Engineering Department

Master's Degree Artificial Intelligence and Data Engineering

Università degli studi di Pisa
Accademic year - 2021-2022

Hydro Food

An hydroponic greenhouse fully automatized

STUDENTS:
Marco Bellia, Marco Ralli

Internet Of Things

Contents

1 Introduction

Current estimates show that nearly 690 million people are hungry, or 8.9 percent of the world population, the world is not on track to achieve Zero Hunger by 2030. If recent trends continue, the number of people affected by hunger would surpass 840 million by 2030 - United Nation.
Greenhouse gas emissions from factory farms account for 17% of total EU emissions, more than those of all cars and vans on the road combined. - greenpeace.org.

Life expectation in developed countries has increased in the last century, but leading to bad eating habits. Humanity needs a better food culture and a healthy and sustainable diet. Hydroponic technology is the best solution to produce fresh vegetables in a sustainable way.

The goal of this project is to create a smart solution for the management of a hydroponic greenhouse, automating frequent procedures through IoT technology. Our application must be able to independently sensor and actuators in a greenhouse context.

We design the core components of an hydroponic system and implemented a proof of concept of a real world application. We use sensor for air humidity, temperature level, and Co2 and actuators for air and watering managment.

The application also leaves a high degree of freedom to the owner of the sauna, who can set the thresholds as he wishes and can take advantage of the application to apply chromotherapy, setting the color of the light among those available.

The project code is available at the following link: Github Repo

2 Application Logic

The greenhouse is made up of lane. On each lane there are crops and a tank from which water for irrigation is recovered.

The temperature, humidity and co2 sensors constantly detect the values inside the greenhouse and send them to a sensor that makes an average. This sensor sends the data to the border router and then to the controller which with an application logic decides whether or not to open irrigation on the lanes.

When irrigation starts, the valve that manages a lane goes into a "watering" state, decreasing the level of water in the tanks.

The lanes all start together when an irrigation command arrives and, based on the irrigation time, they control the level of water in the tank. If the level, which is the same for all the lanes, is not high enough, the tanks are recharged at the same time to bring the level to 100

3 Architecture

The system architecture is as shown in the following image (Figure 1)

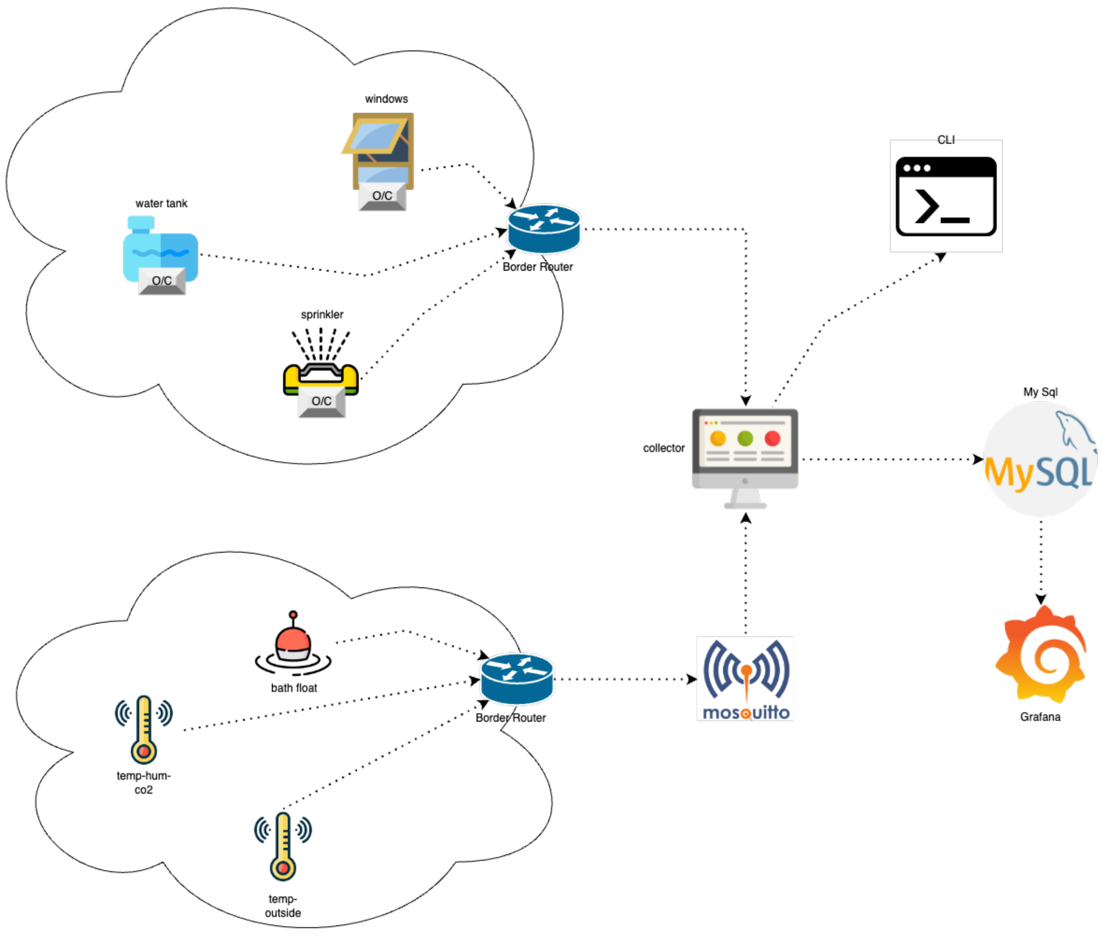


Figure 1: System architecture

As can be seen, the system is composed of two different networks of IoT devices: one network is composed of IoT devices that use MQTT to retrive data, while the other network uses CoAP to communicate directly with the actuators.

Both MQTT and Coap network is deployed using real sensors nRF52840 Dongle. Networks are connected to a border router that allows them external access. The MQTT broker is deployed on the machine.

As will be explained in more detail later, there are three sensors (temperature-humidity-Co2, Outside Temperature and Bath Float) and three actuators (Windows, Water Tank and Sprinkler).

The collector collects data from both MQTT and CoAP sensors and stores them in a MySQL database. The collected data can then be visualized through a web-based interface developed using Grafana. A complex control logic is executed on the collector in order to enable or disable the actuators and modify the external environment based on the data that has been collected and the desired intervals from the sensors that are engaged on data collection. The collector also exposes a simple command line interface through which the user can retrieve the last measured values and change the ranges within which the different physical quantities must lie.

In the next few paragraphs, we will go on to explain the different components of the system in more detail.

3.1 CoAP Network

COAP network is deploy on the sensors that expose some type of resources and with which the application can directly communicate. COAP uses patterns that are similar to the HTTP protocol, so for this reason, the overlay system composed by the COAP sensors represent accesible resources.

3.1.1 Valve Actuator

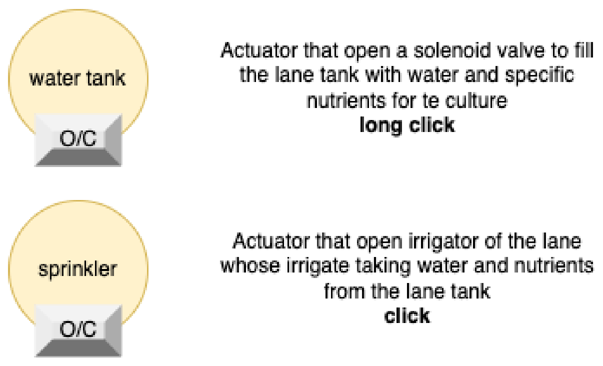
The valve Actuator is used to control the Water Tanks and the Sprinklers. Based on the value that are sense, the application logic communicate to the valve actuators to start the irrigation or the refill of the water tank. The logic behind these valves is described in the following figure (figure 2).

The figure describe the different state in which the valves could be. In each lane there is one valve that is in charge of handle the watering and the refill of the water tank. If the watering is started the refill of the water tank cannot start and vice versa. The figure explain also the activation of the led based on the different status.

3.1.2 Window Actuator

The Window actuator is used to control the opening and closing of the windows. Based on the value that are sense, the application logic communicate to the window actuator to open or close the windows, for example in case of bad quality air, Co2 or humidity. The logic behind these valves is described in the following figure (figure 2).

The figure describe the different state in which the valves could be. The figure explain also the activation of the led based on the different status.


water tank
O/C
Actuator that open a solenoid valve to fill the lane tank with water and specific nutrients for te culture
long click
sprinkler
O/C
Actuator that open irrigator of the lane whose irrigate taking water and nutrients from the lane tank
click

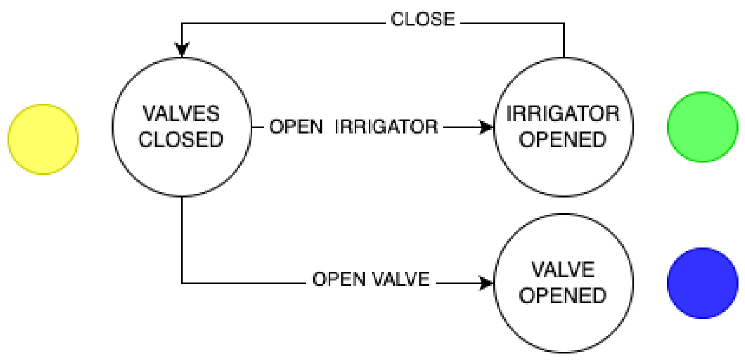


Figure 2: valves actuator diagram

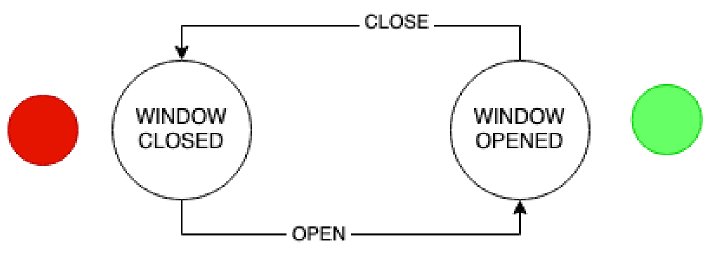


Figure 3: window actuator diagram

3.2 MQTT Network

The main aspect of the hydroponic greenhouse is the collection of the values of temperature, humidity and co2. Together with these values the hydroponic greenhouse catch the level of the water tank on each lane.

The environment has also a ventilation system and to operate with the outside the application is able to collect also values of the outside temperature.

MQTT network is deploy on these type of sensors and the main duty is to collect and generate, using a simulation logic, all the values that are used by COAP network.

3.2.1 Bath Float

The bath float sensors sense the level of the water within the tank. The level of the water decreases when the watering is going.

The code flashed on the sensors simulate the different values of the water, based on the application logic. The values at the start is set to 80 and when the watering start, the logic on the sensor related to the correspondent lane, the level decreases.

3.2.2 Temperature, Humidity and CO2

The temp-hum-Co2 sensors sense the value of temperature, humidity and Co2 within the greenhouse. The code flashed on the sensors simulate the on going changes of these values.

Together with the different status of the actuator the code generates compatible values. If the watering is started, value of temperature decreases and humidity increases.

At the same time the sensor implement the switch between day and night to create a complete simulation of the environment.

3.2.3 Outside Temperature

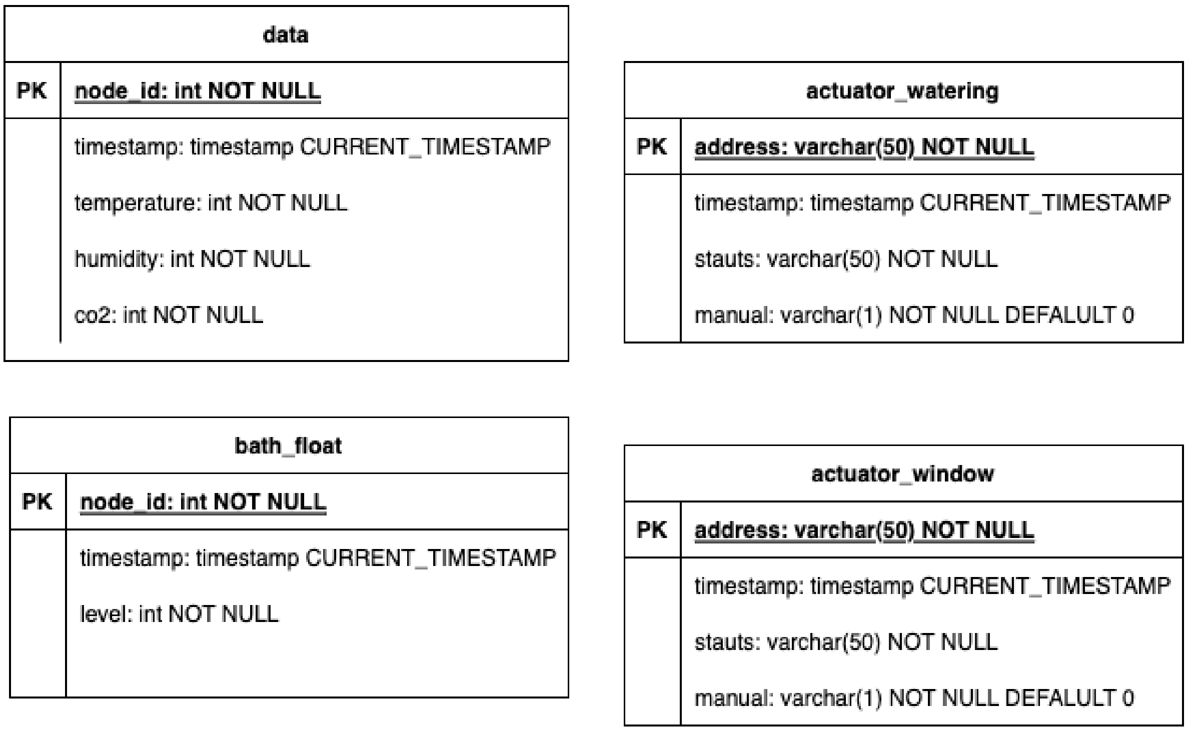
The outside temperature sensor sense the external temperature. The code flashed on the sensor simulate the on going changes of this values. This sensor retrieve only this value and not the co2, we suppose that the value of the co2 outside is always better than the values inside.

3.3 Database

It is essential to store the data collected with the sensors, also to be able to analyze them through Grafana. The database ("iot") is particularly simple, we have defined different tables: data(temperature humidity Co2) bath float (level of the water within the water tank) actuator watering (status of the valves for sprinkler and water tank) actuator window (status of the windows)

There are no dependencies between the tables.

For both humidity and temperature, Co2, bath float and sprinkler we can have multiple devices, so in the tables it is necessary to enter the identifier of the node from which we have received the sample. "Data" this table stores the values of within the greenhouse. "Level" this table stores the current level of the bath float. "Actuator_watering" this table stores the current values of the watering valves. The "manual" column is used to notify the button hold. "Actuator_window" this table stores the current values of the windows valves. The "manual" column is used to notify the button hold.



	data
PK	node_id: int NOT NULL
	timestamp: timestamp CURRENT_TIMESTAMP
	temperature: int NOT NULL
	humidity: int NOT NULL
	co2: int NOT NULL

	actuator_watering
PK	address: varchar(50) NOT NULL
	timestamp: timestamp CURRENT_TIMESTAMP
	stauts: varchar(50) NOT NULL
	manual: varchar(1) NOT NULL DEFALULT 0

	bath_float
PK	node_id: int NOT NULL
	timestamp: timestamp CURRENT_TIMESTAMP
	level: int NOT NULL

	actuator_window
PK	address: varchar(50) NOT NULL
	timestamp: timestamp CURRENT_TIMESTAMP
	stauts: varchar(50) NOT NULL
	manual: varchar(1) NOT NULL DEFALULT 0

Figure 4: database

3.4 Data Encoding

As data encoding, we decided to use JSON, all the data generated by the sensors are transmitted to the collector as JSON objects. This choice is due to the fact that sensors have limited resources, and XML has a too complex structure for our needs. JSON is more flexible and less verbose, resulting in less overhead.

4 Deployment and Execution

4.1 Collector

The collector is the main part of the application.

The logic implemented by the collector is used mainly to collect data from the sensors and to send or to receive information from the sensors that implemented actuator function. The collector exposes a simple CLI that permits to the users to control the whole system. This interface is used to set parameters that the users decides based on the type of the farming.

Once the user has set the parameters than the system is completely autonomous. The collector is fully implemented in python using the CoAPThon3 and Phao library. The application is composed by two thread that are used to handle the COAP and MQTT network.

4.2 CLI

The functions made available to the user through the CLI are the following:

- **!List of commands** show the list of all possible commands.

- **!Info commands** show what commands do.
- **!Change val** command to change value of humidity, temperature and co2
- **!log** command to look for the log of the sensors
- **!bath** command to look for the level of water within the tank.
- **!sim** command that show the mode of operation of the greenhouse.
- **!Exit** close the application but don't stop the business.

4.3 CoAP Network implementation

The server is started on the main by the thread that handle the COAP network. After the initialization of the COAP server, the application registers itself to the resources that manage the actuators.

- **CLASS ResExample** this class is used to implement the get the addresses of the actuators and to observe the resources that they expose.
- **CLASS sendpost** this class menages the post method to communicate directly with the resources exposes by the actuator.
- **CLASS obs_sensors** this class menages the observation of the resources. The constructor of the class is called on the method **render_GET (resExample clss)**. Within the constructor the application creates a connection with the database and some fields are set to handle the communication with the sensors. The **observer** method than implement all the logic that are behind the observation of the resources. If the a message is sent by the sensor, for example "watering start", the application is able to communicate with the MQTT network to notify the event.

4.4 MQTT Network implementation

The client is started on the main by the thread that handle the MQTT network both for the bath float and data.

- **FILE mqtt_collector_bath_float.py** the class in this file is used to menage the sensors the are engaged with the water tank level. The constructor is called on the **Main** when the thread is instantiated. On the constructor is created the connection with the database and with the broker. There is also the call to the **mqtt.Client()** to start the client on the application and the call to the **on_connect** and **on_message**.
  - **on_connect** This method subscribe the client to the topic of interest.
  - **on_message** This method handle the messages that arrived to the client. The method retrieve the fields of the JSON message and it does a write on the database. There is a control to check if the charge of the tank is needed.
  - **checActuatorLevel** This method check if the last value of the level is good. If it isn't the logic call the method to charge the tank.
  - **openCharge** This method is used to start the charge of the tank. There are some logic to control if the actuators is able to start the charge.

  - **closeCharge** This method is used to stop the charge of the tank. There are some logic to control if the actuators is able to start the charging.
  - **communicateToSensors** This method is used to send a post to the correspondent actuator to change the status and so to start or stop the charge.
  - **executeLastState** This method is used to look for the last state of the correspondent actuator.

- **FILE mqtt_collector_values.py**the class in this file is used to menage the sensors the are engaged with the water tank level. The constructor is called on the **Main** when the thread is instantiated. On the constructor is created the connection with the database and with the broker. There is also the call to the **mqtt.Client()** to start the client on the application and the call to the **on_connect** and **on_message**. The parameters that the users, (tempMax/min, humMax/min, co2MAx/min)are passed to the constructor.
  - **on_connect** This method subscribe the client to the topic of interest.
  - **on_message** This method handle the messages that arrived to the client. The method retrieve the fields of the JSON message and it does a write on the database. There is a control to check if the charge of the tank is needed.
  - **checActuatorWatering** This method check if the last values reported are still good based on the value that the user has set.
  - **startWatering** This method is used to start the watering. There is some logic to control if the actuators is able to start the watering.
  - **stopWatering** This method is used to stop the watering. There is some logic to control if the actuators is able to start the watering.
  - **shouldOpenWatering** This method check if the values are still good and if watering is needed.
  - **checActuatorWindow** This method check if the last values reported are still good based on the value that the user has set.
  - **openWindow** This method is used to open the window. There is some logic to control if the actuators is able to open the window.
  - **closeWindow** This method is used to close the window. There is some logic to control if the actuators is able to close the window.
  - **communicateToSensors** This method is used to send a post to the correspondent actuator to change the status and so to start or stop the charge.
  - **executeLastState** This method is used to look for the last state of the correspondent actuator.

- **FILE addresses.py** the class in this file is used to store all the address of the node that join the network. The addresses stored are used by the Collector to communicate with all the node

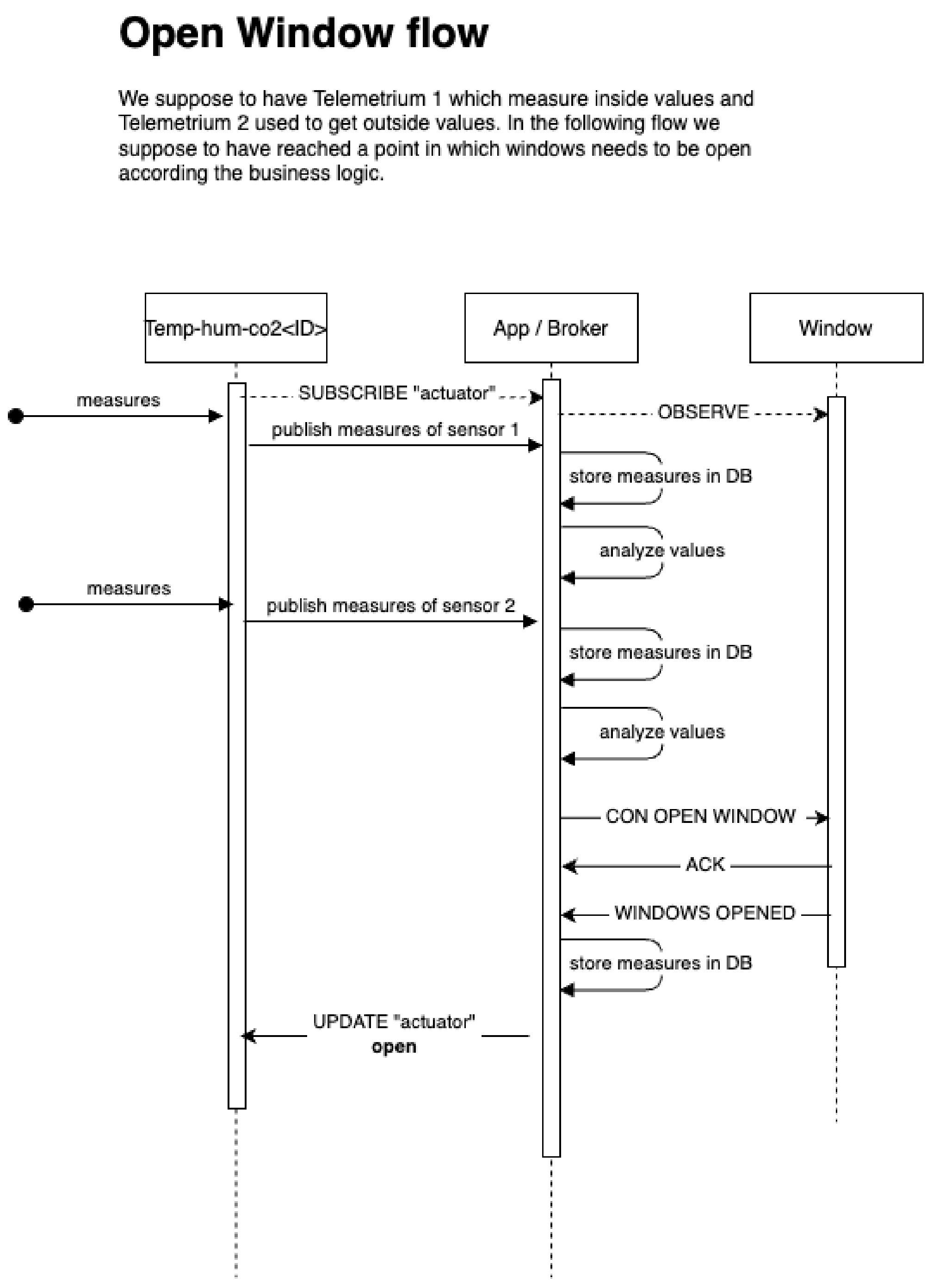
- **FILE obs_sensor.py** the class in this file waits for the connection of the COAP sensor and call the observation on them. There are two variables that are **valves** and **window** that are set to "1" when the connection is established.

- **FILE send_post.py** the class in this file is used by the controller to communicate some information to the COAP sensors when the application logic requires it. The **getClient** method is used to retrieve the client that want to communicate with the sensors, in this case the Controller.
- **FILE globalStatus.py** the class in this file is used to store the current state of the greenhouse that is used by the **sim** command to show the on going simulation

5 Flow of Execution

The flow of the execution is described on the following diagrams.

5.1 Window Flow

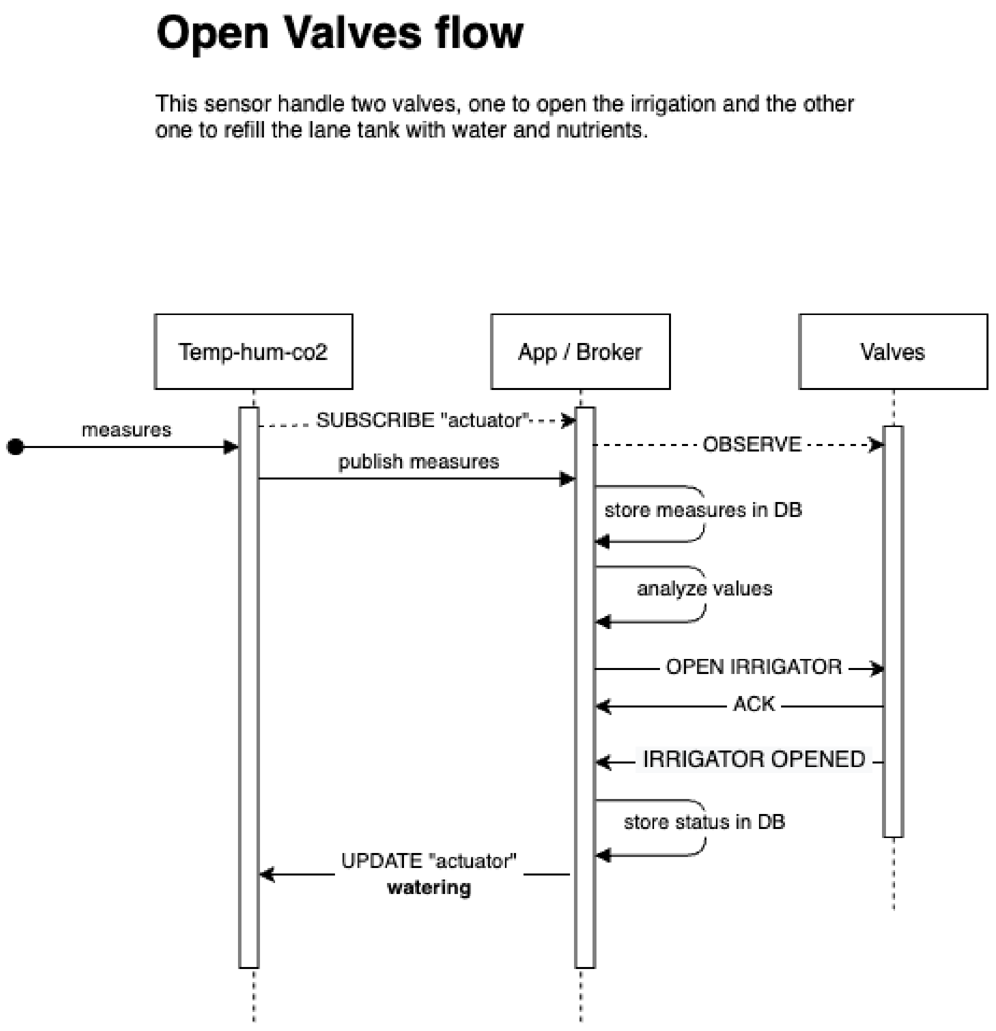


Open Window flow

We suppose to have Telemetrium 1 which measure inside values and Telemetrium 2 used to get outside values. In the following flow we suppose to have reached a point in which windows needs to be open according the business logic.

Figure 5: window flow

5.2 Valves Flow



Open Valves flow

This sensor handle two valves, one to open the irrigation and the other one to refill the lane tank with water and nutrients.

Figure 6: bath float flow

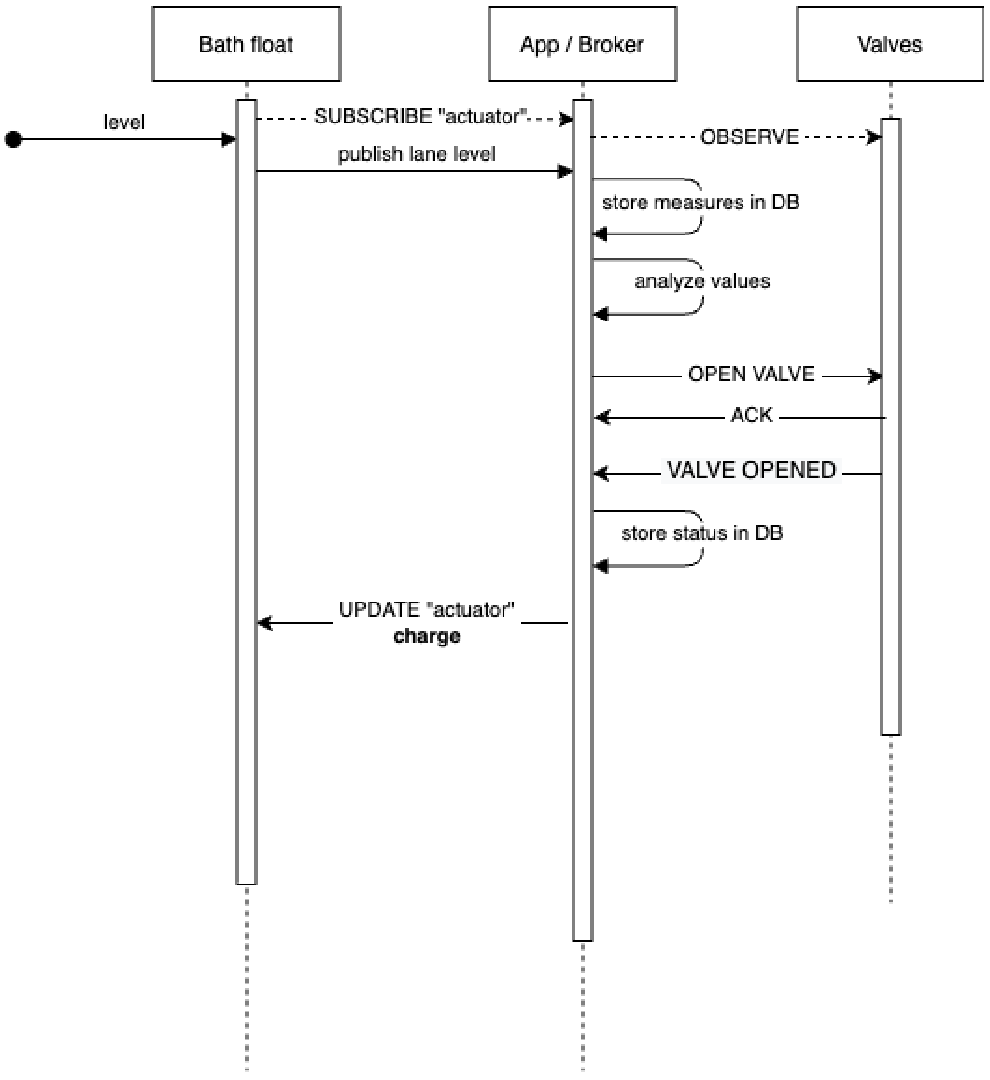


Figure 7: valves flow

6 GRAFANA

Grafana dashboard is used to compose some analytics based on the value that are store on the database.

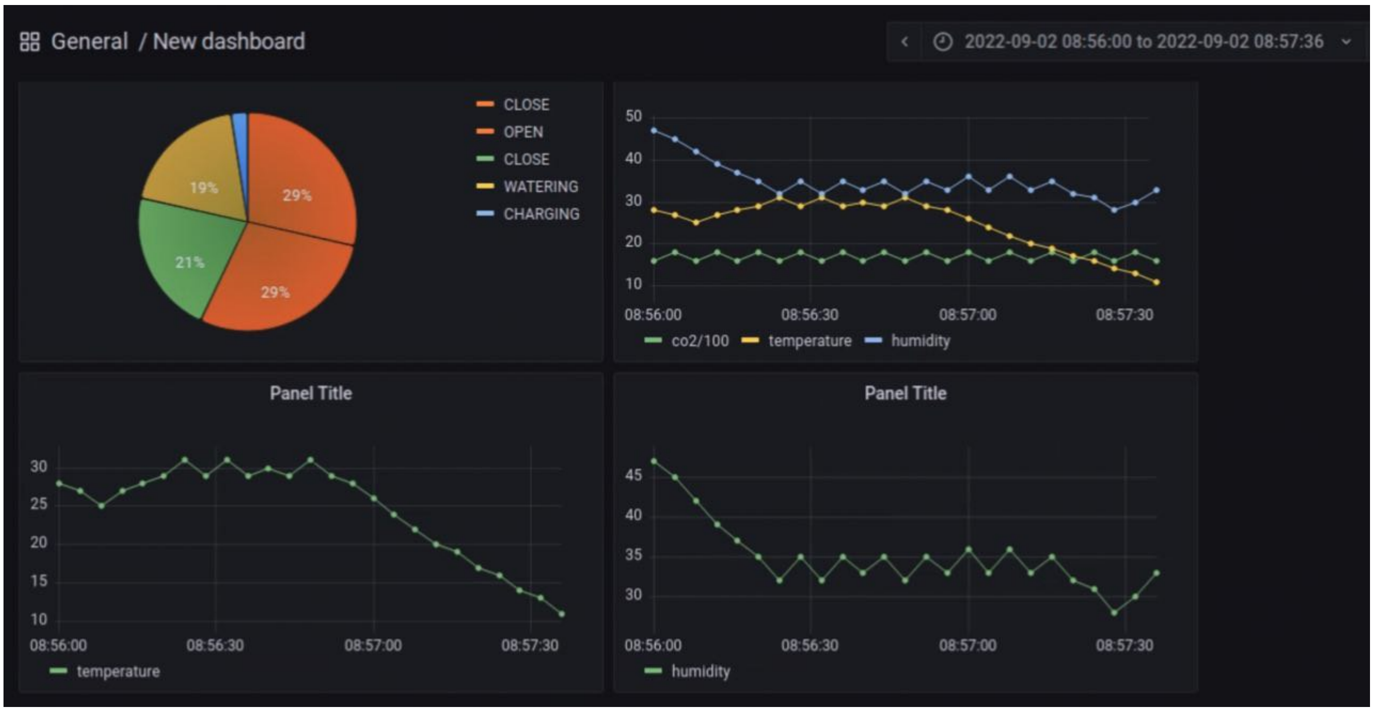


Figure 8: grafana