22.06.2023 04:03

Dokumentation T1 „Pipeline“

Fachbereich Automatisierung/Informatik
Hochschule Harz, Wernigerode
VT I 4.0
Prof. Adler

Norman Bauersfeld
Studiengang bbgl. DaSc(M. Sc.)
Matrikel 30315

Vorrausetzungen

- TCLAB; Aufnahme von Temperatur-Sensor-Daten
  https://tclab.readthedocs.io/en/latest/
- OPCUA Server; Python Programmierung aufbauend auf dem Python Modul *asyncua*
- Node-Red; Installation mit entsprechendem Flow zu Umsetzung der OPCUA Subscription auf ein MQTT Out
- aedes-cli; Installation zur Bereitstellung eines MQTT Brokers
- Telegraf; Installation eines Interface zwischen MQTT Broker und InfluxDB; entsprechende inputs/outputs Konfigurationen
- InfluxDB; Installation einer Datenbank für das Speichern von Zeitreihen

Hinweis: Alle Installationen/Konfigurationen wurden auf einem Windows 11 System realisiert und auf *localhost* (also Lokal) getestet.

TCLAB

Es erfolgt die Aufnahme einer Modellzeitreihe über 1200 s mit zufällig generierten Schaltvorgängen zur Bereitstellung in einer OPCUA Server Simulation; Abbildung 1.

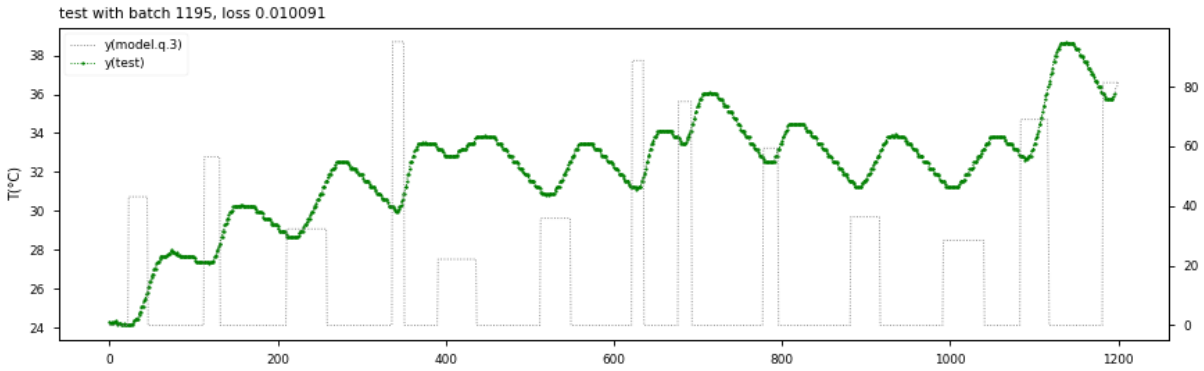

Abbildung 1

OPCUA Server

Die Implementierung eines OPCUA Server mit entsprechender Ereigniss-Loop erfolgt aufbauend auf dem Python Modul *asyncua*. Beispielhaft sei hier nur die dokumentierte *loop_* gezeigt; Abbildung 2. Auf sicherheitsrelevante Konfiguration wird verzichtet. Die Auswertung im Node-Red Flow basiert auf dem Array der OPCUA Variable *a*.

```python
async with self:
    moment = 0
    while not self.cancel:

        nodeid = "ns=2;i=1"

        # retrieve one simulation setup of
        # t_: time, u_: Q-switch, y_: T-temperature by switch
        moment,t_,u_,y_ = await self.moment_(moment)
        u_ = int(u_)
        y_ = float(y_)
        t_ = datetime.now()

        # write to single opcua variables
        await self.write_(nodeid, "u", u_)
        await self.write_(nodeid, "y", y_)
        await self.write_(nodeid, "t", t_)

        # write to an opcua array together
        a = await self.variable_(nodeid, "a")
        y_ = np.int64(y_*1000)
        t_ = np.int64(t_.timestamp())
        value = [u_,y_,t_]
        await self.write_attribute_value(a.nodeid,ua.DataValue(value))

        await asyncio.sleep(self.delay)

        if self.count == 1:

            # send an alive event, too
            u = await self.read_(nodeid, "u")
            y = await self.read_(nodeid, "y")
            t = await self.read_(nodeid, "t")
            await self.event_(message='alive',attr=dict(u=u,y=y,t=t))

            self.count = 0

        self.count += 1
```

Abbildung 2

Node-Red

Die Auswertung und Weiterleitung der OPCUA Daten erfolgt m. H. eines Flows; Abbildung 3. Dabei werden die OPCUA Array Daten in passende T und Q für Temperatur- und Schaltvorgangswerte zerlegt und an einen MQTT Server in der Topic-Nomenklatur <Source>/<Sensor>/<Field> weitergereicht.

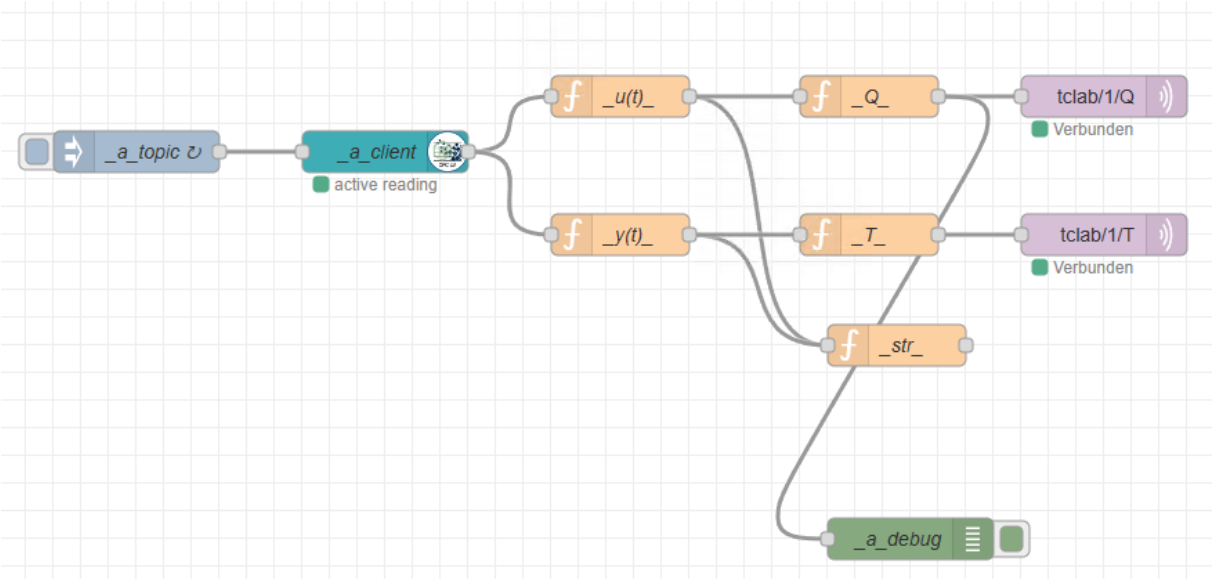

Abbildung 3

Die wichtigsten Bausteine sind wie folgt zu dokumentieren:

_a_topic	Trigger, zyklisch alle 1s auf msg.topic=ns=2;i=5
_a_client	OPCUA-Client; READ auf opc.tcp://localhost:4841/freeopcua/server/
_u(t)_	Payload Konverter; Abbildung 4
_Q_	Payload Konverter für MQTT; Abbildung 5
tclab/1/Q	MQTT Out auf Topic
_y(t)_	Payload Konverter; äquivalent zu u, jedoch mit _v aus Index 1
_T_	Payload Konverter für MQTT; äquivalent zu _Q_, jedoch mit Topic="T"
tclab/1/T	MQTT Out auf Topic

```
var _t = msg.payload[2][1]
var _v = msg.payload[0][1]

_v = _v > 100 || _v < 0 ? undefined : _v;

var _msg = {
    topic: "u",
    payload: {
        "t": _t,
        "v": _v,
        "unit": "%"
    }
    };

return _msg;
```

Abbildung 4

```
msg.topic = 'Q'
msg.payload = msg.payload['v']
return msg;
```

Abbildung 5

Aedes-Cli

Die Installation des MQTT-Brokers erfolgt mit Standard-Einstellungen.

npm install -g aedes-cli

Start auf der Kommandozeile mit
aedes

Damit ist auf localhost:1883 ein MQTT Server als MQTT Relay/Broker in Betrieb genommen.

Telegraf

Das Mapping der MQTT Nachrichten und die Weitergabe an die InfluxDB erfolgt innerhalb der inputs.conf (Abbildung 6) und outputs.conf (Abbildung 7) einer Telegraf Konfiguration; hier nur wichtige Ausschnitte, um die InfluxDB anzusprechen.

```
[[inputs.mqtt_consumer]]
servers=["tcp://localhost:1883"]
data_format="value"
data_type="float"

topics=["tclab/1/Q","tclab/1/T"]

[[inputs.mqtt_consumer.topic_parsing]]
topic="+/+/Q"
measurement="measurement/_/_"
tags=""


[[inputs.mqtt_consumer.topic_parsing]]
topic="+/+/T"
measurement="measurement/_/_"
tags=""
```

Abbildung 6

```
[[outputs.influxdb_v2]]
  # URL to InfluxDB cloud or your own instance of InfluxDB 2.0
  urls = ["http://localhost:8086"]
  ## Token for authentication.
  token = "A2_IJpLrJikf4ULwetngKGGM1DExnmAvgxGJVXoT20dbvCW6uOpGUofK2pGeGNvk4KLMbd6ndt6iord0M8kpXg=="
  ## Organization is the name of the organization you wish to write to; must exist.
  organization = "tclab"
  bucket = "tclab"
```

Abbildung 7

Die outputs.conf beschreibt das Zielsystem; hier eine InfluxDB; in deren Bucket nun die Zeitreihen-Topic's aus der inputs.conf gespeichert werden.

Der Aufruf von Telegraf auf einem Windowssystem erfolgt mit

.\telegraf.exe -config-directory conf

Im conf-Verzeichnis sind die inputs.conf und outputs.conf angelegt.

InfluxDB

Auf einem Windows-System erfolgt das Ausführen einer InfluxDB mit Hilfe eines vorkompilierten Daemon per influxd.exe. Damit steht unter der Adresse localhost:8086 die Datenbank-Web-Oberfläche zur Verfügung, welche nun zur Konfiguration verwendet wird. Für das avisierte Bucket „tclab“ benötigen wir ein TOKEN (siehe Telegraf, outputs.conf) als Authentisierung.

Eine Beispielabfrage auf die Zeitreihendaten des Bucket wäre möglich wie in Abbildung 8.

```
from(bucket: "tclab")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "tclab")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["topic"] == "tclab/1/Q")
```

Abbildung 8

Anschaulicher gestaltet sich doch die Datenlage über ein Dahsboard, wie in Abbildung 9, Beginn und Ende mit rotem Absatz; man vergleiche mit den Simulationsdaten aus Abbildung 1.

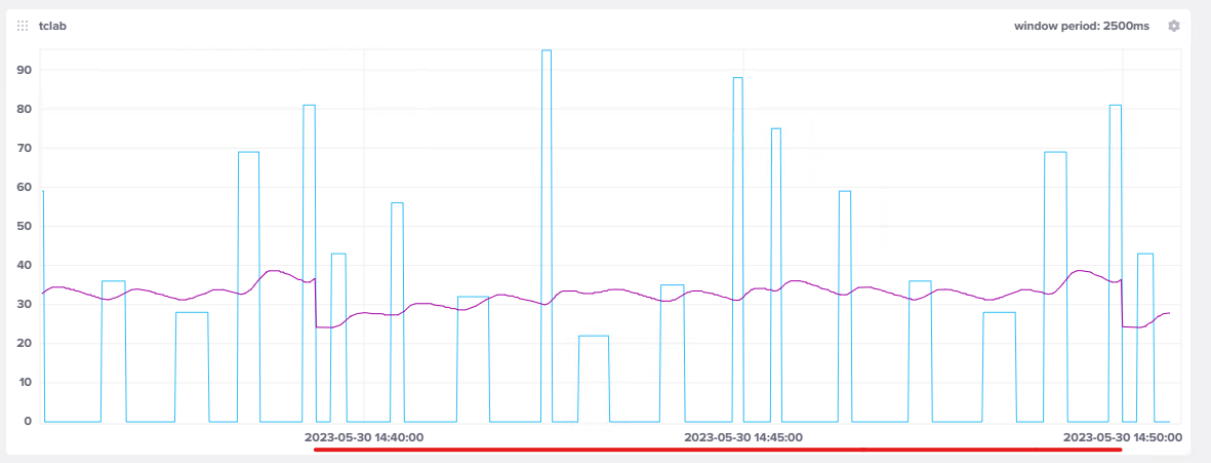

Abbildung 9

Folgende Query entspricht oben gezeigter Abbildung 9.

```
from(bucket: "tclab")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "tclab")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["topic"] == "tclab/1/T" or r["topic"] == "tclab/1/Q")
```