# Projekt-Dokumentation: Dateninfrastruktur von Moki-Pro

## Umsetzung der Infrastruktur

Dieses Dokument beschreibt die Umsetzung der Infrastruktur zur Datenerfassung. Dabei werden Daten von Sensoren an der Schleifmaschine (Ultrasonic) erfasst, in InfluxDB gespeichert und in Grafana visualisiert.

### Überblick über die Infrastruktur

Die Abbildung 1 zeigt den strukturierten Prozessfluss innerhalb der entwickelten Infrastruktur:

Ein Bild, das Text, Screenshot, Schrift, Aufdruck enthält.

Automatisch generierte Beschreibung

Abbildung 1 Prozessfluss

1. **Raspberry Pi**: Erfasst Daten von den angeschlossenen Sensoren.
2. **Python-Skript**: Bereitet die Sensordaten auf dem Raspberry Pi auf.
3. **MQTT-Client**: Überträgt die aufbereiteten Daten an den MQTT-Broker.
4. **MQTT-Broker**: Empfängt die Daten und leitet sie an Telegraf weiter.
5. **Telegraf**: Verarbeitet die empfangenen Daten und speichert sie in InfluxDB.
6. **InfluxDB**: Archiviert die Messwerte.
7. **Grafana**: Visualisiert die gespeicherten Daten in benutzerfreundlichen Dashboards.

### 1.2 Hardwarekomponenten

Die verwendete Hardware umfasst:

* **Raspberry Pi 4 Model B Rev 1.1**
* **(Farbsensor (TCS34725RGB)**: Erfasst den Status der Maschine über Farb-LEDs.)
* **Temperatursensor (DS1820)**: Misst die Temperatur innerhalb der Schleifmaschine.
* **Temperatur- und Feuchtigkeitssensor (DHT22)**: Misst die Umgebungstemperatur und Luftfeuchtigkeit im Labor.

Die Sensoren sind über ein Breadboard mit dem Raspberry Pi verbunden.

### Sensoren-Schaltung

Die Sensoren werden einzeln über ein Breadboard mit dem Raspberry Pi verbunden. Die Schaltpläne für die Sensoren, in den folgenden Abbildungen zusehen, und nähere Details wie die Stromversorgung können auf den jeweiligen Shop-Seiten der Sensoren eingesehen werden (siehe Anhang).

Ein Bild, das Elektronik, Schaltung, Elektronisches Bauteil, Elektrisches Bauelement enthält.

Automatisch generierte Beschreibung

Abbildung 2 Schaltung DHT22

Ein Bild, das Text, Screenshot, Schaltung, Design enthält.

Automatisch generierte Beschreibung

Abbildung 3 Schaltung DS18B20

Ein Bild, das Text, Screenshot, Diagramm, Schaltung enthält.

Automatisch generierte Beschreibung

Abbildung 4 Schaltung TCS34725RGB

Abbildung 5 zeigt die Verdrahtung der Sensoren mit dem Breadboard und dem Raspberry Pi. Hierbei repräsentieren die roten Kabel die Stromversorgung, die schwarzen Kabel den Ground und die gelben sowie blauen Kabel dienen der Datenübertragung.

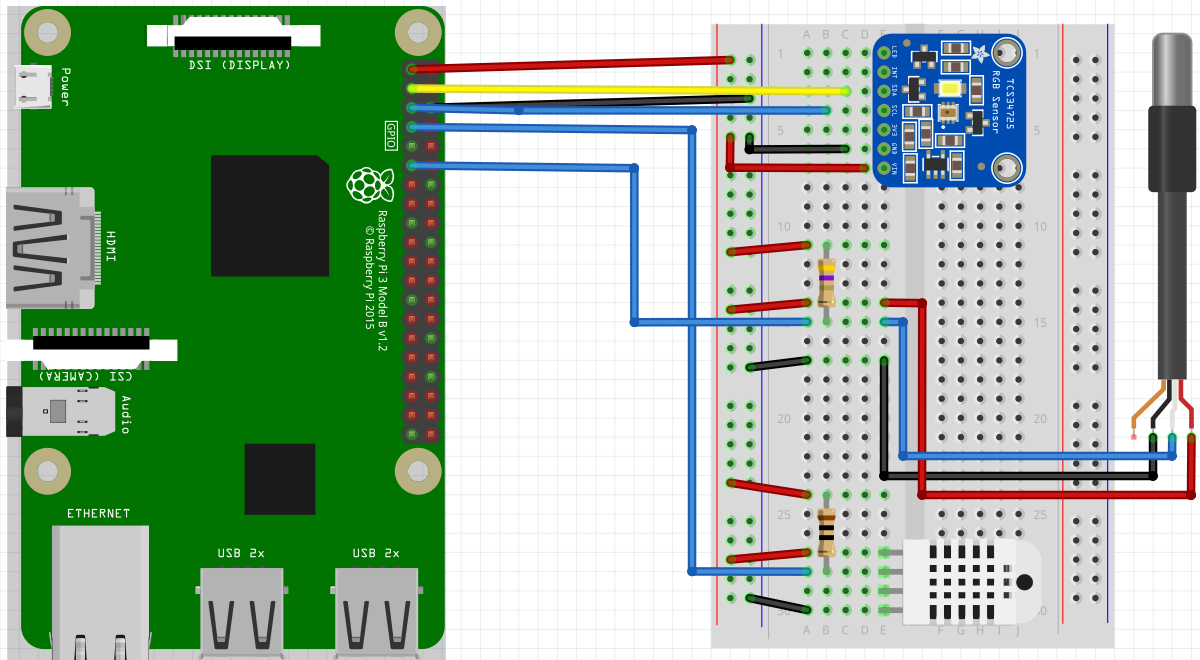


Abbildung 5 Schaltung der Sensoren

### Gehäuse und Kabelmanagement

Um die Hardwarekomponenten geordnet und geschützt aufzubauen, wurde ein speziell für den Raspberry Pi konzipiertes Gehäuse verwendet. Nach der Bohrung von Löchern konnten die Anschlüsse installiert werden. Die Kabel wurden an die Sensoren gelötet und im Gehäuse befestigt. Abbildung 6 zeigt den Versuchsaufbau. Die einzelnen Komponenten sind in den Quellen zu finden.

Ein Bild, das Elektronik, Elektrisches Bauelement, Elektronisches Bauteil, Elektrische Leitungen enthält.

Automatisch generierte Beschreibung

Abbildung 6 Versuchsaufbau

### Softwarekomponenten

Die Softwarekomponenten umfassen:

* **Betriebssystem**: Raspbian GNU/Linux 11 (bullseye)
* **Python-Skript**: Erfasst und überträgt Sensordaten an den MQTT-Broker
* **Docker-compose.yml**: dient zur Definition und Verwaltung von Multi-Container-Docker-Anwendungen
* **MQTT Mosquitto**: Als Broker zur Datenübertragung
* **Telegraf**: Als Datenkollektor und -verarbeiter
* **InfluxDB**: Zur Datenspeicherung
* **Grafana**: Zur Datenvisualisierung

#### Betriebssystem und Systemkonfigurationen

Auf dem Raspberry Pi ist das Betriebssystem Raspbian GNU/Linux 11 (bullseye) installiert. Die Installation erfolgte mittels des Installers von raspberrypi.com auf die SD-Karte.

### Docker-compose.yml

In der Docker-compose.yml werden die Quellen und Konfigurationen der Docker Images festgelegt.

version: "3"

services:

#  mosquitto:

#    image: eclipse-mosquitto:latest

#    container\_name: mosquitto

#    restart: always

#    ports:

#      - "1883:1883"

#      - "9001:9001"

#    networks:

#      - iot

#    volumes:

#      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf

  influxdb:

    image: influxdb

    container\_name: influxdb

    restart: always

    ports:

      - "8086:8086"

    networks:

      - iot

    volumes:

      - influxdb-data:/var/lib/influxdb2

      - influxdb-config:/etc/influxdb2

    environment:

      - DOCKER\_INFLUXDB\_INIT\_MODE=setup

      - DOCKER\_INFLUXDB\_INIT\_USERNAME=Admin

      - DOCKER\_INFLUXDB\_INIT\_PASSWORD=Ka5JY?TP.E!/XbV

      - DOCKER\_INFLUXDB\_INIT\_ORG=EAH

      - DOCKER\_INFLUXDB\_INIT\_BUCKET=Sensordata

      - DOCKER\_INFLUXDB\_INIT\_ADMIN\_TOKEN=lS=wvVM;Duu@-X(U!oZq=p5ix\_Ht)-Xt(2CM\_Bmvwk/zls-BhHP/rpcVq4c?hzRaJpR,/\_7t==TcpBg8\JUzW\_/

  telegraf:

    image: telegraf

    #command: --debug

    container\_name: telegraf

    restart: always

    volumes:

      - ./telegraf.conf:/etc/telegraf/telegraf.conf:ro

    depends\_on:

      - influxdb

    networks:

      - iot

  grafana:

    image: grafana/grafana

    container\_name: grafana

    restart: always

    ports:

      - "3000:3000"

    networks:

      - iot

    volumes:

      - ./grafana-provisioning:/etc/grafana/provisioning

      - grafana-data:/var/lib/grafana

    depends\_on:

      - influxdb

networks:

  iot:

volumes:

  grafana-data:

  influxdb-data:

  influxdb-config:

#### MQTT Mosquitto

MQTT (Message Queueing Telemetry Transport) ist ein weit verbreitetes Kommunikationsprotokoll im IoT. Es ermöglicht die effiziente Übertragung kurzer Nachrichten. Mosquitto dient als MQTT-Broker und Paho als Client, der die Nachrichten an den Broker sendet. Mosquito wurde direkt installiert, und aus den Docker Compose entfernt um möglich Schwierigkeiten beim empfangen der Daten zu verhindern.

MQTT Mosquitto Konfigurition ist durch diesen Befehl zu finden: sudo nano /etc/mosquitto/mosquitto.conf

Mosqitto ist folgendermaßen konfiguriert:

# Place your local configuration in /etc/mosquitto/conf.d/

#

# A full description of the configuration file is at

# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

#pid\_file /run/mosquitto/mosquitto.pid

persistence true

persistence\_location /var/lib/mosquitto/

log\_dest file /var/log/mosquitto/mosquitto.log

include\_dir /etc/mosquitto/conf.d

listener 1883

allow\_anonymous true

Durch die Option „listener 1883“ wird festgelegt, dass Mosquitto auf Port 1883 auf eingehende Verbindungen hört. Die Option „allow\_anonymous true“ ermöglicht es Clients, sich ohne Authentifizierung mit dem Broker zu verbinden. Dies kann für Test- oder Entwicklungsumgebungen nützlich sein, sollte jedoch in Produktionsumgebungen vermieden werden, da es ein Sicherheitsrisiko darstellen kann.

#### Python-Skript zur Sensordatenerfassung

Das Python-Skript sammelt, verarbeitet und sendet Sensordaten mittels MQTT. Es nutzt Bibliotheken wie json, datetime, ntplib, socket, paho.mqtt.client, board, adafruit\_dht und adafruit\_tcs34725 zur Interaktion mit den Sensoren und zur Datenübertragung.

Nicht das aktuelle Skript läuft momentan ohne den Farbsensor (nicht angeschlossen)

import json

from datetime import datetime

import ntplib

import socket

import paho.mqtt.client as mqtt

from paho.mqtt import client as mqtt\_client

import board

import adafruit\_dht

import adafruit\_tcs34725

# Time intervals in seconds

ntp\_update\_interval = 30 \* 60  # 1/2 hours

dht22\_measurement\_interval = 3  # 3 seconds

mqtt\_publish\_interval = 5  # 30 seconds

sleep\_interval = 1  # 3 seconds

# MQTT Broker Settings

mqtt\_broker = "agb-mokipro-db.one.fc.eah-jena.de"

mqtt\_port = 1883

mqtt\_topic = "SensordatenPi1"  # MQTT topic for sensor data and status

# NTP Server Settings

ntp\_server = "time.eah-jena.de"

# Create MQTT Client with automatic reconnection

def connect\_mqtt(client\_id):

    def on\_connect(client, userdata, flags, rc):

        if rc == 0:

            print("Connected to MQTT Broker!")

        else:

            print(f"Failed to connect, return code {rc}")

    def on\_disconnect(client, userdata, rc):

        print(f"Disconnected. Trying to reconnect...")

        client.reconnect()

    client = mqtt\_client.Client(client\_id)

    client.on\_connect = on\_connect

    client.on\_disconnect = on\_disconnect

    while True:

        try:

            client.connect(mqtt\_broker, mqtt\_port)

            return client

        except:

            print("MQTT connection failed. Retrying in 30 seconds.")

            time.sleep(30)  # Retry after 30 seconds

# Function to connect to NTP server

def check\_ntp\_time():

    try:

        c = ntplib.NTPClient()

        response = c.request('time.eah-jena.de', version=3, timeout=5)  # Set timeout to 5 seconds

        ntp\_time = datetime.fromtimestamp(response.tx\_time)

        print("NTP time successfully updated.")

        return ntp\_time

    except socket.timeout as timeout\_err:

        print(f"NTP request timed out: {str(timeout\_err)}")

        return None

    except ntplib.NTPException as ntp\_err:

        print(f"NTP exception: {str(ntp\_err)}")

        return None

    except Exception as e:

        print(f"An unexpected error occurred during NTP time update: {str(e)}")

        return None

# Function to measure humidity and temperature (DHT22 Sensor)

def read\_dht22\_values(pin):

    try:

        dhtDevice = adafruit\_dht.DHT22(pin, use\_pulseio=False)

        dht22\_temperature = dhtDevice.temperature

        dht22\_humidity = dhtDevice.humidity

        return dht22\_temperature, dht22\_humidity

    except RuntimeError as error:

        print("Error reading DHT22 data:", error)

        return None, None

# Function to measure temperature (DS1820 Sensor)

def read\_ds1820\_temperature(device\_path):

    try:

        tempfile = open(device\_path)

        content = tempfile.read()

        tempfile.close()

        temp\_data = content.split("\n")[1].split(" ")[9]

        ds1820\_temperature = float(temp\_data[2:]) / 1000

        return round(ds1820\_temperature, 1)

    except Exception as e:

        print("Error reading DS1820 temperature:", str(e))

        return None

# Function for color sensor (TCS34725 Sensor)

def read\_color\_data():

    i2c = board.I2C()  # uses board.SCL and board.SDA

    sensor = adafruit\_tcs34725.TCS34725(i2c)

    color\_rgb = sensor.color\_rgb\_bytes

    return color\_rgb

def publish\_sensor\_data\_and\_status(client, current\_time):

    # Initialize the status variable

    NTPstatus = "NTP server not reachable, using local time"

    # Check and update NTP time when the script starts

    ntp\_time = check\_ntp\_time()

    if ntp\_time:

        current\_time = ntp\_time

        NTPstatus = "NTP time successfully updated."

    # Set the initial value for last\_ntp\_update\_time

    last\_ntp\_update\_time = current\_time

    dht22\_last\_measurement\_time = None

    dht22\_last\_temperature = None

    dht22\_last\_humidity = None

    last\_successful\_dht22\_measurement\_time = None

    # Set the loop for DHT22 measurement

    last\_dht22\_measurement\_time = current\_time

    # Set the loop for data transmission

    last\_mqtt\_publish\_time = current\_time

    while True:

#        print("Before NTP update check")  # Debugging Option

        current\_time = datetime.now()

        # Check if it's time to update NTP time

        if (current\_time - last\_ntp\_update\_time).total\_seconds() >= ntp\_update\_interval:

            ntp\_time = check\_ntp\_time()

            if ntp\_time:

                current\_time = ntp\_time

                NTPstatus = "NTP time successfully updated."

            else:

                NTPstatus = "NTP server not reachable, using local time"

            last\_ntp\_update\_time = current\_time

#        print("After NTP update check")  # Debugging Option

        # Check if it's time to measure DHT22 data

        if (current\_time - last\_dht22\_measurement\_time).total\_seconds() >= dht22\_measurement\_interval:

            try:

                dht22\_temperature, dht22\_humidity = read\_dht22\_values(board.D4)

                if (dht22\_temperature is not None and dht22\_humidity is not None):

                    last\_successful\_dht22\_measurement\_time = current\_time

                    dht22\_last\_measurement\_time = current\_time

                    dht22\_last\_temperature = dht22\_temperature

                    dht22\_last\_humidity = dht22\_humidity

                else:

                    dht22\_last\_measurement\_time = None

                    dht22\_last\_temperature = None

                    dht22\_last\_humidity = None

            except Exception as e:

                print("Error reading DHT22 data:", str(e))

            last\_dht22\_measurement\_time = current\_time

        # Check if a successful measurement was taken in the last 30 seconds

        if last\_successful\_dht22\_measurement\_time is not None and (current\_time - last\_successful\_dht22\_measurement\_time).total\_seconds() > 30:

            last\_successful\_dht22\_measurement\_time = None

            dht22\_last\_temperature = None

            dht22\_last\_humidity = None

        # Check if it's time to transmit data

        if (current\_time - last\_mqtt\_publish\_time).total\_seconds() >= mqtt\_publish\_interval:

            try:

                ds1820\_temperature = read\_ds1820\_temperature("/sys/bus/w1/devices/28-0301a279b45b/w1\_slave")

                color\_rgb = read\_color\_data()

                current\_datetime = datetime.now()

                timestamp = current\_datetime.strftime("%d/%m/%Y %H:%M:%S")

                timestampinflux = int(time.time() \* 1e9)

                data = {

                    "dht22\_temperature": dht22\_last\_temperature,

                    "dht22\_humidity": dht22\_last\_humidity,

                    "ds1820\_temperature": ds1820\_temperature,

                    "red": color\_rgb[0],

                    "green": color\_rgb[1],

                    "blue": color\_rgb[2],

                    "NTPstatus": NTPstatus,

                    "timestamp": timestamp,

                    "timestampinflux": timestampinflux

                }

                msg = json.dumps(data)

                result = client.publish(mqtt\_topic, msg)

                status = result.rc

                print(f"Current status: {NTPstatus}")  # Debugging-Anweisung

                if status == mqtt.MQTT\_ERR\_SUCCESS:

                    print(f"Data `{msg}` sent to Topic `{mqtt\_topic}`")

                else:

                 print(f"Error sending data to Topic {mqtt\_topic}")

            except Exception as e:

                print("Error transmitting data:", str(e))

            last\_mqtt\_publish\_time = current\_time

        time.sleep(sleep\_interval)

# Main Program

if \_\_name\_\_ == '\_\_main\_\_':

    client = connect\_mqtt("data\_publisher")

    client.loop\_start()

    while True:

        ntp\_time = check\_ntp\_time()

        if ntp\_time:

            publish\_sensor\_data\_and\_status(client, ntp\_time)

        else:

            publish\_sensor\_data\_and\_status(client, datetime.now())

#### Telegraf

Telegraf ist ein flexibler Open-Source-Agent zur Datenerfassung aus verschiedenen Quellen. Die Konfiguration erfolgt über eine Datei, die leicht verständlich ist und sich automatisch generieren lässt. In diesem Projekt wird Telegraf genutzt, um Daten von MQTT an InfluxDB weiterzuleiten.

[[inputs.mqtt\_consumer]]

  servers = ["tcp://agb-mokipro-db.one.fc.eah-jena.de:1883"]

  qos = 0

  connection\_timeout = "30s"

  topics = [ "dht22", "ds1820" ]

  data\_format = "json\_v2"

  [[inputs.mqtt\_consumer.json\_v2]]

    measurement\_name = "data"

    timestamp\_path = "timestampinflux"

    timestamp\_format = "unix\_ns"

    timestamp\_timezone = "Europe/Berlin"

    [[inputs.mqtt\_consumer.json\_v2.field]]

      path = "dht22\_temperature"

      type = "float"

      topic = "dht22"

    [[inputs.mqtt\_consumer.json\_v2.field]]

      path = "dht22\_humidity"

      type = "float"

      topic = "dht22"

    [[inputs.mqtt\_consumer.json\_v2.field]]

      path = "ds1820\_temperature"

      type = "float"

      topic = "ds1820"

    [[inputs.mqtt\_consumer.json\_v2.tag]]

      path = "NTPstatus"

    [[inputs.mqtt\_consumer.json\_v2.tag]]

      path = "timestamp"

[[outputs.influxdb\_v2]]

  urls = ["http://agb-mokipro-db.one.fc.eah-jena.de:8086"]

  token = "M2TcVhTYRYrDdXHYUw1aZbAcmEpUc97nifcw1wQV6ZVXUN61RYO6KZqLawCASTQfV\_JeR3-p2ATImh91HAfphw=="

  organization = "EAH"

  bucket = "Sensordaten"

  insecure\_skip\_verify = true

#### Grafana

Grafana ist ein leistungsstarkes Tool zur Visualisierung und Analyse von Daten. Es unterstützt zahlreiche Datenquellen und bietet umfassende Möglichkeiten zur Echtzeitanalyse und -überwachung. In Kombination mit InfluxDB ermöglicht Grafana eine zuverlässige Lösung für die Datenvisualisierung.

Obwohl Grafana im Docker Compose enthalten ist, wird die zuvor verwendete Grafana-Instanz genutzt, da sie bereits wie gewünscht konfiguriert ist.

Quellen:

|  |  |
| --- | --- |
|  | Informationen zu den Sensoren |
| Datenblatt DHT22: | <https://cdn.shopify.com/s/files/1/1509/1638/files/DHT_22_-_AM2302_Temperatur_und_Luftfeuchtigkeitssensor_Datenblatt.pdf?11983326290748777409> |
| Schaltungsplan  DHT22: | <https://cdn.shopify.com/s/files/1/1509/1638/files/DHT_22_-_AM2302_Temperatur-_und_Luftfeuchtigkeitssensor_atmega_Schematics_1.pdf?v=1677138007> |
| Datenblatt  DS18B20: | <https://cdn.shopify.com/s/files/1/1509/1638/files/DS18B20_TO-92_datasheet_0ec6827c-2bee-4eca-8a6d-5d68ae26bd8f.pdf?v=1656592898> |
| Schaltungsplan  DS18B20: | <https://cdn.shopify.com/s/files/1/1509/1638/files/DS18B20_Temperatursensors_Connection_Diagrams.pdf?v=1615282043> |
| Datenblatt  TCS3472: | <https://cdn.shopify.com/s/files/1/1509/1638/files/TCS3472_datasheet.pdf?v=1675927629> |
| Schaltungsplan  TCS3472: | <http://www.getmicros.net/raspberry-pi-and-tcs34725-color-sensor-java-example.php> |
|  |  |
|  | Komponenten |
| Raspberry Pi 4 Model B Rev 1.1 | <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/> |
| DHT22-Sensor | <https://www.reichelt.de/entwicklerboards-temperatur-feuchtigkeitssensor-dht22-debo-dht-22-brd-p266104.html?PROVID=2788&gclid=CjwKCAjwgsqoBhBNEiwAwe5w07Tew4_fYczufxC7GeSqlt5688w8__-LK-xz1gMHgw_wZBId205oRRoCEHkQAvD_BwE> |
| DS18B20-Sensor | <https://www.reichelt.de/shelly-temperatur-sensor-ds18b20-shelly-ds18b20-p287127.html?&trstct=pos_0&nbc=1> |
| TCS3472-Sensor | <https://www.az-delivery.de/products/tcs34725-rgb-color-sensor?_pos=1&_sid=addfee278&_ss=r> |
| Gehäuse Cube | <https://www.reichelt.de/gehaeuse-fuer-raspberry-pi-4-cube-dunkelgrau-rpi-case4-cb-dg-p279773.html> |
| Breadboard | <https://www.reichelt.de/experimentier-slide-steckboard-300-100-kontakte-steckboard-s4-p177331.html?&trstct=pos_6&nbc=1> |
| M 8, 4-pol, Buchse | <https://www.reichelt.de/sal-m-8-4-pol-buchse-gerade-conec-42-01245-p249162.html?&trstct=vrt_pdn&nbc=1> |
| Sensorleitung, M8, 4 pol | <https://www.reichelt.de/sensorleitung-m8-4-pol-st-offenes-ende-2-m-lut-0810-0-2-p223081.html?&trstct=vrt_pdn&nbc=1> |
|  |  |
|  |  |
| Betriebssystem  Raspberry: | <https://www.raspberrypi.com/software/operating-systems/> |
|  |  |

Konfiguration mit Farbsensor

  GNU nano 7.2                                                                             telegraf.conf

# Read metrics from MQTT topic(s)

[[inputs.mqtt\_consumer]]

    servers = ["tcp://agb-mokipro-db.one.fc.eah-jena.de:1883"]

    qos = 0

    topics = [ "SensordatenPi2","SensordatenPi1" ]

    connection\_timeout = "30s"

    data\_format = "json\_v2"

    [[inputs.mqtt\_consumer.json\_v2]]

        measurement\_name = "data"

        timestamp\_path = "timestamp"

        timestamp\_format = "unix"

        timestamp\_timezone = "Europe/Berlin"

    [[inputs.mqtt\_consumer.json\_v2.field]]

            path = "dht22\_temperature"

            type = "float"

    [[inputs.mqtt\_consumer.json\_v2.field]]

            path = "dht22\_humidity"

            type = "float"

    [[inputs.mqtt\_consumer.json\_v2.field]]

            path = "ds1820\_temperature"

            type = "float"

    [[inputs.mqtt\_consumer.json\_v2.field]]

             path = "red"

             type = "float"

    [[inputs.mqtt\_consumer.json\_v2.field]]

             path = "green"

             type = "float"

    [[inputs.mqtt\_consumer.json\_v2.field]]

             path = "blue"

             type = "float"

    [[inputs.mqtt\_consumer.json\_v2.tag]]

            path = "NTPstatus"

    [[inputs.mqtt\_consumer.json\_v2.tag]]

      path = "timestamp"

# Output Configuration for telegraf agent

[[outputs.influxdb\_v2]]

  urls = ["http://agb-mokipro-db.one.fc.eah-jena.de:8086"]

  token = "2enimJByiFilHAn\_0MEFL\_GsY4I6B-gxYqrryUfnY4nFHN4urko0x6k6CdS53i5cKx4BImmyqT-pAfdDLIX2MA=="

  organization = "some\_org"

  bucket = "Sensordaten"

  insecure\_skip\_verify = true

wie der DS1820 Sensor bzw. wie der Pi konfiguriert werden muss für den Sensor

<https://randomnerdtutorials.com/raspberry-pi-ds18b20-python/>

bei 5V wurde der Sensor nicht erkannt