: Internet of Things*

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] Open Source MQTT Broker Mosquitto
<https://mosquitto.org/>
- [14] Server Agent welcher Daten aus verschiedenen Quellen sammelt und weiterleitet
<https://www.influxdata.com/time-series-platform/telegraf/>
- [15] MQTT Plugin für Telegraf
https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mqtt_consumer

- [16] InfluxDB, Open Source Zeitreihendatenbank
<https://www.influxdata.com/products/influxdb-overview/>
- [17] Authentication and authorization in InfluxDB
https://docs.influxdata.com/influxdb/v1.7/administration/authentication_and_authorization/
- [18] Datenvisualierungs-Software Grafana
<https://grafana.com/>
- [19] Grafana Graph Panel
<https://grafana.com/docs/features/panels/graph/>