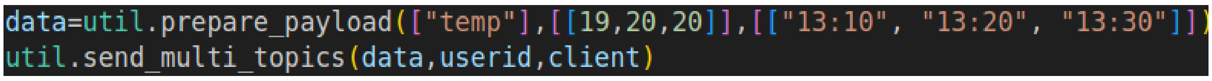
Usage outline

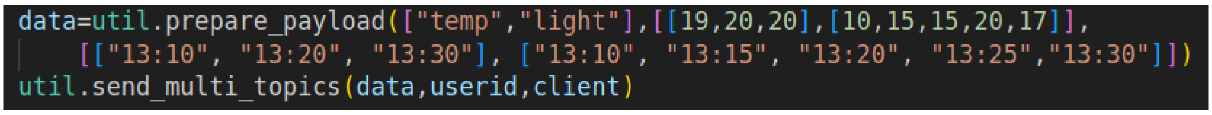
Git repository:
https://github.com/andreascas/aau_ubiss
When you start it will be necessary to execute “git pull” in the folder /home/pi/mqtt_client.
Required for a small update - This may in the future be suggested again.

The program **publisher.py** is deployed with an MQTT client which publishes data to a subscribed user. This enables transmission of data via the MQTT protocol. This is a typically used protocol for IoT communication due to low overhead and good scalability with multiple publishers and subscribers.
Packets sent are retrieved by a subscriber, which notes the time of reception, and adds the sensor data to a database. This is done externally on a different server.
The database maintains data entries for the different sensor values and timestamps so it later can be retrieved and analyzed. A published entry could be as below.



```
data=util.prepare_payload(["temp"],[[19,20,20]],[["13:10", "13:20", "13:30"]])
util.send_multi_topics(data,userid,client)
```

The first command prepares the datatype (temp for temperature), with the 3 entries of the temperature measured at 3 different times. This will then be sent to the MQTT subscriber via the second command, which wraps it up and publishes it. This in particular enables wrapping multiple sensor reading up at once if desired, or single ones i.e.



```
data=util.prepare_payload(["temp","light"],[[19,20,20],[10,15,15,20,17]],
    [["13:10", "13:20", "13:30"], ["13:10", "13:15", "13:20", "13:25","13:30"]])
util.send_multi_topics(data,userid,client)
```

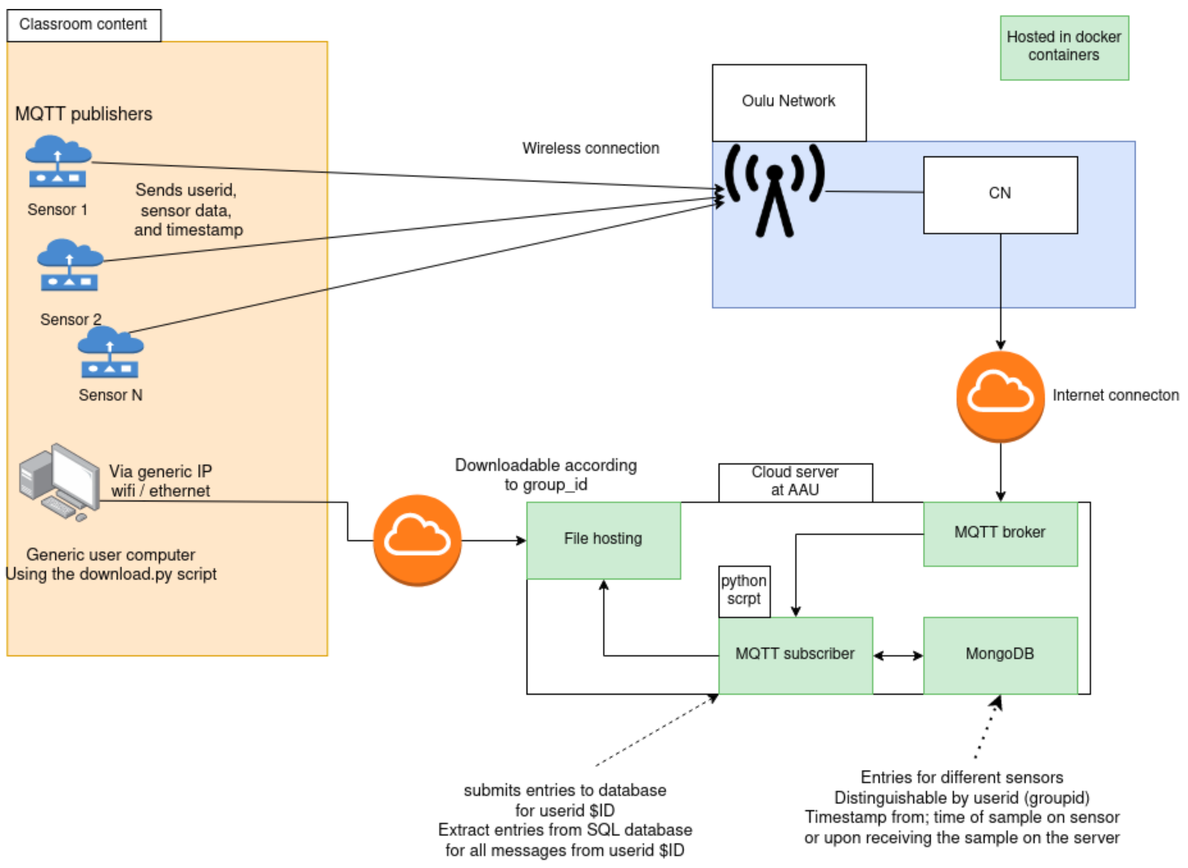
is also a possible call, which would prepare 2 different sensor data types at once. Thus reducing overhead of transmitting multiple times.

Relevant to consider in this context is that the timestamps, the third entry in the **prepare_payload(topic,sensor_data,timestamps)** function, must be of equal size to the amount of topics (**temp and light in this case**). The individual size of lists for each of these can be either a single timestamp, or one for each submitted sample point. - example **publisher_single_ts.py** will show this
If a single timestamp is sent for the sensor, it will be added to the database, where all samples will have the exact same timestamp.
This could be used if a deterministic measurement is used, where each 10th and 5th minute the sensors records a measurement. The result would be a reduced amount of transmitted bits, where the time of measurement could be extrapolated in post-processing..

Each group will be assigned an ID for their group, which must be written into the **util.py file overwriting the “userid=guest” to userid=<groupid>**
It will be used when transmitting sensor data to keep track of what the different groups have sent and when.

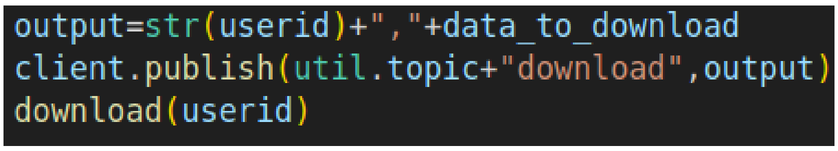
Architecture

Architecturally it is defined as the figure below, te sensor will upload data (with publisher.py), to the access point, which is sent via a CN to the public internet. This is received on a cloud server at Aalborg University (AAU), which hosts an MQTT broker to receive the messages, a subscriber executing a python script to submit data to a database.


Classroom content
MQTT publishers
Sensor 1
Sends userid, sensor data, and timestamp
Sensor 2
Sensor N
Via generic IP wifi / ethernet
Generic user computer Using the download.py script
Wireless connection
Oulu Network
CN
Hosted in docker containers
Internet connecton
Downloadable according to group_id
Cloud server at AAU
File hosting
MQTT broker
python scrpt
MQTT subscriber
MongoDB
submits entries to database for userid $ID Extract entries from SQL database for all messages from userid $ID
Entries for different sensors Distinguishable by userid (groupid) Timestamp from; time of sample on sensor or upon receiving the sample on the server

Downloading data

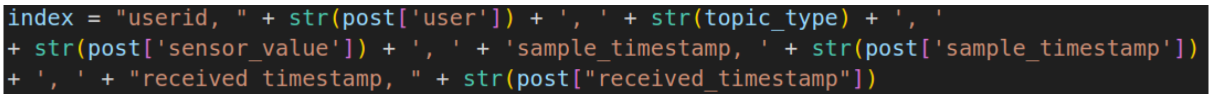
To retrieves data from the database, the **download.py** function must be called. It prepares the data from the database, and downloads a file with all the sample points given the group name, and the topic requested.



```
output=str(userid)+","+data_to_download
client.publish(util.topic+"download",output)
download(userid)
```

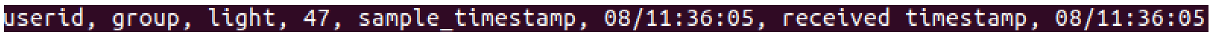
In the example, the **“data_to_download”** variable is the topic desired. This could be **“all”** to retrieve all data sent by the userid. It could also be a singular topic name e.g. “temp”. Which would be retrieve all relevant data points for the sensor values tagged as “temp” for the give userid.
Relevant is that the **“data_to_download”** is either “all”, or a topic previously used. Only one can be prepared at a time.

When executed, each data entry will be retrieved and prepared as;



```
index = "userid, " + str(post['user']) + ', ' + str(topic_type) + ', '
+ str(post['sensor_value']) + ', ' + 'sample_timestamp, ' + str(post['sample_timestamp'])
+ ', ' + "received_timestamp, " + str(post["received_timestamp"])
```

This would when received look like;



```
userid, group, light, 47, sample_timestamp, 08/11:36:05, received_timestamp, 08/11:36:05
```

When received on the computer the output of the entry would be:
The ID of the group (group in this case), the "topic" or sensor (light), its value (47). Then the timestamp of the sample (day,hour,minute,second) and when it correspondingly was received.
Each entry is written to a file called <userid>.csv, in the folder the python program is called.

The downloaded file will contain an entry on the size of the messages submitted. The submitted message length displayed is in Bytes, and will be attached to all database entries from the same MQTT message.
To uniquely identify the entries submitted in an MQTT message, look for the entries with same "received timestamp", these will all share the same Message length.
The message length will only be considered ones for any scoring mechanism.

Connecting to the Raspberry PI

The PI is configured with a static IP over its ethernet connection.
It's ip is configured to be 192.168.10.11

Connecting to it can be done via SSH if you are on the same network.

Hence the first step is setting up a static IP to connect to the raspberry pi

Windows
https://nordvpn.com/blog/how-to-set-up-static-ip-address/

Mac
https://www.macinstruct.com/tutorials/how-to-set-a-static-ip-address-on-a-mac/

Linux (ubuntu example)
https://www.freecodecamp.org/news/setting-a-static-ip-in-ubuntu-linux-ip-address-tutorial/

Set the IP to something in the same subnet e.g. 192.168.10.1

The password will be "**ubiss**"

In the folder mqtt-client (/home/pi/mqtt-client), you will find the implementation which contains an MQTT publisher, **publisher.py**, which is used to publish the sensor data.
APIs for the various sensors are also implemented and can be called with generic "get_measurement" calls as can be observed.
It will return a timestamp and a sample.
These can then be sent to the preconfigured ip, which hosts the MQTT broker and will admit it to a database.

Examples

As an initial point of reference a few examples are presented for how the logical operations could occur. We present **4 options** in this context, where the deeper implementation can be observed in the example folders. **publisher.py** shows an example of transmitting data points every 5th second. This is done for 2 sensors at once.

In **publisher_max.py** it publishes only the max value within a minute.
This could be relevant to reduce the amount of data transmitted, and only transfer the maximum sensor value observed.Thus the event of light turning on/off could be done as a result of max/min of of the samples acquired. This would reduce redundancy of data transmitted.

In **publisher_single_ts.py** we operate within a while loop, which sleeps 30 seconds between measurements. Every 10 seconds a sample is made. Only a single timestamp is transmitted at a time for each of the sensors. In this case, the first timestamp of the samples. From this we can acquire when the other samples occurred due to the deterministic behavior.
Effectively this reduces redundant information, by utilizing knowledge of when data transmissions occur.

In **publisher_threshold.py**, we similarly measure every 30 seconds, these are added to a buffer which contains the last 5 samples. Upon reaching the threshold, the data is published with the past 5 samples. Afterwards we change the threshold (assuming we do not wait for an event such as the light turning off.), and first submit again when this applies.
Here we apply an asynchronous transmission when an event occurs.

In the “example_results” the downloaded file’s content can be seen for each of the examples. Named as e.g. “max.txt”, “threshold.txt” etc. These only contain dummy data for the purpose of showing an example of how data can be transferred.

Relevant to note

The server that you connect to should be "130.225.57.224". This should be configured in the util.py file. It is possible it is set it with the “server” variable
The group id must be set to be unique for each group. When downloading or uploading data it will be saved using the id as a key to retrieve the data in the future.