IoT Monitoring

Pia Bereuter

15.10.2025

IoT Monitoring ermöglicht die Erfassung von Sensordaten, die Kommunikation und Visualisierung von Daten in Echtzeit. Dies erfordert eine Architektur, die die Daten erfasst, Echtzeitkommunikation, asynchroner Kommunikation, Datenspeicherung und Visualisierung ermöglicht. Diese Übung baut einen typischen IoT Stack auf mit MQTT, Node-Red, InfluxDB und Grafana und zeigt wie Sensordaten über MQTT publiziert und abonniert werden können.

Inhaltsverzeichnis

1 Einführung

Ein IoT Datenfluss erstreckt sich über verschiedene Instanzen, die für die einzelnen Prozesse zuständig sind, von der Erfassung von Sensormesswerten im IoT Gerät über die Kommunikation der Messwerte bis zur Datenverarbeitung, Speicherung und Visualisierung (Abb. 1). Hierbei können alle Schritte auf einem Gerät durchgeführt werden oder jeder einzelne über ein anderes Gerät oder Server. In der Abbildung aufgezeigt sind typische Softwarekomponenten, die in IoT eingesetzt werden, wie Node-Red für die Datenverarbeitung, die InfluxDB Datenbank, welche für das Speichern von Zeitreihendaten entwickelt wurde und Grafana eine Visualisierungsplattform, die für Messdaten optimiert ist. Es sind Werkzeuge die über ihre graphische Oberfläche einen guten Einstieg ermöglichen, sogenannte *low-code* Tools, die sich gut eignen für die Entwicklung von Prototypen mit geringem zeitlichen Aufwand. Je nach Anwendung können die einzelnen

Prozessschritte auch gut in einer Scriptsprache wie Python durchgeführt oder eine andere Datenbank verwendet werden.

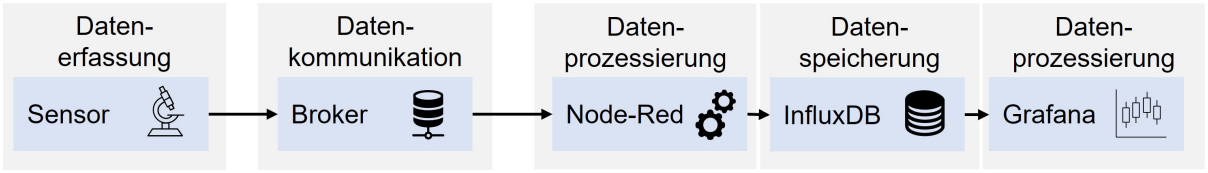

Abb. 1: Typischer IoT Datenfluss und Verarbeitung über diverse Instanzen, von dem IoT Gerät mit Sensorik, Datenkommunikation mit MQTT und dem MQTT Broker, zur Datenprozessierung mit Node-Red, Datenspeicherung und und Visalisierung.

Ziel dieser Übung ist es MQTT näher kennenzulernen und die MQTT Kommunikation mit dem Raspberry Pi zu testen. MQTT ist ein leichtgewichtiges Kommunikationsprotokoll, welches das *Publish-Subscribe* Muster verwendet und sich gut für die Anwendung in IoT Projekten geeignet.

Unterlagen: *E07_MQTT.zip*

2 Übungsaufbau

- Schliesse den Raspberry Pi an Monitor, Keyboard und Maus an oder verbinde Dich mit diesem über SSH (und SFTP).
- Erstelle auf dem Raspberry Pi im `Documents` Ordner einen neuen Ordner `mqtt`, in welchem Du Änderungen und neue Dateien für diese Übung speichern kannst.
- Schliesse den Sensor **BME688** an den Raspberry Pi über die Breakout Garden **I2C** Schnittstelle korrekt an (siehe E01 Luftqualität), so dass die Beschriftung der Anschlüsse am Sensor und bei der Schnittstelle übereinstimmen.
- Kontrolliere mit dem Befehl `i2cdetect -y 1` ob der Raspberry Pi mit dem Sensor verbunden ist. Der Sensor sollte auf der Adresse `0x76` erkannt werden.
- Kontrolliere, ob die Library `bme680` installiert ist mit `python -c "import bme680"`. Installiere die Library mit in der aktivierten virtuellen Environment `source ~/.env/bin/activate` mit `pip install bme680`, falls sie nicht installiert ist.
- Wechsle in den Ordner *Documents* und erstelle einen Ordner *mqtt* für diese Übung.

3 Aufgabe 1: MQTT kennenlernen

Mosquitto ist eine kompakter open-source Message Broker der Eclipse Foundation, welcher das MQTT Protokoll implementiert und auf Produktionsserver, wie auch auf stromsparenden Geräten wie dem Raspberry Pi eingesetzt werden kann. Wir testen die Kommunikation mit dem MQTT Broker und den Clients auf dem Raspberry Pi. Der Mosquitto Broker ist auf dem Raspberry Pi Image vorkonfiguriert und läuft als Service im Hintegrund. Die Clients `mosquitto_pub` und `mosquitto_sub` sind ebenfalls installiert und können für das Testen der Kommunikation MQTT verwendet werden. Mosquitto Dokumentation: https://mosquitto.org/documentation

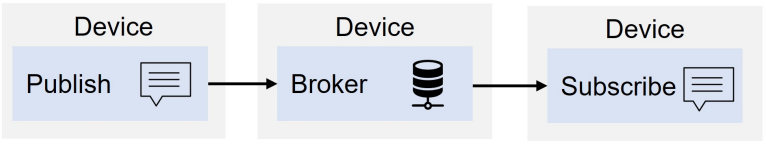

Abb. 2: MQTT Device publiziert an den MQTT Broker über ein definiertes Topic Messdaten. Der Broker publiziert die Daten über die definierten Topics. Diese werden vom Subscriber abonniert und empfangen mit dem Vorteil, dass keines der Geräte zur gleichen Zeit synchron senden und empfangen muss.

Erstelle einen Subscriber der für das Topic `iot/temperature` eine Subscription erstellt.

```
mosquitto_sub -h 127.0.0.1 -v -t 'iot/temperature'
```

Öffne ein zweite Shell und erstelle einen Publisher für dasselbe Topic

```
mosquitto_pub -h 127.0.0.1 -t 'iot/temperature' -m 'Aussentemperatur: 22° Celsius'
```

MQTT Message Die MQTT Nachrichten (Messages) bestehen aus einem *Topic* (Thema) und einer *Payload* dem Nachrichteninhalt.

MQTT Topics MQTT verwendet Themen (Topic) und hierarchische Themenebenden getrennt über einem Schrägstrich `/` ähnlich einem Ordnersystem auf einem Computer. Dies ermöglicht eine Strukturierung und auch Filterung der Messages. Topics sind case-sensitive, sollten nicht mit einem `/`, `$` beginnen und keine Leerzeichen enthalten, sowie kurz und aussagekräftig sein. Filtern von Topics ist mit dem Wildcard `+` für ein einzelnes Thema und `#` für alle Themen möglich[1].

MQTT Payload Die Nachrichten (Messages) die über MQTT übertragen werden, werden als Payload bezeichnet. Diese Payload ist nicht an eine bestimmte Struktur gebunden und kann ein Text- oder Binärformat sein. Eine festgelegte Struktur für die Payload ist jedoch sinnvoll für eine reibungslose Datenverarbeitung beispielsweise im JSON Format. Übliche Fromate sind Text, JSON, Binärdaten, Hex Strings oder auch Protocol Buffer.

Falls der Mosquitto Brokers und Clients nicht installiert sind, können diese mit folgenden Befehlen installiert werden.

```
sudo apt install mosquitto -y
sudo apt install mosquitto-clients -y
sudo systemctl enable mosquitto.service
sudo systemctl status mosquitto
```

[1] Topics, die mit `$` beginnen sind reserviert für interne Informationen des Brokers und Clientstatistiken, oft in der Form `$SYS/`. Beispielsweise zeigt `$SYS/broker/clients/connected` die Anzahl der verbundenen Clients. Mit dem Wildcard `#` Filter könnt Ihr mit `$SYS/#` alle Systeminformationen der Brokers anzeigen lassen.

Übung 3.1.

- Teste nun den MQTT mit unterschiedlichen Topics und Nachrichten.
- Teste einen weiteren Mosquitto Server auf einem anderen Raspberry Pi und teste die Kommunikation. Hierbei muss die IP Adresse entsprechend angepasst werden.
- Einigt Euch auf einen MQTT Broker und eine Topic-Struktur beispielweise `iot/temperature` und `iot/humidity` und publiziert und abonniert Nachrichten. Was fällt Euch auf, wenn Ihr die Nachrichten empfängt?
- Welche Angaben sollten zusätzlich zu den Messwerten in der Payload der Nachrichten enthalten sein?

Hinweis

MQTTX und MQTT Explorer sind zwei MQTT Clients, die für die Entwicklung und das Testen von MQTT Applikationen verwendet werden können. Jedoch scheinen diese nicht mehr oder eher sporadisch weiterentwickelt zu werden.

4 Aufgabe 2: MQTT mit Python verwenden

Nun verwenden wir die Bibliothek `paho-mqtt` in Python um MQTT Messages zu senden oder empfangen. Folgende zwei Code Snippets zeigen wie ein Publisher und Subscriber in Python minimal implementiert werden können.

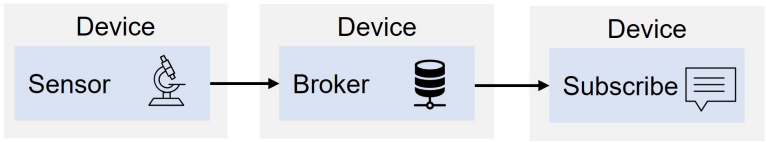

Abb. 3: Device sendet Sensordaten und Topic an den MQTT Broker, ein weiteres Gerät abonniert (subscribe) dieses Topic und kann so die gesendeten Daten an das entsprechende Topic empfangen. In dieser Übung über ein Python Script für *Publish* und *Subscribe*

Script 1: `mqtt_sub.py` - Subscriber

```
import paho.mqtt.client as mqtt
ip = "127.0.0.1"                                                    ①
topic = "iot/temperature"                                           ②

# Callback Funktion für den Verbindungsaufbau
def on_connect(client, userdata, flags, rc):                        ③
    print("Connected - code: "+str(rc))
    client.subscribe(topic)                                         ④

# Callback Funktion für eingehende Nachrichten
def on_message(client, userdata, msg):                              ⑤
```

```
    print(msg.topic+" "+str(msg.payload))

# Erstellen des MQTT Clients
client = mqtt.Client()                                                    ⑥
client.on_connect = on_connect
client.on_message = on_message
client.connect(ip, 1883, 60)                                              ⑦
client.loop_forever()
```

① IP Adresse des MQTT Brokers
② Topic auf welches der Subscriber hört
③ Callback Funktion on_connect wird ausgeführt, wenn die Verbidnung steht
④ Subscription für das Topic
⑤ Callback Funktion on_publish wird ausgeführt, wenn eine Nachricht empfangen (publish) wird
⑥ Erstellen eines MQTT Clients und Zuweisung der Callback Funktionen
⑦ Verbindung zum MQTT Broker herstellen und auf eingehende Nachrichten warten (loop_forever)

Script 2: `mqtt_pub.py` - Publisher

```
import paho.mqtt.publish as publish
ip = "127.0.0.1"                                                          ①
topic = "iot/temperature"
publish.single(topic, "22.0", hostname=ip, port=1883)                     ②
```

① IP Adresse des MQTT Brokers und Topic definieren
② Nachricht mit Topic an den MQTT Broker (ip,port) senden

Übung 4.1.
- Speichere die beiden Dateien `mqtt_pub.py` und `mqtt_sub.py` im Ordner `mqtt` ab.
- (Optional: Passe die IP Adresse des MQTT Brokers an, falls ein anderer MQTT Broker genutzt werden soll.)
- Öffne zwei Terminals und führe diese mit `python mqtt_pub.py` und `python mqtt_sub.py` aus.

Falls die Library paho-mqtt nicht installiert ist, kann diese in der aktivierten virtuellen Environment `source ~/.env/bin/activate` mit `pip install paho-mqtt`.

5 Aufgabe 3: Sensordaten mit MQTT übertragen

Passe nun das Script `mqtt_pub.py` an, so dass die Temperatur vom Sensor `BME688` ausgelesen und über MQTT übertragen wird. Die Temperatur soll alle 10 Sekunden übertragen werden.

Nutze hierfür das Script zum Auslesen der Sensordaten aus der Übung E01 Luftqualität und passe dieses an.

Übung 5.1.
- Speichere das `mqtt_pub.py` als `mqtt_pub_bme688.py` im Ordner `mqtt` ab.
- Ergänze das Script `mqtt_pub_bme688.py` mit Codezeilen für das Auslesen der Temperatur des BME688 Sensors.
- Ergänze das Script für das Auslesen der Luftfeuchtigkeit und Luftdruck und ergänze die topics mit `iot/humidity` und `iot/pressure`.
- Erweitere das Script `mqtt_sub.py` mit den Subscriptions für die Topics `iot/humidity` und `iot/pressure` und speichere es als `mqtt_sub_bme688.py` ab.
- Öffne zwei Terminals und führe diese mit `python mqtt_pub_bme688.py` und `python mqtt_sub_bme688.py` aus.

6 Aufgabe 4: MQTT mit Node-RED verwenden

Node-Red ist ein grafisches Entwicklungswerkzeug für IoT, ein low-code Tool für *event-driven applications*. Es bietet eine browserbasierte und datenstromorientierte “Flow” Programmierung für die Verarbeitung von Sensordaten (ähnlich wie FME oder graphische Modellierungswerkzeuge in GIS). Die Implementation von Node-Red ist in JavaScript und basiert auf node.js. Datenverarbeitungsflows können gespeichert und wiederverwendet werden. Im Arbeitsbereich (Abb. 4) können sogenannte Nodes, die unterschiedliche Funktionen erfüllen, zu einem Daten-“Flow” werden. Nodes können auch selbst erstellt werden. Es gibt eine Vielzahl von Nodes, die von der Community entwickelt wurden, die über den Node-Red Palette Manager installiert werden können. Core Nodes von Node-Red sind:

- **Inject Node**: Kann einen Flow über den Button direkt auslösen oder in regelmässigen Abständen mit Zeitstempel oder vordefinierten Nachrichten (msg) senden.
- **Debug Node**: Kann die Nachrichten (msg) in Debug Sidebar anzeigen lassen.
- **Function Node**: Kann mit JavaScript Funktionen den Inhalt der Nachrichten (msg) verändern.
- **Change Node**: Ermöglicht das Ändern von Eigenschaften einer Nachricht (ohne den Funktion Node zu nützen) um beispielsweise Eigenschaften zu setzen, ändern oder löschen.
- **Switch Node**: Kann Nachrichten (msg) anhand von Regeln auswerten in verschiedene Ausgänge leiten (wie ein Switch Case in der Programmierung).
- **Template Node**: Kann über Eigenschaften einer Nachricht und einer Vorlage (Template) neue Nachricht nach Vorlage erstellen: `Nachricht: {{payload}}!` wird `Nachricht: 1570439577309 !`.

Node-Red kann über den Browser auf dem Raspberry Pi oder via Laptop gestartet werden. Öffne den Browser und gebe die IP Adresse des Raspberry Pi mit dem Port 1880 ein: `http://<ip-adresse>:1880`. Führt nun das Kurztutorial zu Node-Red aus: Node-Red Getting Started:

First Flow. In diesem Tutorial wird ein Flow erstellt, der eine Nachricht mit einem Zeitstempel ausgibt.

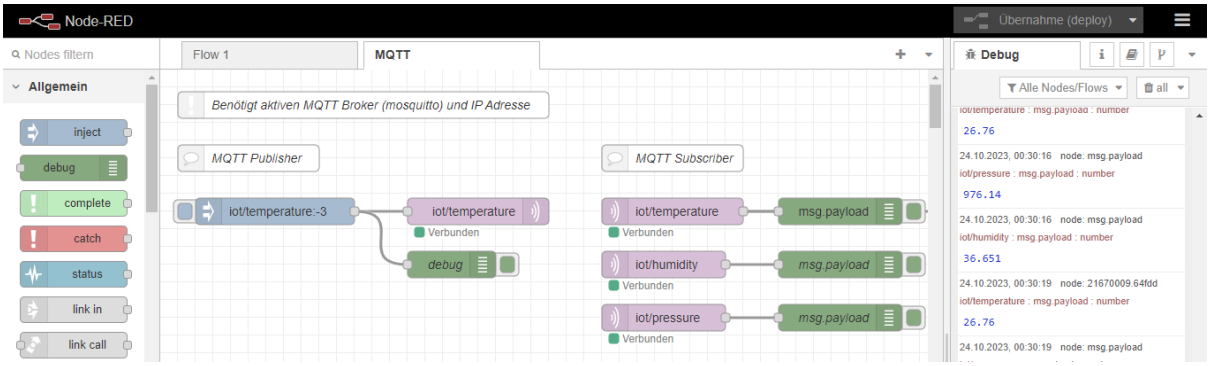

Abb. 4: Node-Red Flow der Daten in das Topic *iot/temperature* published, und Nachrichten aus dem Topic *iot/temperature* subscribed.

Übung 6.1. Erstelle einen Flow der Nachrichten an den MQTT Broker senden und Nachrichten vom MQTT Broker empfangen kann.

1. Füge hierfür einen Inject Node, einen Debug Node und einen MQTT Output Node hinzu. Öffne die Einstellungen des MQTT Output Nodes (siehe Abb. 5) und setze die IP Adresse des MQTT Brokers und das Topic auf `iot/temperature`. Starte den Flow und überprüfe, ob die Nachrichten an den MQTT Broker gesendet werden.
2. Erstelle nun einen weiteren Flow, der die Nachrichten vom MQTT Broker empfängt und in der Debug Sidebar anzeigt. Füge hierfür einen MQTT Input Node (siehe Abb. 5) und einen Debug Node hinzu. Öffne die Einstellungen des MQTT Input Nodes und setze die IP Adresse des MQTT Brokers und das Topic auf `iot/temperature`. Starte den Flow und überprüfe, ob die Nachrichten vom MQTT Broker empfangen werden.
3. Ergänze den Flow um einen MQTT Input Node für die Topics `iot/humidity` und `iot/pressure` und je einen Debug Node. Starte den Flow und überprüfe, ob die Nachrichten vom MQTT Broker empfangen werden.

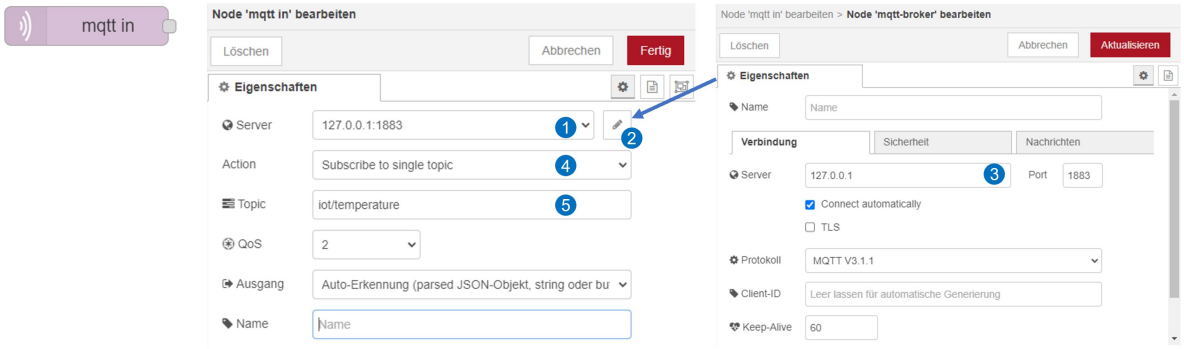

Abb. 5: Node-Red MQTT Node Einstellungen setzen, (1) Name angeben, (2) Server Einstellungen öffnen und (3) IP Adresse des MQTT Brokers setzen, (4) subscribe to single topic wählen und (5) Topic `iot/temperature` setzen.

Hinweis

Flows stoppen: Um einen Flow in der Ausführung zu stoppen, kann dieser über den Tab Flow (rechte Maustaste) deaktiviert werden (disable flow) und mit einem erneuten Übernehmen (Deploy) wird der Flow gestoppt.

7 Aufgabe 5: Node-Red MQTT mit InfluxDB verwenden

InfluxDB ist eine Datenbank, die für die Erfassung, Speicherung, Verarbeitung und Visualisierung von Zeitreihendaten entwickelt wurde[2]. Zeitreihendaten sind Datenpunkte, die in zeitlicher Sequenz erfasst wurden und bestehen in der Regel aus aufeinanderfolgenden Messungen aus derselben Quelle, wie beispielsweise die Temperaturdaten des BME688.

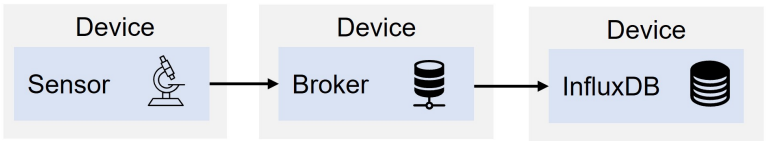

Abb. 6: IoT Datenfluss mit Node-Red MQTT und InfluxDB. Python Script publiziert (publish) die Sensormesswerte und das Topic an den MQTT Broker, NodeRed abonniert (subscribe) ein oder mehrere Topics und speichert die Sensormesswerte in einer InlfuxDB Datenbank.

InfluxDB organisiert die Zeitreihendaten in *Buckets* (anstatt Datenbanken) und Messungen (Measurements). Ein Bucket kann mehrere Messungen enthalten, wobei Messungen Felder und Tags enthalten können (influxdata, 2023). Eine Messung enthält *Felder* mit key-value Paaren

[2] Wir verwenden InfluxDB 2 in dieser Übung. Wobei zu bemerken ist, dass sich die Abfragesprache der InfluxDB seit der Veröffentlichung drei mal geändert hat, von InfluxQL bei Version 1.x, zu Flux in Version 2.x zurück zu InfluxQL in Version 3.x mit jeweils einem kompletten Rewrite des Backends, was bei den Anwendenden jeweils zu einem entsprechenden Migrationsaufwand führte. Eine Alternative wäre TimescaleDB ein Zeitreihendatenbank auf der Basis von postgreSQL.

von Messwerten, die sich über die Zeit ändern. *Tags* sind key-value Paare, die sich nicht über die Zeit ändern und für die Filterung und Gruppierung verwendet werden können.

Das Tutorial Getting Started zeigt die ersten Schritte mit InfluxDB[3].

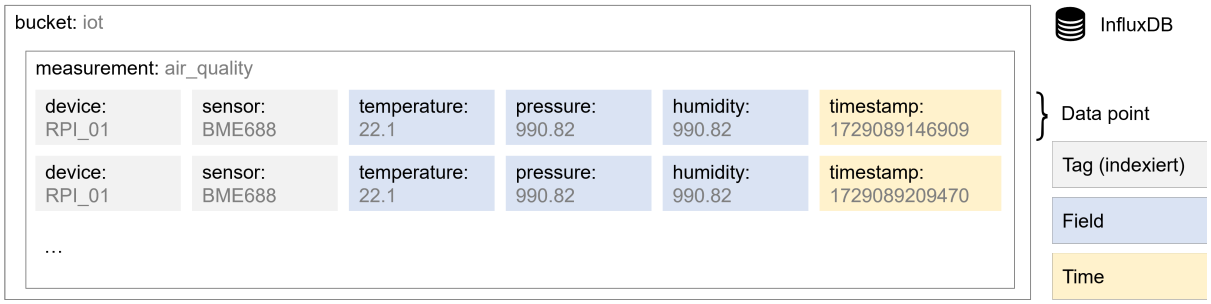

Abb. 7: Datenorganisation in der InfluxDB, über Buckets und Measurements, mit Tags für Metadaten, Feldern für die Speicherung der Messwerte und dem Zeitstempel.

Glossary

Bucket Ein Bucket ist ein benannter Container zur Speicherung der Zeitreihendaten. Buckets können mehrere Messungen enthalten und sind die oberste Ebene der Organisation in InfluxDB.

Measurement Eine Messung ist eine logische Gruppierung von Zeitreihendaten, mehreren Einzelnmessungen (Data Point, points). Alle Messungen sollten dieselben Tags enthalten. Messungen können mehrere Tags und Felder enthalten

Data Point Ein Datenpunkt ist eine Einzelnmessung, die zu einem bestimmten Zeitpunkt erfasst wurde. Ein Datenpunkt besteht aus einem *measurement*, *field set*, *tag set* und einem *timestamp*.

Tags Tags sind key-value Paare, die sich nicht (oder selten) über die Zeit ändern und für die Filterung und Gruppierung verwendet werden können. Tags können für die Organisation und Strukturierung der Daten verwendet werden und werden indiziert. Die Indexierung ermöglicht eine effiziente Abfrage und Aggregation der Daten.

Fields Felder sind key-value Paare von Messwerten, die sich über die Zeit ändern, wie Temperatur, Luftdruck etc. Felder können für die Speicherung von Messwerten verwendet werden. Felder werden nicht indiziert.

Timestamp Der Zeitstempel ist der Zeitpunkt, zu dem die Messung erfasst wurde. Der Zeitstempel wird in InfluxDB automatisch hinzugefügt, wenn er nicht explizit in der Messung definiert ist.

Die graphische Oberfläche von InfluxDB kann über den Browser auf dem Raspberry Pi oder via Laptop gestartet werden. Öffnet nun den Browser und gebe die IP Adresse des Raspberry Pi mit dem Port 8086 ein: `http://<ip-adresse>:8086`.

Erstellt über die linke Menuleiste unter *Load Data* einen neuen Bucket mit dem Namen `iot`, falls dieser nicht schon existiert[4]. Erstellt ein sogenanntes *API Token* über “Generate API Token”[5] und

[3] Die Dokumentation über die InfluxDB key concepts und das InfluxDB Schema Design führen übersichtlich in die Best Practices des Schema Designs und effiziente Abfragen ein.

[4] Falls es sich um den ersten Bucket handelt, der beim Erstellen des Adminkonto erstellt wurde, enthält dieser bei der ersten Filteroption viele Filtermöglichkeiten, die eher verwirren. Da kann es hilfreich sein, den Bucket zu löschen und einen neuen Bucket zu erstellen.

[5] Über *Generate API Token* können *All Access API Token* oder *Custom API Token* erstellt werden. Für diese Übung reicht der Einfachheit halber ein *All Access API Token* aus, welches den Token alle Rechte vergibt. In einer produktiven Umgebung sollten jedoch *Custom API Tokens* mit den entsprechenden Rechten erstellt werden.

kopiert unter *load Data / API Tokens* den Token für den Zugang zur Datenbank.

Diese Angaben (*bucket* und *API Token*) werden für den InfluxDB Node in Node-Red benötigt. Die Einstellungen des InfluxDB Node benötigen die Angaben, wohin die Daten in der Datenbank gespeichert werden (links in Abb. 8) mit Angaben zum Bucket (Datenbankname), Organisation[6] und measurement und die Serververbindung (rechts in Abb. 8) mit der IP Adresse und das Token für den Zugang zur Datenbank.

Hinweis

Falls keine Nodes für InfluxDB in Node-Red vorhanden sind, kann die Erweiterung `node-red-contrib-influxdb` über das Hauptmenu *Palette verwalten* im Tab *Installation* installiert werden.

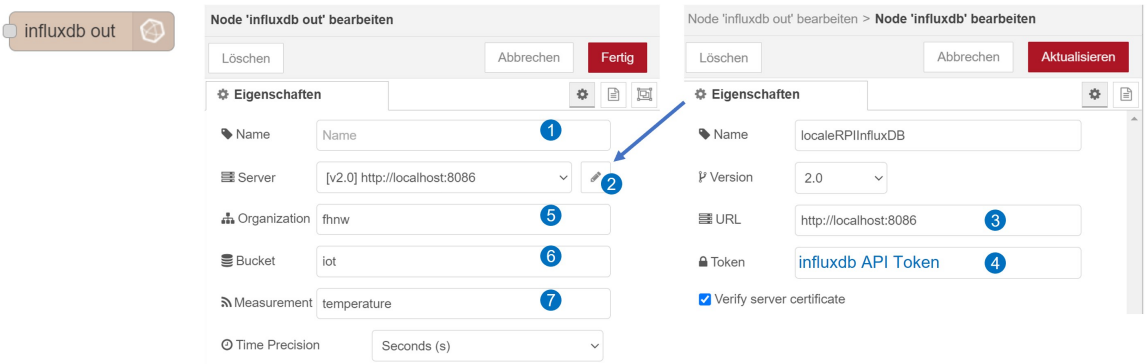

Abb. 8: Einstellungen der InflxDB Nodes in Node-Red, links: Einstellungen zur Datenbank (1) mit Angaben zur Organisation (5), Bucket (6) und Namen der Messung *measurement* (7), rechts: Einstellungen zur Datenbankverbindung (2) mit der URL und Port der InfluxDB (3) und dem API Token (4), für den Zugang zur Datenbank.

Die Messwerte können direkt in die InfluxDB geschrieben werden, jedoch fehlen da noch die Tags, die für die Filterung und Gruppierung verwendet werden können. Diese können über den Change Node hinzugefügt werden, wie in Abb. 9 gezeigt. Hierbei wird der Wert der Payload in ein Feld mit dem Namen *temperature* geschrieben und die Tags *device* und *sensor* werden hinzugefügt. Dieselbe Struktur kann alternativ auch mit dem Function Node erstellt werden, siehe Abb. 9 rechts. Hierbei wird der Wert der Payload in ein Feld mit dem Namen *temperature* geschrieben und die Tags *device* und *sensor* werden hinzugefügt. Die Bezeichnung der Messung *temperature* wird im InfluxDB Node definiert oder kann im Function Node überschrieben werden.

[6] wird bei der Erstellung der Datenbank definiert

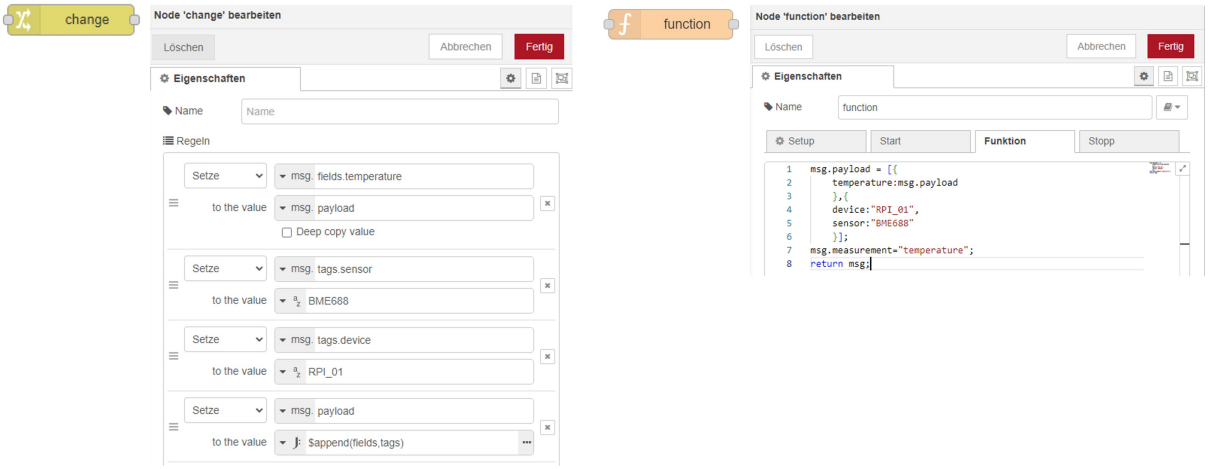

Abb. 9: Über den Change Node, wie auch den Function Node kann mit JavaScript der Inhalt der Payload verändert werden und diesen für den Import in die InfluxDB angepasst werden.

```
msg.payload = [{
    temperature:msg.payload                                    ①
}, { device:"RPI_01", sensor:"BME688" }];                      ②
msg.measurement="temperature";                                 ③
return msg;
```

① Werte der Payload in Payloadstruktur für die InfluxDB schreiben
② Tags der Messung zuweisen
③ Optional *measurement* kann im InfluxDB Node oder im Funktionsknoten definiert werden

InfluxDB bietet über das Menu *Data Explorer* eine einfache Möglichkeit die Daten zu visualisieren. Der Query Builder (Abb. 10) hilft bei der Erstellung von Abfragen mit der über die Fields und Tags gefiltert werden kann, die dann über den Button *Submit* ausgeführt und dargestellt werden können. Die erstellte Query kann über den *Script Editor* (Abb. 10) angezeigt werden.

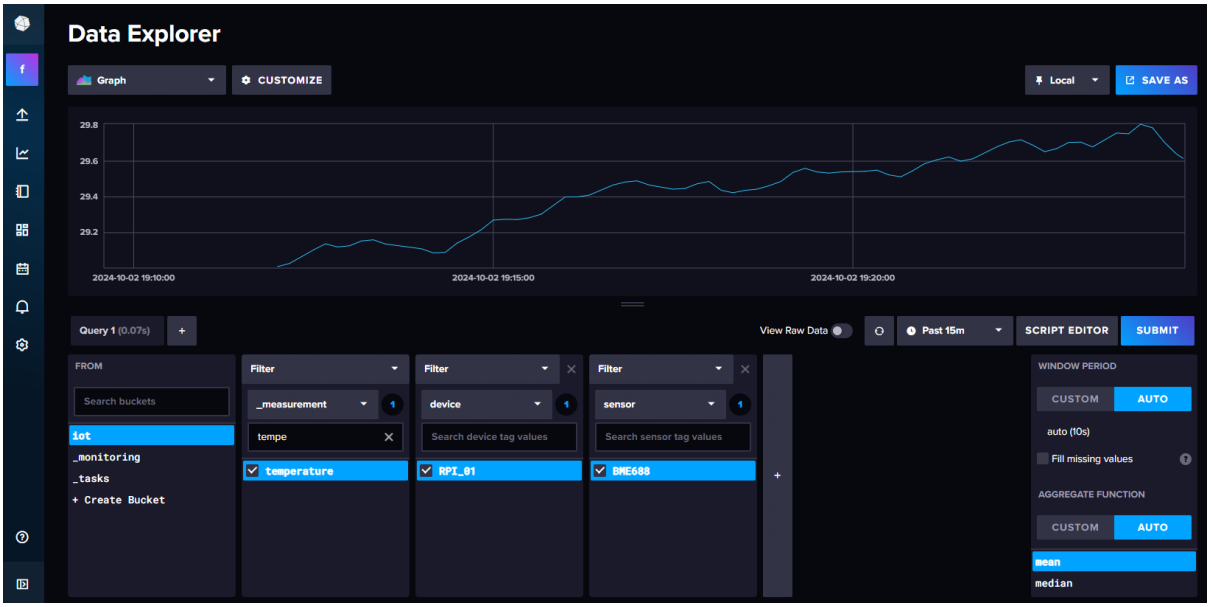

Abb. 10: Über den *Data Explorer* können in InfluxDB einfach Abfragen zusammengestellt und über *submit* visualisiert werden

Übung 7.1.

- Erstellt nun einen Flow der die Nachrichten des Topics `iot/temperature` vom MQTT Broker empfängt und in die InfluxDB schreibt.
- Erstellt erst einen flow der nur die Nachrichten ohne Tags in die InfluxDB schreibt
- Ergänzt den Flow mit einem Change oder Function Node um die Tags *device* und *sensor* hinzuzufügen.
- Visualisiert den Temperaturverlauf des BME688 in der InfluxDB mit dem *Data Explorer* und studiert die Flux Abfragesprache.
- Erstellt einen Flow der die Nachrichten der Topics `iot/humidity` und `iot/pressure` vom MQTT Broker empfängt und in die InfluxDB schreibt.
- (Optional) erstellt im Menu *Dashboard* eine Visualisierung der Messwerte.

8 Aufgabe 6: InfluxDB mit Grafana verwenden

Grafana ist eine Open-Source Anwendung für die grafische Darstellung von Daten aus den verschiedenen Datenquellen, wie Postgres, SQLite oder InfluxDB. Über Grafana können interaktive Dashboards mit vielen Visualisierungsansätzen erstellt werden. In dieser Übung wird Grafana mit der InfluxDB Datenbank verwendet.

Öffnet nun den Browser und gebt die IP Adresse des Raspberry Pi mit dem Port 3000 ein: `http://<ip-adresse>:3000`. Unter *Connections / Add new connection* können Grafana Datenquellen hinzugefügt werden, wählt nun *InfluxDB* um Grafana mit der InfluxDB zu verbinden. Setzt folgende Einstellungen (Abb. 12) und speichert diese:

Hinweis

Monitoring der Lichtverschmutzung am Nachthimmel TESS (Telescope Encoder and Sky Sensor) ist ein Photometer (Zamorano et al., 2016) für das Stars4All Citizen Science Projekt mit dem Ziel die Lichtverschmutzung weltweit in der Nacht zu vermessen, welche nicht nur für die Umwelt problematisch ist, sondern auch für astronomische Beobachtungen. Die TESS Geräte senden von verschiedenen Standorten weltweit ihre Messungen welche über der folgende Dashboard mit Grafana der Öffentlichkeit zur Verfügung gestellt werden. Dashboard: https://tess.dashboards.stars4all.eu/, Messungen der Lichtverschmutzung am Beispiel des Naturparks Gantrisch in der Schweiz. https://tess.dashboards.stars4all.eu/d/datasheet_stars926/stars926?orgId=1

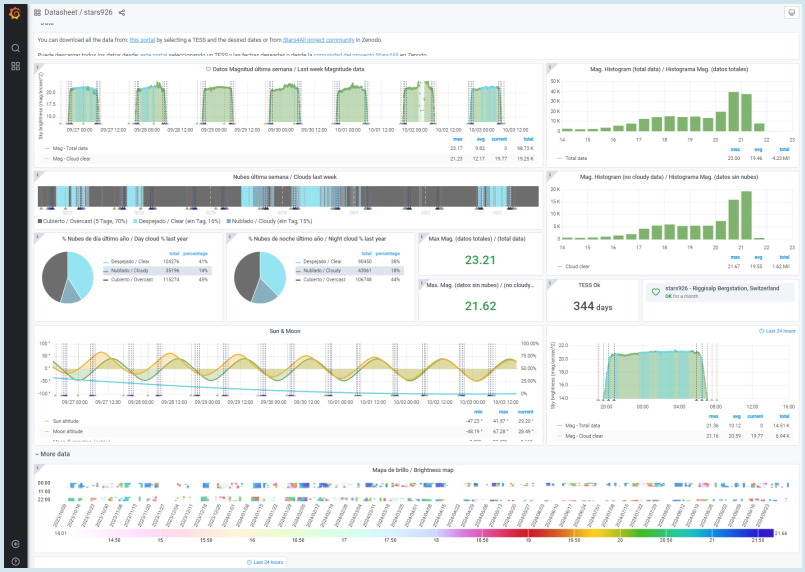

Abb. 11: Grafana Dashboard der Messdaten des TESS Photometes im Naturpark Gantrisch dem ersten Dark Sky Park der Schweiz

– *Query Language*: Wählt in den Einstellungen zu *Query Language* die Abfragesprache `Flux`.
– *HTTP*: Setzt die IP Adresse des Raspberry Pi und den Port 8086: `http://localhost:8086`.
– *Auth*: aktiviert *Basic Auth*
– *Basic Auth Details*: Setzt den InfluxDB Benutzer und das Passwort für die InfluxDB
– *InfluxDB Details*: Setzt die InfluxDB Organisation *fhnw*, den *API Token*, und den InfluxDB Bucket *iot*.

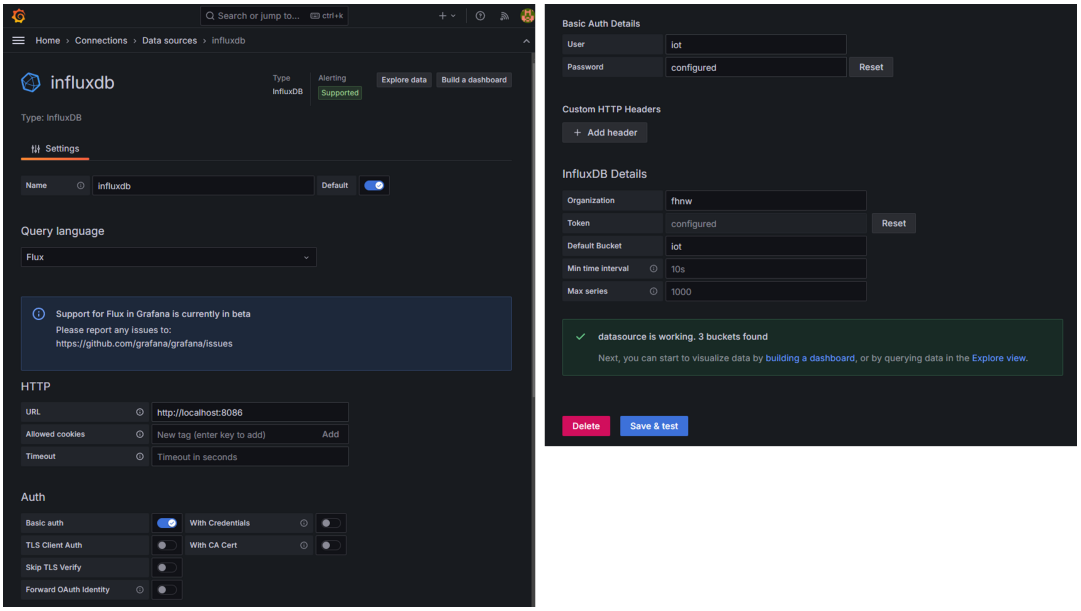

Abb. 12: InfluxDB als Datenquelle in Grafana hinzufügen mit den entsprechenden Angaben.

Ist die Verbindung erstellt, können neue Dashboards erstellt werden. Die Queries, die für die Visualisierung verwendet werden sollen, können über den *Query Builder* in InfluxDB erstellt und in Grafana beim Erstellen der Panels (Abb. 13) reinkopiert werden.

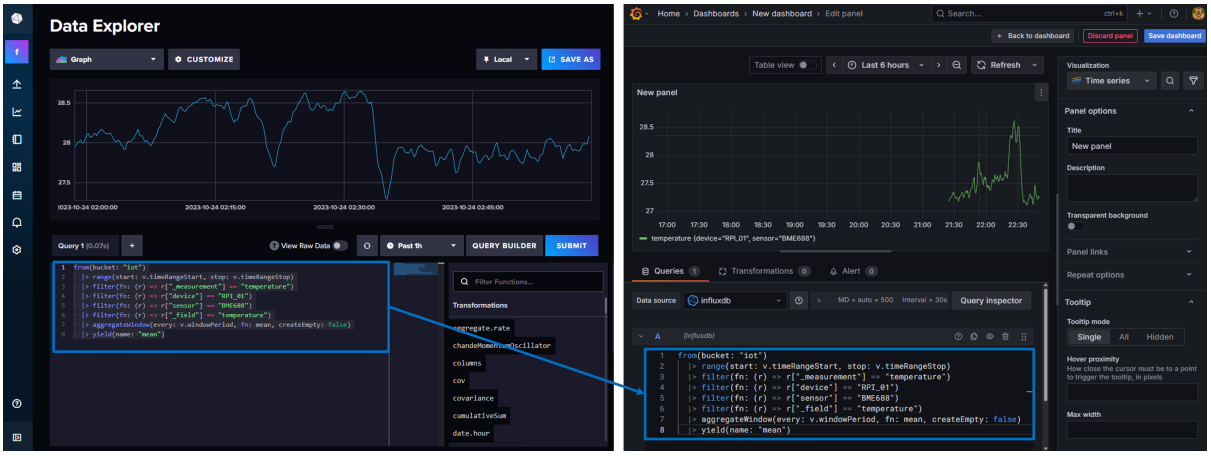

Abb. 13: Flux Queries der Datenabfragen im Data Explorer können in Grafana für die Visualisierung kopiert und genutzt werden.

Übung 8.1.

- Erstellt eine Verbindung zwischen Grafana und der InfluxDB.
- Erstellt ein Dashboard mit einer Visualisierung der Temperatur, Luftfeuchtigkeit und Luftdruck des BME688 Sensors.

9 Aufgabe 7: InfluxDB mit Python verwenden (optional)

Anstatt mit Node-Red können die Daten des Sensors direkt in die InfluxDB geschrieben werden. Folgender Code zeigt wie die Daten des BME688 Sensors mit Python ausgelesen und in die InfluxDB geschrieben werden können. Teste erst, ob der `influxdb-client` installiert ist und installiere diesen falls nicht mit dem Befehl `pip3 install influxdb-client`. Folgendes Tutorial zeigt wie die Daten mit Python in die InfluxDB geschrieben werden können: Getting Started with Python and InfluxDB v2.0.

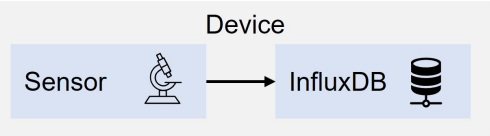

Abb. 14: Direktes Speichern der Sensormessdaten in der Influxdb ohne MQTT Protokoll. Hierbei gilt zu beachten, dass beide Geräte eine aktive Verbindung haben.

```
from datetime import datetime
import time

from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

# Generieren ein Token in der InfluxDB UI unter dem Tab "Data / Tokens Tab"
token = "<influxdb-API-token>"
org = "fhnw"
bucket = "iot"

client = InfluxDBClient(url="http://localhost:8086", token= token, org= org)
write_api = client.write_api(write_options = SYNCHRONOUS)

temperature = 22.0
data =
↪  Point("measures").tag("sensor","BME688").tag("device","RPI_01").field("temperature",
↪  temperature)
write_api.write(bucket = bucket, record = data)
```

Übung 9.1. Schreibe nun mit Hilfe des Tutorials und dem Beispielcode ein Python Script, welches die Temperatur, Luftfeuchtigkeit und Luftdruck des BME688 Sensors ausliest und direkt in die InfluxDB schreibt.

Referenzen

influxdata. (2023). *Get Started with InfluxDB | InfluxDB OSS v2 Documentation*. https://docs.influxdata.com/influxdb/v2/get-started/.

Zamorano, J., García, C., Tapia, C., Miguel, A. S. de, Pascual, S., & Gallego, J. (2016). STARS4ALL Night Sky Brightness Photometer. *International Journal of Sustainable Lighting*, *18*, 49–54. https://doi.org/10.26607/ijsl.v18i0.21