

PiLogger Prometheus

Zuerst müssen braucht es das Python Paket `prometheus-client`

```
# Installieren von pip
sudo apt-get install python3-pip

# Installieren von prometheus-client
sudo pip3 install prometheus-client
```

Einige PiOS Versionen werden sich beschweren und vorschlagen, das Paket lieber via `apt` zu installieren:

```
sudo apt install python3-prometheus-client
```

Jetzt kann die `PiLogger-bottle.py` modifiziert werden. Ich empfehle, vorher die originale Datei zu sichern.

```
cp PiLogger-bottle.py PiLogger-bottle.py.org
```

Jetzt kopieren wir die original Datei noch zu `PiLogger-bottle.py.new`. Später wird die `.new` die originale `PiLogger-bottle.py` ersetzen. Somit kann man einfach zwischen den Versionen hin und her schalten.

Modifikationen von `PiLogger-bottle.py` im Überblick

1. Einbindung von Prometheus Client:

- Import von `Gauge` und `start_http_server` aus dem Modul `prometheus_client`.
- Erstellung von Gauge-Metriken für verschiedene Messgrößen (Temperatur, Leistung, Spannung, Strom, Puls, Widerstand, Bilanz, Ertrag, Verbrauch).

2. Initialisierung der Gauge-Metriken:

- Definition und Initialisierung der Gauge-Metriken für alle zu messenden Parameter direkt nach dem Import der benötigten Module.

3. Erweiterung der Funktion `dotheLog`:

- Nach dem Loggen der Daten in die CSV-Datei werden die Werte in die entsprechenden Prometheus Gauge-Metriken gesetzt.

4. Starten eines HTTP-Servers für die Prometheus-Metriken:

- Implementierung eines HTTP-Servers, der auf Port 8000 die Metriken für Prometheus bereitstellt.
- Dieser wird im `__main__`-Block des Skripts gestartet.

Detaillierte Änderungen

1. Einbindung von Prometheus Client

In der Nähe der anderen Importe, ganz am Anfang des Skripts:

Vorher:

```
import smbus
from gpiozero import Button
import os
import bottle
import json
from datetime import datetime, date, time, timedelta
import threading
import locale
from shutil import copyfile
from time import sleep
from math import log
from PiLo_Thermistor import PtcTable, NtcTable
```

Nachher:

```
import smbus
from gpiozero import Button
import os
import bottle
import json
from datetime import datetime, date, time, timedelta
import threading
import locale
from shutil import copyfile
from time import sleep
from math import log
from PiLo_Thermistor import PtcTable, NtcTable
# Import Gauge and start_http_server from prometheus_client
from prometheus_client import Gauge, start_http_server
```

2. Initialisierung der Gauge-Metriken

Direkt unter den Importen:

Vorher:

```
import smbus
# ...
from PiLo_Thermistor import PtcTable, NtcTable
```

Nachher:

```
import smbus
# ...
from PiLo_Thermistor import PtcTable, NtcTable
# Import Gauge and start_http_server from prometheus_client
```

```

from prometheus_client import Gauge, start_http_server

gTmom = Gauge('Temperatur_momentan', 'Temperature measured by the Pt1000 Sensor')
gTavg = Gauge('Temperature_average', 'Average Temperature')
gTmin = Gauge('Temperature_min', 'Minimum Temperature')
gTmax = Gauge('Temperature_max', 'Maximum Temperature')

gWmom = Gauge('Power_momentary', 'Momentary Power')
gWavg = Gauge('Power_average', 'Average Power')
gWmin = Gauge('Power_min', 'Minimum Power')
gWmax = Gauge('Power_max', 'Maximum Power')

gVmom = Gauge('Voltage_momentary', 'Momentary Voltage')
gVavg = Gauge('Voltage_average', 'Average Voltage')
gVmin = Gauge('Voltage_min', 'Minimum Voltage')
gVmax = Gauge('Voltage_max', 'Maximum Voltage')

gAmom = Gauge('Current_momentary', 'Momentary Current')
gAavg = Gauge('Current_average', 'Average Current')
gAmin = Gauge('Current_min', 'Minimum Current')
gAmax = Gauge('Current_max', 'Maximum Current')

gPmom = Gauge('Pulse_momentary', 'Momentary Pulse')
gPavg = Gauge('Pulse_average', 'Average Pulse')
gPmin = Gauge('Pulse_min', 'Minimum Pulse')
gPmax = Gauge('Pulse_max', 'Maximum Pulse')

gRohm = Gauge('Resistance', 'Resistance')
gBila = Gauge('Bila', 'Bila')
gErtr = Gauge('Ertr', 'Ertr')
gVerb = Gauge('Verb', 'Verb')

gBilTg = Gauge('BilTg', 'BilTg')
gErtTg = Gauge('ErtTg', 'ErtTg')
gVerTg = Gauge('VerTg', 'VerTg')

```

3. Erweiterung der Funktion `dotheLog`

In der Funktion `dotheLog`, nach dem Loggen der Daten in die CSV-Datei:

Vorher:

```

def dotheLog():
    # ... bestehender Code ...
    with open(WorkDir+'/showdata.csv','a') as datafile:
        print(line,file=datafile)
    # ... bestehender Code ...

```

Nachher:

```

def dotheLog():
    # ... bestehender Code ...
    with open(WorkDir+'/showdata.csv','a') as datafile:
        print(line,file=datafile)
    # Split the line using ";"

```

```

values = line.split(';')
Tmom = float(values[1]) # Assuming index 2 corresponds to the second value
Tavg = float(values[2])
Tmin = float(values[3])
Tmax = float(values[4])
Wmom = float(values[5])
Wavg = float(values[6])
Wmin = float(values[7])
Wmax = float(values[8])
Vmom = float(values[9])
Vavg = float(values[10])
Vmin = float(values[11])
Vmax = float(values[12])
Amom = float(values[13])
Aavg = float(values[14])
Amin = float(values[15])
Amax = float(values[16])
Pmom = float(values[17])
Pavg = float(values[18])
Pmin = float(values[19])
Pmax = float(values[20])
Rohm = float(values[21])
Bila = float(values[22])
Ertr = float(values[23])
Verb = float(values[24])
BilTg = float(values[25])
ErtTg = float(values[26])
VerTg = float(values[27])
gTmom.set(Tmom)
gTavg.set(Tavg)
gTmin.set(Tmin)
gTmax.set(Tmax)
gWmom.set(Wmom)
gWavg.set(Wavg)
gWmin.set(Wmin)
gWmax.set(Wmax)
gVmom.set(Vmom)
gVavg.set(Vavg)
gVmin.set(Vmin)
gVmax.set(Vmax)
gAmom.set(Amom)
gAavg.set(Aavg)
gAmin.set(Amin)
gAmax.set(Amax)
gPmom.set(Pmom)
gPavg.set(Pavg)
gPmin.set(Pmin)
gPmax.set(Pmax)
gRohm.set(Rohm)
gBila.set(Bila)
gErtr.set(Ertr)
gVerb.set(Verb)
gBilTg.set(BilTg)
gErtTg.set(ErtTg)
gVerTg.set(VerTg)
# ... bestehender Code ...

```

4. Starten eines HTTP-Servers für Prometheus

Im `__main__`-Block des Skripts, direkt vor dem Aufruf der `bottle.run`-Funktion:

Vorher:

```
if __name__ == "__main__":
    button = Button(4)
    button.when_pressed = JetztAber

    bottle.run(host='0.0.0.0', port=8080, quiet=True)

    printD('\nQuit\n')
    printE(getLogTime()+" Quit")
```

Nachher:

```
if __name__ == "__main__":
    # Expose metrics
    metrics_port = 8000
    start_http_server(metrics_port)
    print("Serving sensor metrics on :{}".format(metrics_port))

    button = Button(4)
    button.when_pressed = JetztAber

    bottle.run(host='0.0.0.0', port=8080, quiet=True)

    printD('\nQuit\n')
    printE(getLogTime()+" Quit")
```

Zusammenfassung der Einfügepositionen

1. **Einfügen der Prometheus-Imports:** Direkt nach den anderen Imports am Anfang des Skripts.
2. **Initialisierung der Gauge-Metriken:** Direkt unter den Importen.
3. **Erweiterung der Funktion `dotheLog`:** Nach dem Loggen der Daten in die CSV-Datei, innerhalb der Funktion `dotheLog`.
4. **Starten des HTTP-Servers für Prometheus:** Im `__main__`-Block, direkt vor dem Aufruf der `bottle.run`-Funktion.

Diese Änderungen fügen die notwendigen Funktionen zur Überwachung und Exposition der Sensor-Metriken mithilfe von Prometheus hinzu.

Damit wird auf Port 9095 ein Webserver gestartet und die Daten als Scrape-Target für Prometheus bereitgestellt. Man hat die freie Wahl des Ports. Eine Übersicht über die aktuell belegten Ports kann man folgendermaßen bekommen:

```
sudo ss -ltnp
```

Austausch der PiLogger-bottle.py.new zu PiLogger-bottle.py

```
cp PiLogger-bottle.py.new PiLogger-bottle.py
```

Danach den pilo-webmon Service neu starten:

```
sudo systemctl status pilo-webmon.service
sudo systemctl stop pilo-webmon.service
sudo systemctl start pilo-webmon.service
sudo systemctl status pilo-webmon.service
```

Prometheus scratch config

der Prometheus Exporter auf Port 8000 steht nun bereit. Um die Daten in den Prometheus Server zu bekommen, muss das neue Scrape Target in der `prometheus.yml` hinzugefügt werden.

Tipp: Je nach Installation kann die `prometheus.yml` auch woanders liegen, zB in `/etc/prometheus` o.ä.

```
sudo nano ~/prometheus/prometheus.yml
```

add new job:

```
# my global config
global:
  scrape_interval: 1s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"
```

```
# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
  - targets: ["localhost:9090"]

- job_name: "pi58"
  #honor_labels: true
  static_configs:
    - targets: ["localhost:8000"]
```

Danach den Prometheus Server neu starten:

```
sudo systemctl status prometheus.service
sudo systemctl stop prometheus.service
sudo systemctl start prometheus.service
sudo systemctl status prometheus.service
```

Prometheus Datenlimit ändern

Per default speichert Prometheus die Daten nur 4 Wochen. Das Limit kann man auch hochsetzen, in meinem Fall auf 400GB oder 5 Jahre. Das Limit, welches zuerst erreicht wird, greift.

```
sudo mcedit /etc/systemd/system/multi-user.target.wants/prometheus.service
```

Dann editieren:

```
[Unit]
Description=Prometheus Server
Documentation=https://prometheus.io/docs/introduction/overview/
After=network-online.target

[Service]
User=pi
Restart=on-failure

ExecStart=/home/pi/prometheus/prometheus \
  --config.file=/home/pi/prometheus/prometheus.yml \
  --storage.tsdb.path=/home/pi/prometheus/data \
  --storage.tsdb.retention.size 400GB \
  --storage.tsdb.retention.time 5y \
  --storage.tsdb.retention 5y

[Install]
WantedBy=multi-user.target
```

```
sudo service prometheus restart
```

Grafana 1 Sekunden refresh

Grafana erlaubt standardmäßig minimal 5 Sekunden Aktualisierung. Mit folgender Änderung können wir das Limit auch auf 1s runtersetzen.

Öffne grafana.ini mit vi oder nano und editiere:

```
min_refresh_interval = 1s
```

Danach Grafana neu starten.

```
sudo service grafana restart
```