rsensor Documentation

Author

Oct 26, 2020

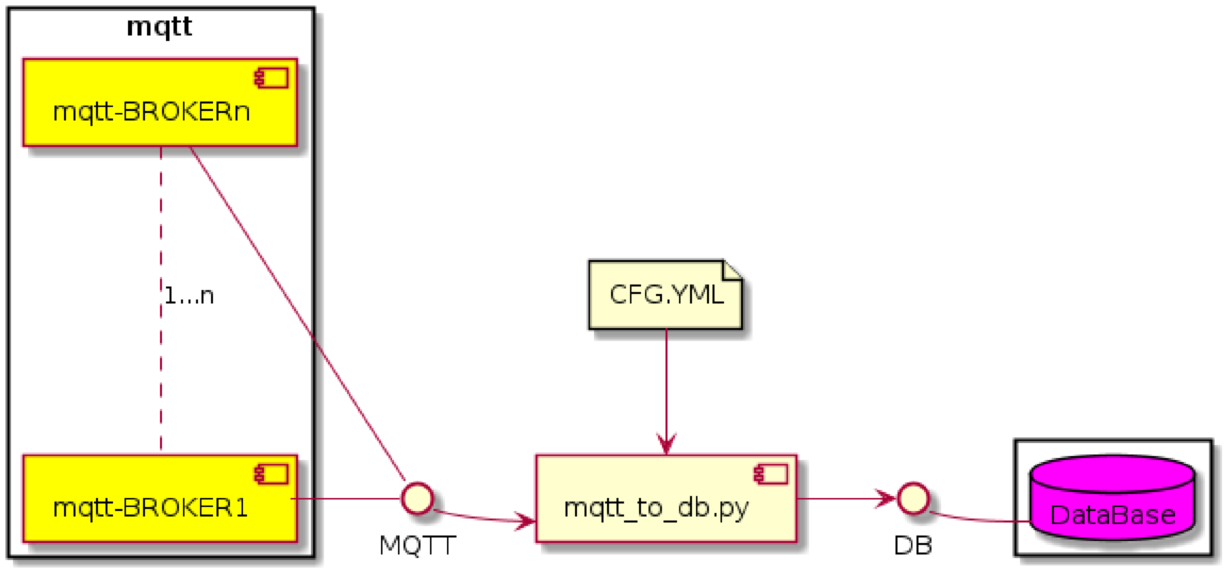
Contents:

The uracoli-rsensor project stores sensor data, received from a MQTT broker in a database.

CHAPTER 1

Introduction

The script `mqtt_to_db.py` reads sensor data from one ore more MQTT brokers and stores it together with additional meta information in a database.



Sensor values are characterized by their meaning, e.g. the physical quantity a value represents. Beside their basic meaning (numeric value and physical unit), a sensor value can also have meta information attached.

The following information shall be stored in the database:

- What was measured? (numeric value and physical unit)
- When? (the timestamp of the measurement)
- Where? (the location of the sensor)
- How? (the type of the sensor, system information, software version, . . .)

CHAPTER 2

Usage

2.1 Installation

The installation of the latest version can be done with:

```
pip install git+https://gitlab.com/uracoli-project/uracoli-rsensor.git
```

2.2 Getting Started

After successfull installation run:

```
mqtt_to_db -h
```

to see the available command line options of the script.

Create now the initial configuration file `myrsensor.cfg`:

```
mqtt:
 - host: test.mosquitto.org
   port: 1883
   prefix: rsensor-test

database:
    dbtype: sqlite
    dbname:  /tmp/rsensor/rsensor.sqlitedb
```

This configuration connects to the public available MQTT broker `test.mosquitto.org` and stores the received data in a SQlite database file `/tmp/rsensor/rsensor.sqlitedb`.

Run the script and see the messages on the console:

```
mqtt_to_db -C myrsensor.cfg
```

The file `/tmp/rsensor/rsensor.sqlitedb` is created and can be accessed with the SQLite tool:

```
sqlite3 /tmp/rsensor/rsensor.sqlitedb
```

The next section describes how to receive test data from MQTT, that are stored in the database

2.3 Generating Test Data

The package contains also a script `rs_testgen` that generates appropriate formatted test data.

The script uses per default the MQTT broker on `localhost:1883`.

After start you will see an output:

```
$ rs_testgen -H test.mosquitto.org
INFO:MqttRandomGenerator:create generator class
using default sensor config: /opt/esp/ve_esp/local/lib/python2.7/site-packages/
↪rsensor/data/test_node.yml
INFO:MqttRandomGenerator:connected to test.mosquitto.org:1883, prefix=rsensor-test
INFO:MqttRandomGenerator:info message on: rsensor-test/s_1111_0001/info
INFO:MqttRandomGenerator:info message on: rsensor-test/s_1111_0002/info
INFO:MqttRandomGenerator:info message on: rsensor-test/s_1111_0003b/info
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0001/data
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0002/data
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0002/data
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0003b/data
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0001/data
INFO:MqttRandomGenerator:data message on: rsensor-test/s_1111_0002/data
```

With the command `mosquitto_sub -h localhost -t "rsensor-test/#" -v` you can watch the generated messages.

See `rs_testgen -h` for available configuration options.

CHAPTER 3

Config File Format

3.1 Format

The config file is written in YAML syntax.

The file contains a YAML-dictionary with two predefined keywords:

- mqtt
- database

3.2 MQTT Section

This section starts with the YAML-key `mqtt` and contains a list of MQTT-Brokers:

```
mqtt:
 - host: iot.eclipse.org
   port: 1883
   prefix: 'clt2020/thlog'
   info_topic: 'clt2020/thlog/+/info'
   data_topic: 'clt2020/thlog/+/data'
 - host: test.mosquitto.org
   port: 1883
   prefix: 'aw/thlog'
   info_topic: 'aw/thlog/+/info'
   data_topic: 'aw/thlog/+/data'
```

Each configured MQTT broker requires a YAML structure with the following keys:

host hostname or IP address of the broker

port portnumber of the mqtt service (1883 unencrypted, 3883 TLS secured)

prefix the subscritpion prefix (mqtt path to device level)

info_topic the topic where the nodes publish their description information per default (if not given) *<prefix>/+/info* is used.

data_topic the topic where the nodes publish their sensor data per default (if not given) *<prefix>/+/data* is used.

Example

:: myprefix/mydevice/info myprefix/mydevice/data <prefix>: myprefix info_topic: +/info data_topic: +/data

3.3 Database section

This section starts with the YAML-key `database` and requires the `dbtype` key:

```
database:
    dbtype: {sqlite|mysql}
```

The rest of the data-structure depends on the database type used.

3.3.1 SQLITE Database

The SQLITE DB is configured as follows:

```
database:
    dbtype: sqlite
    dbname:  /tmp/rsensor/rsensor.sqlitedb
```

dbtype *sqlite*

dbname the path and name of the SQLITE-file

If `dbname` refers to a file that not exists, the file, including a none existing directory, is created and initialized as sqlite database.

The data in the sqlite database can be accessed with the command:

```
sqlite3  /tmp/rsensor/rsensor.sqlitedb
sqlite> .tables
locations   nodes       sensors     timeseries
```

More details about how to work with this DB refer to section *SQlite Commands*.

3.3.2 MySQL Database

A MySQL Database is configured as follows:

```
database:
    dbtype: mysql
    dbname: rsensordb
    dbuser: rsensorusr
    dbpasswd: topsecret
    dbhost: database-server.mydomain.com
```

dbtype *mysql*

dbname database name on the server

dbuser database user name that has write access to the database

dbpasswd password for database user

dbhost server name that hosts the database (hostname, fqdn or IP number)

The database *dbname* must exist and the the *dbuser* must have access. If the database is empty, all required tables are created by the script.

The created tables can be listed with the command:

```
mysql -h <dbhost> -u <dbuser> -p <dbname>
Enter password: <dbpasswd>
mysql> show tables;
+--------------------+
| Tables_in_rsensor |
+--------------------+
| locations          |
| nodes              |
| sensors            |
| timeseries         |
+--------------------+
4 rows in set (0,00 sec)
```

CHAPTER 4

MQTT Interface

4.1 Info Topic rsensor/node/+/info[r]

The =node_info= topuc is a retained topic that contains information about the sensors and actors that it contains. An example of the JSON-data is given here:

```
{
    "actors": [],
    "node": {
        "id": "51fe4a00",
        "name": "thlog-51fe4a00",
        "board": "esp8266+bm280",
        "firmware": "TempHumLogger.py",
        "version": "1.02"
        "geoloc": [ 1, 2],
        "loctag": "WH.KG.K2",
    },
    "sensors": [
        {
            "id": "temp",
            "name": "temp",
            "type": "float",
            "unit": "\u00b0C"
        },
        {
            "id": "rh",
            "name": "rh",
            "type": "float",
            "unit": "%"
        }            ]
}
```

4.1.1 Data Structure

The data in the info topic is a dictionary with three keywords:

```
{
        "node": {<node_description>}
        "sensors": [{<sensor_description>}, {...}, ...]
        "actors": [{<actor_description>}, {}, ...]
}
```

4.1.2 Node Information <node_description>

name	description
id	unique id of the sensor node
name	symbolic name
board	type of the board
firmware	name of the firmware image
version	version of the firmware image
loctag	location description
geoloc	geographic location of the node (optional)

4.1.3 Sensor Information <sensor_description>

name	description
id	identifier of the sensor, e.g. 'temp'
name	long name of the sensor, e.g. 'Temperature'
type	data type, e.g. float, int, bool, ...
unit	name of the measurment unit

4.1.4 Actor Information <actor_description>

name	description
id	identifier of the sensor, e.g. 'coil1'
...	...

4.2 Data Topic rsensor/node/+/sensor/data

In this topic data from sensors are published.

```
{ '<sensor_id>' : value,
  '<sensor_id1>' : value
}
```

Note: A published sensor_data message need not to contain all data defined in node_info.sensors. This is because of, that the sensors might have different update rates.

CHAPTER 5

Database Interface

5.1 Supported Databases

- SQlite
- MySQL

5.2 Tables, Views and Indexes

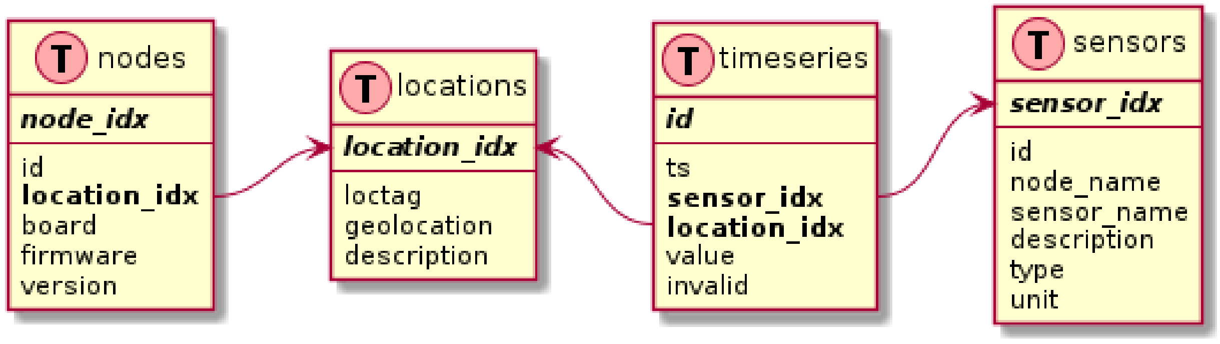
The database definition is stored in the YAML-file `data/schema.yml`.

5.2.1 Tables

The rsensor-database consists of four tables:

- nodes
- locations
- sensors
- timeseries

The following diagram shows the relations between the table fields.



The table *nodes* stores node specific information (see MQTT-info topic):

```
mysql> select * from nodes;
+----------+----------+--------------+----------------+------------------+---------+
| node_idx | id       | location_idx | board          | firmware         | version |
+----------+----------+--------------+----------------+------------------+---------+
|        1 | 36004b00 |            1 | esp8266+bm280  | TempHumLogger.py | 1.00    |
+----------+----------+--------------+----------------+------------------+---------+
1 row in set (0.00 sec)
```

The field *location_idx* references to the table *locations*.

The table *locations* stores a list of all seen locations, based on the field *loctag*:

```
mysql> select * from locations;
+--------------+----------+-------------+-------------+
| location_idx | loctag   | geolocation | description |
+--------------+----------+-------------+-------------+
|            1 | WH.KG.K2 | NULL        | NULL        |
+--------------+----------+-------------+-------------+
1 row in set (0.00 sec)
```

Each node serves one or sensors, all of these are stored in the table *sensors*:

```
mysql> select * from sensors;
+------------+---------------+-----------+-------------+-------------+-------+-------+
| sensor_idx | id            | node_name | sensor_name | description | type  | unit  |
+------------+---------------+-----------+-------------+-------------+-------+-------+
|          1 | 36004b00_temp | 36004b00  | temp        | NULL        | float | C    |
|          2 | 36004b00_rh   | 36004b00  | rh          | NULL        | float | %     |
|          3 | 36004b00_p    | 36004b00  | p           | NULL        | float | hPa   |
|          4 | 36004b00_ah   | 36004b00  | ah          | NULL        | float | g/m^3 |
|          5 | 36004b00_svp  | 36004b00  | svp         | NULL        | float | hPa   |
+------------+---------------+-----------+-------------+-------------+-------+-------+
5 rows in set (0.00 sec)
```

The individual measurement values are stored in table *timeseries*:

```
mysql> select * from timeseries;
+-----+------------+------------+--------------+----------+---------+
| idx | ts         | sensor_idx | location_idx | value    | invalid |
+-----+------------+------------+--------------+----------+---------+
|   1 | 1586237223 |          4 |            1 | 6.21505  |       0 |
|   2 | 1586237223 |          2 |            1 |       34 |       0 |
|   3 | 1586237223 |          5 |            1 | 24.8118  |       0 |
```

(continues on next page)

(continued from previous page)

```
|   4 | 1586237223 |          1 |            1 |   20.97 |       0 |
|   5 | 1586237223 |          3 |            1 |    1019 |       0 |
|   6 | 1586237258 |          4 |            1 | 6.21867 |       0 |
|   7 | 1586237258 |          2 |            1 |      34 |       0 |
|   8 | 1586237258 |          5 |            1 | 24.8271 |       0 |
|   9 | 1586237258 |          1 |            1 |   20.98 |       0 |
|  10 | 1586237258 |          3 |            1 |    1019 |       0 |
+-----+------------+------------+--------------+---------+---------+
10 rows in set (0.00 sec)
```

5.2.2 Views

Todo: add support views in schema.yml

5.2.3 Indexes

Todo: describe indexes

5.3 SQlite Commands

The initial content of the SQLite database can be explored with the commands `.tables` and `.schema <tablename>`:

```
sqlite> .tables
locations   nodes       sensors     timeseries
sqlite> .schema nodes
CREATE TABLE `nodes`
(
   node_idx INTEGER PRIMARY KEY NOT NULL  /*!40101 AUTO_INCREMENT */,
   id VARCHAR(64) DEFAULT NULL,
   location_idx INTEGER DEFAULT -1,
   board VARCHAR(64) DEFAULT NULL,
   firmware VARCHAR(64) DEFAULT NULL,
   version VARCHAR(64) DEFAULT NULL
);
sqlite>
```

If data was received, the table `timeseries` contains data:

```
sqlite> select * from timeseries

idx         ts          sensor_idx  location_idx  value       invalid
----------  ----------  ----------  ------------  ----------  ----------
1           1586844220  4           1             5.68866     FALSE
2           1586844220  2           1             29.0        FALSE
3           1586844220  5           1             26.7363     FALSE
4           1586844220  1           1             22.19       FALSE
```

(continues on next page)

(continued from previous page)

```
5           1586844220  3           1             1010.0      FALSE
6           1586844352  4           1             5.46655     FALSE
7           1586844352  2           1             29.0        FALSE
8           1586844352  5           1             25.6324     FALSE
9           1586844352  1           1             21.5        FALSE
10          1586844352  3           1             1011.0      FALSE
```

More about the representation of the data in the `timeseries` table can be found in section *Database Interface*.

5.4 MySQL Commands

Query a specific sensor value:

```
select from_unixtime(t.ts), s.id, t.value, s.unit from timeseries as t join sensors␣
↪as s on t.sensor_idx = s.sensor_idx where s.id like "%ah";
```

```
SELECT
    FROM_UNIXTIME(t.ts), s.id, t.value, s.unit
FROM
    timeseries AS t
        JOIN
    sensors AS s ON t.sensor_idx = s.sensor_idx
WHERE
    s.id LIKE '%ah';
```

CHAPTER 6

Using the Sensor Data

6.1 Dashboards with Grafana

A Dashboard displays the collected sensor data in a graphical user interface (GUI). They are used to monitor the current and past state of a technical system. The data are repesented by widgets like graphs, bar-charts, gauges and heat-maps. An easy to use and powerful application is Grafana, https://grafana.com.

Install and start a standard grafana container:

```
docker pull grafana/grafana
docker run -d -p 3000:3000 grafana/grafana
```

Open the URL http://172.17.0.1:3000 in the Web-Browser (usually the docker container runs on this address). Follow the instructions to set a new password, initial login is done with user: admin, password: admin.

Setup a Data Source

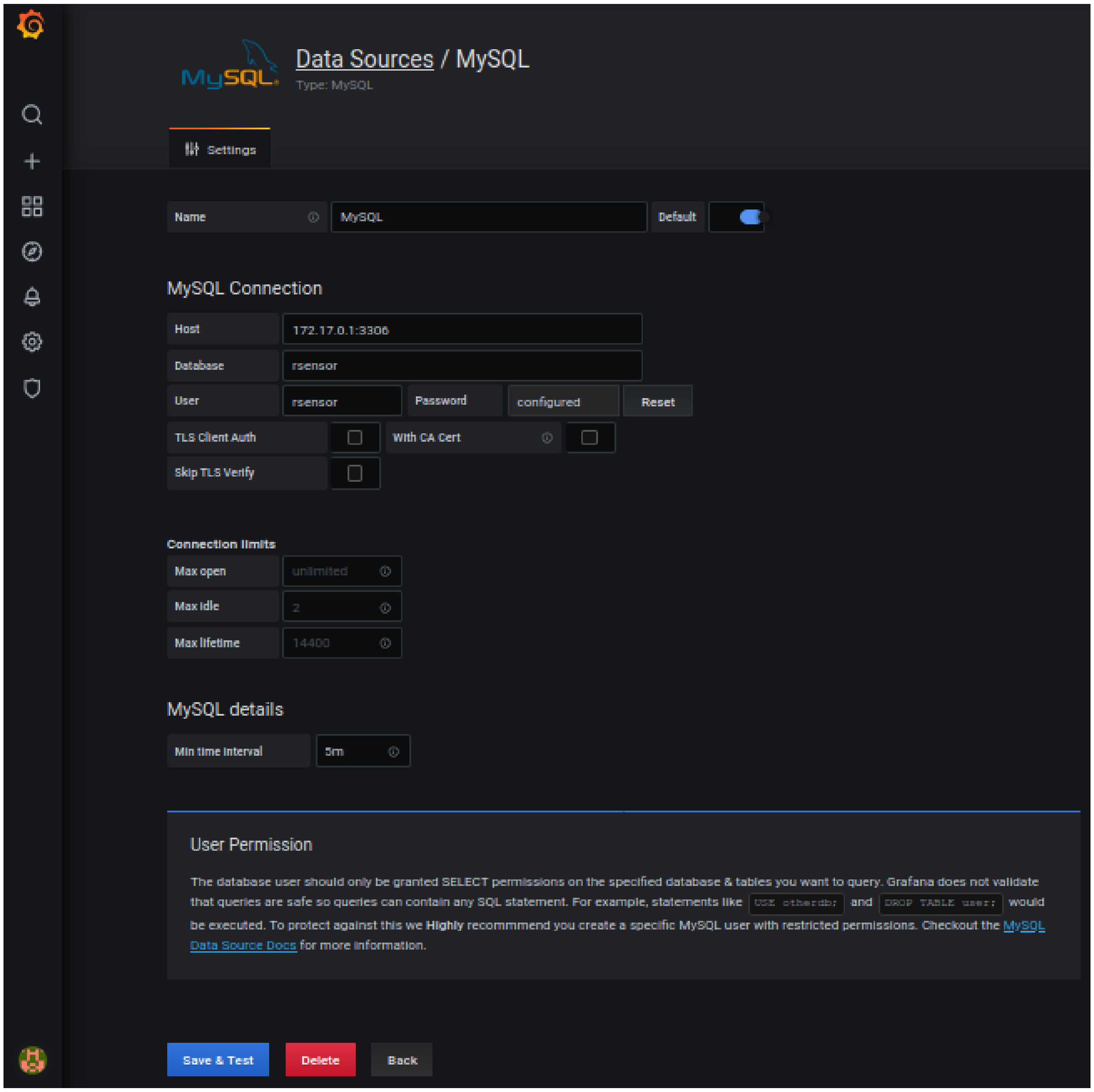
At first Configure the MySQL-DB used by rsensor as data source in grafana. After all parameters are entered, press "Save and Test".

Setup a Dashboard with a Panel

If the test was succesful, the next step is to configure a dashboard. Select "Create Dashboard" and "Add Panel".

In the "Query" tab press "Edit SQL" and enter the following query:

```
select t.ts as time_sec,
       s.id as metric,
       t.value
  from timeseries as t
  join sensors as s on t.sensor_idx = s.sensor_idx
  where s.id like "%temp"
```



Name	MySQL	Default	✓

MySQL Connection

Host	172.17.0.1:3306			
Database	rsensor			
User	rsensor	Password	configured	Reset
TLS Client Auth	☐	With CA Cert	☐	
Skip TLS Verify	☐			

Connection limits

Max open	unlimited
Max idle	2
Max lifetime	14400

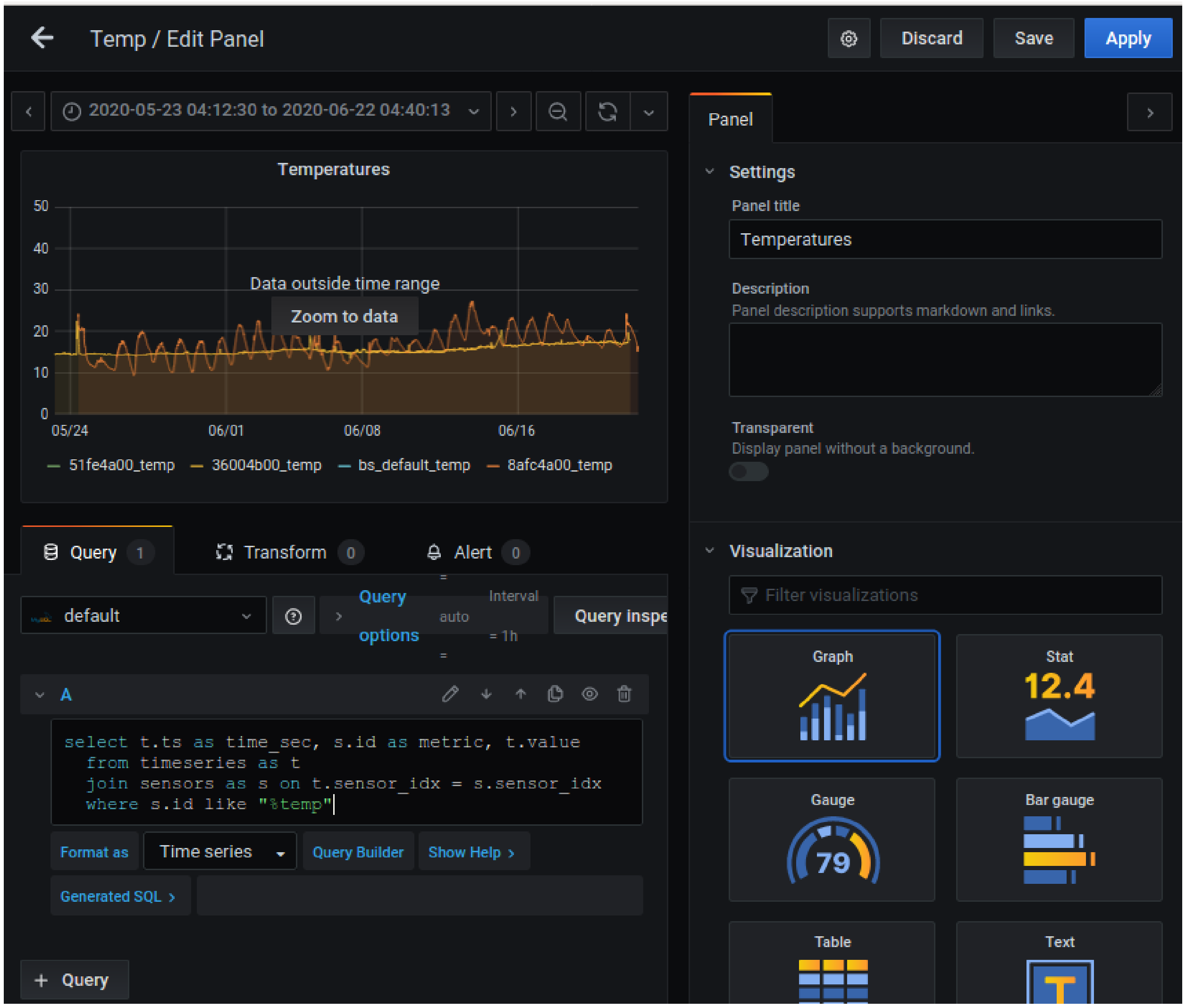
MySQL details

Min time interval	5m

User Permission

The database user should only be granted SELECT permissions on the specified database & tables you want to query. Grafana does not validate that queries are safe so queries can contain any SQL statement. For example, statements like `USE otherdb;` and `DROP TABLE user;` would be executed. To protect against this we **Highly** recommmend you create a specific MySQL user with restricted permissions. Checkout the MySQL Data Source Docs for more information.

Fig. 1: Configure dialog of the data source



Data source: default

```
select t.ts as time_sec, s.id as metric, t.value
  from timeseries as t
  join sensors as s on t.sensor_idx = s.sensor_idx
  where s.id like "%temp"
```

Format as: Time series

Panel title: Temperatures

Description: Panel description supports markdown and links.

Transparent: Display panel without a background.

Fig. 2: Configure dialog of a panel

Use and fine tune the Dashboard

Finally press “Apply” and watch the result.

By entering the “Edit” submenue, the dashboard can be further optimized and fine tuned.

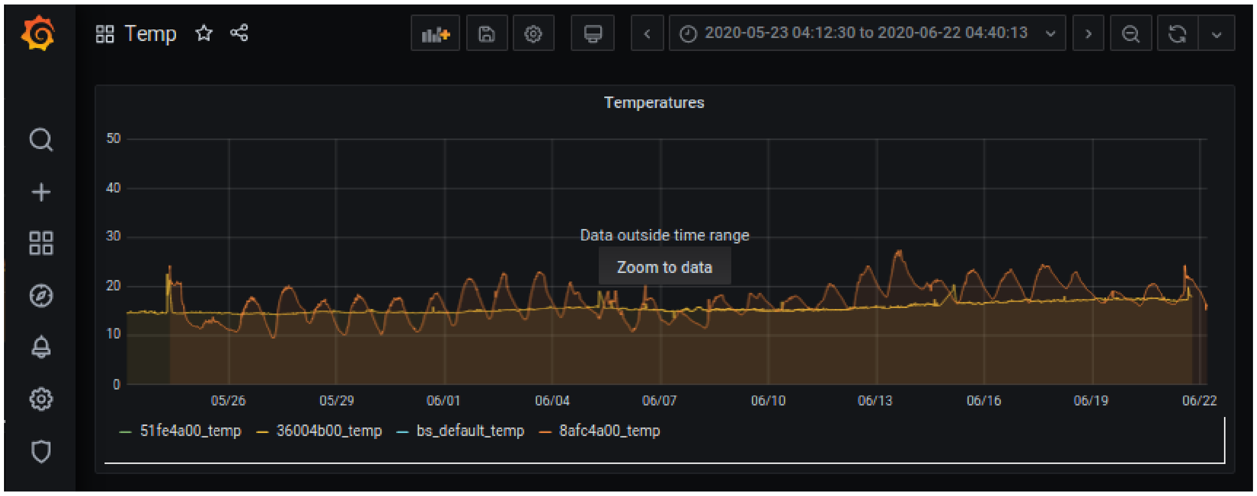


Fig. 3: Dashboard from the above example

6.2 Data Analysis with JupyterLab

. . . coming soon

CHAPTER 7

Raspberry Pi as AP with Rsensor Server

This is a collection of notes on how to configure a Raspberry Pi as IOT node.

7.1 Initial Setup

7.1.1 Prepare Image

- Copy image to SD Card: https://www.raspberrypi.org/documentation/installation/installing-images/ or:

```
sudo dd if=2020-05-27-raspios-buster-lite-armhf.img  bs=4M status=progress␣
↪conv=fsync of=/dev/sdb
```

- Mount card (on Linux)
- To enable SSH server (just create an empty file):

```
touch  /<SD-MOUNTPOINT>/boot/ssh
```

7.1.2 Configure Image

- login to raspi (console or with `ssh pi@<RASPI-IP>`)
- expand file system and reboot:

```
sudo raspi-config
"7 Advanced ..." / "A1 Expand Filesystem"
"Finish" / "Reboot" => YES
```

- wait until reboot is finished and login again

7.2 Install Software

Do the following steps:

```
sudo apt update
sudo apt upgrade # optional, may take longer

sudo apt install mosquitto
```

Prepare a virtual environment that runs uracoli-rsensor:

```
sudo bash
cd /opt
mkdir -p rsensor
apt install python-pip
python3 -m pip2 install virtualenv
python3 -m virtualenv ve_rsensor
ve_rsensor/bin/pip install uracoli-rsensor
```

To install alternatively the latest development version, run this command:

```
/opt/rsensor/ve_rsensor/bin/pip install -e git+https://gitlab.com/uracoli-project/
↪uracoli-rsensor.git
```

Write a config file `/opt/rsensor/rsensor.cfg`, e.g.:

```
mqtt:
 - host: 10.65.87.1
   port: 1883
   prefix: 'mkawdt'

database:
    dbtype: mysql
    dbname: rsensor
    dbuser: rsensor
    dbpasswd: rsensor
    dbhost: 172.16.1.20
```

7.3 Writing a SystemD-Daemon Script

To collect sensor data over a long time, the script `mqtt_to_db` can run as Daemon-service on a Linux server, e.g.on a Raspberry-Pi.

Create a file `/etc/systemd/system/rsensor.service` with the following contents:

```
[Unit]
Description=MqttToDb Service
After=network-online.target

[Service]
Type=simple
User=pi
Group=pi
WorkingDirectory=/opt/rsensor
ExecStart=/opt/rsensor/ve_rsensor/bin/mqtt_to_db -C /opt/rsensor/rsensor.cfg -L ERROR
```

(continues on next page)

(continued from previous page)

```
SyslogIdentifier=rsensor
StandardOutput=syslog
StandardError=syslog
Restart=always
RestartSec=3

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

If you choose another location then `/opt/rsensor`, then adapt in the file above adapt the path-names (see also section python-ve) . Also note the option `-L ERROR`, which reduces the amount of messages written to the system-log. In case of problems, change it temporarily to `-L DEBUG`

The script `rsensor.service` is activated with the following commands:

```
systemctl daemon-reload
systemctl enable rsensor.service
systemctl start rsensor.service
```

The status of the script can be verified with these commands:

```
# see if the script is running or it is crashed.
systemctl status rsensor.service

# see all log messages from the beginning of the log.
journalctl -u rsensor.service

# see actual incoming messages
journalctl -u rsensor.service -f
```

The script can be stopped with:

```
systemctl stop rsensor.service
```

7.4 Configure Raspi Rsensor AP

Follow the article How to use your Raspberry Pi as a wireless access point (https://thepi.io) and skip the bridge settings in this article, they are not needed for the sensor network. (alternativly see also Raspberry Pi als WLAN-Router einrichten)

Use `raspi-config` and set country code to “DE” or the country where the raspi will be operated.

Software to install:

```
sudo apt install hostapd dnsmasq
```

Before Edit cofig, stop daemons:

```
sudo systemctl stop hostapd
sudo systemctl stop dnsmasq
```

Here are the current used configuration files.

`/etc/dhcpcd.conf`:

```
hostname
clientid
persistent
option rapid_commit
option domain_name_servers, domain_name, domain_search, host_name
option classless_static_routes
option interface_mtu
require dhcp_server_identifier
slaac private
interface wlan0
static ip_address=10.65.87.1/24
denyinterfaces eth0
denyinterfaces wlan0
```

`/etc/dnsmasq.conf`:

```
interface=wlan0
  dhcp-range=10.65.87.100,10.65.87.200,255.255.255.0,24h
```

`/etc/hostapd/hostapd.conf`:

```
interface=wlan0
#bridge=br0
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ssid=MyRsensorAP
wpa_passphrase=sovershennosekretno
```

`/etc/default/hostapd`:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

`/etc/sysctl.conf`:

```
net.ipv4.ip_forward=1
```

`/etc/rc.local`:

```
#!/bin/sh -e
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
# appended iptables-restore for AP, generated by:
#       sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
#       sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
iptables-restore < /etc/iptables.ipv4.nat
exit 0
```

Note: Also check the permissions of /etc/rc.local. It needs to be executable. `-rwxr-xr-x 1 root root 306 Apr 4 2017 /etc/rc.local`

Enable and restart the services:

```
systemctl enable hostapd dnsmask
systemctl unmask hostapd dnsmask
systemctl start hostapd dnsmask
```

7.4.1 Bind USB Stick to fixed interface address

Adapt udev rules:

```
cat /etc/udev/rules.d/72-xxx.rules
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="b8:27:eb:31:22:db", NAME="wlan0"
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="b8:27:eb:64:77:8e", NAME="eth0"
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="d0:37:45:70:c6:88", NAME="wlan1"
```

7.4.2 Wifi Issues

Be carefull with trouble shooting, the trouble always shoots back.

Note: After hours of wasted time, the use of the Wifi stick WN722N was abondoned.

After a week of operation, the WN722N blinked more frequently then before and a setup of the AP was not possible, even not after cold start or normal reboot.

Also switchin back to the internal Wlan0 does not bring back wifi AP. Hence the SD Card was somehow damaged and was installed again from scratch.

Not sure if this was the reason, but I used an IOT device which frequently reconnects to the AP because of deep sleep (the error occured after > 2500 reconnects).

In case of an error message from `hostapd`:

```
>> journalctl -u hostapd.service
-- Logs begin at Sat 2020-06-06 12:17:01 CEST, end at Sat 2020-06-06 13:16:17 CEST. --
Jun 06 12:46:06 gretel systemd[1]: Starting Advanced IEEE 802.11 AP and IEEE 802.1X/
↪WPA/WPA2/EAP Authenticator...
Jun 06 12:46:07 gretel hostapd[379]: Configuration file: /etc/hostapd/hostapd.conf
Jun 06 12:46:07 gretel hostapd[379]: nl80211: Driver does not support authentication/
↪association or connect commands
Jun 06 12:46:07 gretel hostapd[379]: nl80211: deinit ifname=wlan0 disabled_11b_rates=0
```

If a Wifi-USB stick like TP-LINK TL-WN722N is used, you need to install new drivers. Easy way is using MrEngmans precompiled modules. (https://www.raspberrypi.org/forums/viewtopic.php?f=28&t=62371&sid=97f79dbe9f8ac40727b1c4ba236c9454)

Do this steps on Raspberry:

```
sudo wget http://downloads.fars-robotics.net/wifi-drivers/install-wifi -O /usr/bin/
↪install-wifi
sudo chmod +x /usr/bin/install-wifi
sudo /usr/bin/install-wifi -h
sudo /usr/bin/install-wifi
```

After reboot the error should have been gone.

7.5 Write Protect SD Card

Finaly, after finishing the configuration, the SD card is write protected with an overlay file system. Write protection can be enabled with:

```
sudo raspi-config
"7 Advanced Options"
"AB Overlay FS"
"Would you like the overlay file system to be enabled? " => yes
"Enable overlay file system on boot partition" => yes
```

Now the system is protected and withstands even sporadic power losses. All changes to the file system during runtime are lost at reboot or power cycle, because they are stored in the RAM overlay section.

CHAPTER 8

Development Information

8.1 Makefile

The important command lines for development are collected in the *Makefile*.

8.2 Docker Environment

The rsensor-Docker-Container contains the following components:

- **Mysql Server**
    - the database is initialized with the script `test/init_db.sql`
- sqlite3
- Python
- the latest rsensor package (todo: really needed?)
- (todo: maybe add mosquitto)

The Container is created with the command:

```
cd test
docker build -t rsensor .
```

The container is started with:

```
cd test
docker run --rm --name rsensordb --hostname rsensordb -v $(pwd):/test -p 5555:5000 -p␣
↪3306:3306 -t -i rsensor
```

Options:

–rm delete container at exit

–name .... set container name

–hostname .... set hostname of the container

-v ... map local directory as volume

-p ... map ports to host system

-t ???

-i ???

rsensor name of the container (see docker build)

At the docker-prompt you can testwise access the database:

```
mysql -u root
```

The script `prepare_mysql.sql` creates the user rsensor and the database rsensor:

```
mysql -u root < prepare_mysql.sql
```

8.3 Using the MySQL Docker DB from extern

Acess the container via its IP address with:

```
mysql -u rsensor -h 172.17.0.1  rsensor -p
```

The password is “rsensor” (see prepare_mysql.sql)

Todo: check how to add docker-container to DNS automatically

CHAPTER 9

License

```
The contents of uracoli-rsensor is licensed with a Modified BSD License.

All of this is supposed to be Free Software, Open Source, DFSG-free,
GPL-compatible, and OK to use in both free and proprietary applications.

See the license information in the individual source files for details.

Additions and corrections to this file are welcome.

------------------------------------------------------------------------

Portions of µracoli-rsensor are Copyright (c) 2014 - 2020
  Axel Wachtler,
  All rights reserved.

Portions of µracoli documentation are Copyright (c) 2014-2020
  Axel Wachtler,
  All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions
  are met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
  * Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
  * Neither the name of the authors nor the names of its contributors
    may be used to endorse or promote products derived from this software
    without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

(continues on next page)

(continued from previous page)

```
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.

------------------------------------------------------------------------
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search