Anlage II: Quellcode des pv_collector.py Skripts

Beschreibung

Dieses Python-Skript dient als zentrale Komponente zur Datenerfassung von einem Eastron SDM120 Modbus-Zähler und zur Weiterleitung der Daten an einen MQTT-Broker.

Quellcode

1. Importe und Konfiguration

Hier werden notwendige Bibliotheken geladen und Umgebungsvariablen (Environment Variables) eingelesen. Dies ermöglicht die Konfiguration über Docker Compose.

```
import os
import time
import json
import logging
from datetime import datetime
import paho.mqtt.client as mqtt
import minimalmodbus
import serial

KONFIGURATION

MQTT Einstellungen
    MQTT_BROKER = os.getenv("MQTT_BROKER_HOST", "mosquitto")
    MQTT_TOPIC = os.getenv("MQTT_TOPIC", "pv/anlage/data")

Modbus Einstellungen (Verbindung zum Zähler)
    MODBUS_PORT = os.getenv("MODBUS_PORT", "/dev/ttyUSB0")
    MODBUS_BAUDRATE = int(os.getenv("MODBUS_BAUDRATE", 9600))
    SLAVE_ID = 1

Datei zur Speicherung des Zählerstands
    STATE_FILE = "/app/data/meter_state.json"

Logging-Konfiguration
    logging.basicConfig(format='%(asctime)s %(levelname)s: %(message)s',
    level=logging.INFO)
```

2. Zustandsverwaltung (State Management)

Diese Funktionen speichern und laden den Zählerstand vom Tagesbeginn, um den Tagesertrag berechnen zu können.

ZUSTANDSVARIABLEN

```
midnight_counter_kwh = 0.0  # Zählerstand um 0:00 Uhr
last_reset_day = -1
def load_state():
    """Lädt den gespeicherten Zustand beim Start."""
    global midnight_counter_kwh, last_reset_day
    if os.path.exists(STATE_FILE):
        try:
            with open(STATE_FILE, 'r') as f:
                state = json.load(f)
                midnight_counter_kwh = state.get('midnight_kwh', 0.0)
                last_reset_day = state.get('day', -1)
                logging.info(f"Zustand geladen. Startwert: {midnight_counter_kwh} kWh")
        except Exception as e:
            logging.error(f"Fehler beim Laden des Zustands: {e}")
def save_state(current_total_kwh):
    """Speichert den aktuellen Zählerstand als neuen Tagesstartwert."""
    global midnight_counter_kwh, last_reset_day
    now = datetime.now()
    try:
        os.makedirs(os.path.dirname(STATE_FILE), exist_ok=True)
        state = {
            'midnight_kwh': current_total_kwh,
            'day': now.day,
            'updated': str(now)
        }
        with open(STATE_FILE, 'w') as f:
            json.dump(state, f)
        midnight_counter_kwh = current_total_kwh
        last_reset_day = now.day
        logging.info(f"NEUER TAG. Startwert gesetzt auf: {current_total_kwh} kWh")
    except Exception as e:
        logging.error(f"Fehler beim Speichern des Zustands: {e}")
```

3. Modbus-Kommunikation

Diese Funktion liest die elektrischen Parameter (Spannung, Strom, Leistung, Gesamtenergie) vom Zähler aus.

```
def get_modbus_data(instrument):
    """Liest Register vom Eastron SDM120."""
    data = {}
    try:
        # 1. Spannung (Protokolladresse 0x0000)
        data["spannung_V"] = round(instrument.read_float(0, functioncode=4,
number_of_registers=2), 2)
        # 2. Strom (Protokolladresse 0x0006)
        data["strom_A"] = abs(round(instrument.read_float(6, functioncode=4,
number_of_registers=2), 2))
        # 3. Leistung (Protokolladresse 0x000C)
        data["leistung_W"] = abs(round(instrument.read_float(12, functioncode=4,
number_of_registers=2), 2))
        # 4. Gesamtenergie (Protokolladresse 0x0156)
        data["total_kwh"] = abs(round(instrument.read_float(342, functioncode=4,
number_of_registers=2), 4))
        return data
    except Exception as e:
        logging.error(f"Modbus-Lesefehler: {e}")
        return None
```

4. Hauptschleife (Main Loop)

Hier wird die Verbindung initialisiert und die Daten werden zyklisch abgerufen und per MQTT versendet.

```
def main():
    global midnight_counter_kwh, last_reset_day
    # Zustand laden
    load_state()
    # MQTT Verbindung herstellen
    mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    try:
        mqtt_client.connect(MQTT_BROKER, 1883, 60)
        mqtt_client.loop_start()
        logging.info(f"Verbunden mit MQTT Broker: {MQTT_BROKER}")
    except Exception as e:
```

```
    logging.error(f"MQTT Fehler: {e}")
    return
# Modbus Initialisierung
try:
    instrument = minimalmodbus.Instrument(MODBUS_PORT, SLAVE_ID)
    instrument.serial.baudrate = MODBUS_BAUDRATE
    instrument.serial.bytesize = 8
    instrument.serial.parity = serial.PARITY_NONE
    instrument.serial.stopbits = 1
    instrument.serial.timeout = 1
    logging.info(f"Modbus konfiguriert: {MODBUS_PORT}")
except Exception as e:
    logging.error(f"Modbus Konfigurationsfehler: {e}")
    return
logging.info("Starte Datenerfassung...")
while True:
    try:
        # Daten lesen
        readings = get_modbus_data(instrument)
        if readings and "total_kwh" in readings:
            current_total = readings["total_kwh"]
            now = datetime.now()
            # Tageswechsel erkennen
            if last_reset_day != now.day:
                logging.info("Tageswechsel erkannt.")
                save_state(current_total)
            # Erster Start Initialisierung
            if midnight_counter_kwh == 0.0 and current_total > 0:
                save_state(current_total)
            # Tagesertrag berechnen
            daily_yield = current_total - midnight_counter_kwh
            if daily_yield < 0: daily_yield = 0.0
            # JSON Payload erstellen
            payload = {
                "spannung_V": readings.get("spannung_V", 0.0),
                "strom_A": readings.get("strom_A", 0.0),
                "leistung_W": readings.get("leistung_W", 0.0),
                "tagesertrag_kWh": round(daily_yield, 3),
                "total_kWh": current_total,
                "timestamp": int(time.time())
            }
            # An MQTT senden
```

```
                mqtt_client.publish(MQTT_TOPIC, json.dumps(payload))
                logging.info(f"Daten gesendet: {payload['leistung_W']}W")
            else:
                logging.warning("Keine gültigen Daten vom Zähler.")
            time.sleep(5)
        except KeyboardInterrupt:
            break
        except Exception as e:
            logging.error(f"Fehler in Hauptschleife: {e}")
            time.sleep(10)
    mqtt_client.loop_stop()
if __name__ == "__main__":
    main()
```