



# UNIVERSITÀ DI PISA

---

882II - INTERNET OF THINGS

2022-2023 ACADEMIC YEAR

## SMART GREEN HOUSE CONTROL SYSTEM

Berfin Yüksel

Stefano Bianchettin

## TABLE OF CONTENTS

1. INTRODUCTION.....	3
2. USE CASE .....	3
3. ARCHITECTURE.....	4
a. MQTT NETWORK.....	5
b. COAP NETWORK .....	5
d. DATABASE .....	9
e. DATA ENCODING.....	9
f. GRAFANA .....	10
4. DEPLOYMENT.....	12

## 1. INTRODUCTION

Greenhouses are made for creating an optimized environment for plants. Each plant needs specific conditions that need to be tracked well to sustain an optimized environment. However, controlling these criteria manually is time consuming.

The idea behind the Green House Control System is creating a smart environment for a plant that needs specific conditions of some factors through IOT technology.

In our system we control 3 factors which are *Temperature, Light Intensity and Soil Humidity*. Therefore, our system detects the latest values that the greenhouse has and decides if these values are beneath or above of the upper boundaries and starts *Ventilation System* for Temperature optimization, *Light System* for Light Intensity optimization and *Watering System* for Soil Humidity optimization. These 3 systems are working separately from each other according to need. Thus, the user can reach automatic control of setting in the greenhouse.

Temperature is a critical factor in plant growth, as it influences various physiological processes and metabolic activities within the plants. Controlling the temperature in a smart greenhouse allows for the following benefits: Prevention of stress and diseases, extended growing seasons and enhanced photosynthesis.

The level of soil moisture directly impacts a plant's ability to take up water and nutrients from the soil. Maintaining appropriate soil moisture levels is crucial for several reasons: nutrient uptake, plant hydration and root health.

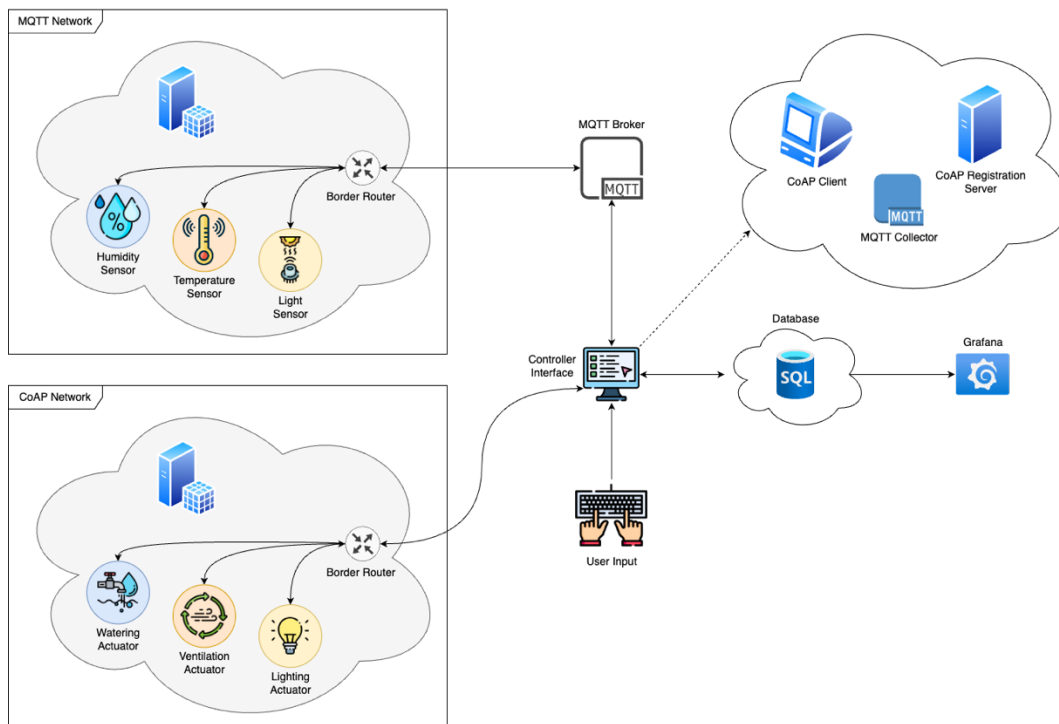
Light is the primary source of energy for photosynthesis and, therefore, an indispensable factor in plant growth. Smart greenhouse systems must consider light control for several reasons, such as energy efficiency: smart greenhouse systems can optimize the use of artificial lighting by providing light only when needed, reducing energy costs, and minimizing environmental impact.

## 2. USE CASE

1. The user will be able to use the interface to set the upper boundary and lower boundary for each parameter, such as humidity, temperature, and light intensity.
2. After this, the system will start collecting sample data.
3. The system takes an average of the last 30 seconds for humidity and temperature samples of data collected. For light intensity it considers the last sample.
4. Ventilation System:
  - a. If the average temperature is higher than the upper boundary, the ventilation system will run at decreased mode and the led of the device will be set to **green** color automatically.

- b. Likewise, if the average temperature is lower than the lower boundary, ventilation system will run at increased mode and the led of the device will be set to **green** color automatically.
  - c. if the average temperature is on the threshold range the ventilation system will turn off and the led of the device will be set to **red** color automatically.
  - d. If the device button is pressed the mode changes to INC/DEC.
5. Light System:
  - a. If the last sample for light intensity is below the lower boundary, the lighting system is turned on and the led of the device will be set to **green** color automatically.
  - b. If the last sample for light intensity is higher than the higher boundary, the lighting system is turned off and the led of the device will be set to **red** color automatically.
  - c. If the device button is pressed the mode changes to INC/DEC.
6. Watering System:
  - a. If the average humidity of the soil is below the lower boundary, the watering system is turned on and the led of the device will be set to **green** color automatically.
  - b. If the average humidity of the soil is higher than the upper boundary, the watering system is turned off and the led of the device will be set to **red** color automatically.
  - c. If the device button is pressed the mode changes to INC/DEC.
7. When the system is making detections between the upper and lower boundaries, the system will keep working at its current state at that point to sustain the values.

### 3. ARCHITECTURE



### a. MQTT NETWORK

The MQTT-device connects as a client to an MQTT broker publishing and subscribing to dedicated topics to communicate with the collector.

#### Temperature Sensor:

The temperature device publishes to the topic: "temperature" to notify the collector of the current temperatures inside the greenhouse. The format of the message is {"nodeId": VALUE, "temperature": VALUE }.

It subscribes to the topic: "temperature-command" to receive notification from the ventilation actuator.

#### Light Sensor:

The light device publishes to the topic: "light" to notify the collector of the current intensity of natural light inside the greenhouse. The format of the message is {"node": VALUE, "intensity": VALUE }.

It subscribes to the topic: "light-command" to receive notification from the light actuator.

#### Humidity Sensor:

The humidity device publishes to the topic: "humidity" to notify the collector of the current moisture content in the soil. The format of the message is {"node": VALUE, "humidity": VALUE }.

It subscribes to the topic: "humidity-command" to receive notification from the watering actuator.

### b. COAP NETWORK

Each device in the CoAP Network registers itself to the Collector Server, thus allowing the server to take trace of which devices are active and what kind of signals they are sampling. Once registered, the server creates a dedicated process, for each device, responsible for observing the data and uploading it to the database. Processes control that the data collected by the sensors are within the thresholds, otherwise they have to set the alarm status and send to the relative actuator the command to activate itself until the values do not return within the limits. This closed-loop control logic is fundamental in order to maintain growth plant.

the actuator registers as a client to the cloud application server using the POST method with service url **/registration**. Furthermore, an actuator can be deleted from the database using the DELETE method.

**Ventilation System:** Manages the opening and closing of greenhouse vents to regulate temperature. If the temperature exceeds a predefined threshold, the ventilation system runs in to cool down the greenhouse which is DEC mode. If the temperature falls below the threshold, the ventilation system runs to heat up the greenhouse which is INC mode.

The actuator can be triggered by a PUT request at the following address: **coap://[" + ip + "]/ventilation-system/switch**. The payload of the request must contains "INC", "DEC" or "OFF". The status of the actuator is indicated by green/red LED.

**Light System:** The actuator works with the trigger of the light sensor that detects insufficient or too much natural light which makes light system to sustain an artificial lighting system actuator to supplement the required light levels for plant growth. It works ON/OFF mode. The aim of the actuator is to keep the light intensity between the lower and upper boundary.

The actuator can be triggered by a PUT request at the following address: **coap://[" + ip + "]/light-system/switch**. The payload of the request must contains "ON" or "OFF". The status of the actuator is indicated by green/red LED.

**Watering System:** Controls the irrigation system to provide the required amount of water to the plants. When activated, the actuator opens the water valves to allow water to flow into the irrigation system. It works ON/OFF mode. If the soil moisture level falls below a predefined threshold, indicating dry soil, the control system triggers the Watering System Actuator to start watering the plants. If the soil moisture level rises above predefined threshold, indicating sufficient moisture, the actuator stops the watering process.

The actuator can be triggered by a PUT request at the following address: **coap://[" + ip + "]/watering-system/switch**. The payload of the request must contains "ON" or "OFF". The status of the actuator is indicated by green/red LED.

### c. Code Implementation

Sensors Example Class: *TemperatureDevice.java*

The **lastSamples** field is a list that will hold temperature samples. The **lowerBound** and **upperBound** are floats representing the minimum and maximum acceptable temperature values.

```
public void addSample(TemperatureSample temperatureSample){
    temperatureSample.setTimestamp(new Timestamp(System.currentTimeMillis()));
    lastSamples.add(temperatureSample);

    Iterator<TemperatureSample> iterator = lastSamples.iterator();

    while (iterator.hasNext()) {
        TemperatureSample obj = iterator.next();
        long secondsDifference = java.time.Duration.between(obj.getTimestamp().toInstant(), java.time.Instant.now()).getSeconds();
        if (secondsDifference >= 30) {
            iterator.remove();
        }
    } // remove old samples from the lastSamples list, if it has been done in the last 30sec
    DBDriver.getInstance().insertTemperatureSample(temperatureSample);
}
```

The `addSample()` method is used to add a new temperature sample to the `lastSamples` list. It sets the current timestamp for the sample, adds it to the list, and then iterates over the list to remove any samples that are older than 30 seconds. Finally, it calls the `insertTemperatureSample()` method of the `DBDriver` class to insert the temperature sample into a database.

```

public float getAvgTemperature(){
    return (float) lastSamples.stream()
        .mapToDouble(TemperatureSample::getTemperature)
        .average()
        .orElse(0.0); // provide a default value if the list is empty
}

```

The `getAvgTemperature()` method calculates the average temperature value from all the samples stored in the `lastSamples` list. It maps each sample to its temperature value, calculates the average, and returns it as a float. If the list is empty, it returns a default value of 0.0.

#### Actuator Example Class: *VentilationSystem.java*

The class `VentilationSystem` is defined. It contains two member variables:

*clientVentilationSystemList*: a list of `CoapClient` objects that represent the ventilation system devices.

*state*: a `String` variable that represents the current state of the ventilation system ("OFF", "INC", or "DEC").

```

public static void switchVentilationSystem(String mode){
    if(clientVentilationSystemList.isEmpty()){
        return;
    }

    if (Objects.equals(mode, b:"INC"))
        state = "INC";
    else if (Objects.equals(mode, b:"DEC"))
        state = "DEC";
    else
        state = "OFF";

    String msg = "mode=" + mode;
    for(CoapClient client: clientVentilationSystemList) {
        client.put(new CoapHandler() {
            @Override
            public void onLoad(CoapResponse coapResponse) {
                if (coapResponse != null) {
                    if (!coapResponse.isSuccess())
                        System.out.print(s:"[ERROR]Ventilation system switch: PUT request unsuccessful\n");
                }
            }

            @Override
            public void onError() {
                System.err.print("[ERROR] Ventilation system switch " + client.getURI() + "];");
            }
        }, msg, MediaTypeRegistry.TEXT_PLAIN);
    }
}

```

The `switchVentilationSystem()` method is a static method that switches the ventilation system to a specified mode. It takes a `String` parameter `mode` indicating the desired mode of operation ("INC", "DEC", or any other value representing "OFF"). If the `clientVentilationSystemList` is empty (no devices registered), the method simply returns without performing any action.

Depending on the value of mode, the state variable is updated accordingly. If mode is "INC", the state is set to "INC". If mode is "DEC", the state is set to "DEC". Otherwise, the state is set to "OFF".

The method sends a CoAP (Constrained Application Protocol) PUT request to each registered ventilation system device to switch its mode. It creates a CoapHandler to handle the CoAP response. If the response is not successful, it prints an error message. The payload of the PUT request is the msg string, which contains the mode value. The media type of the payload is set to TEXT\_PLAIN.

#### *MQTTHandler.java*

This class is responsible for handling MQTT communication in the smart greenhouse system.

```
public MQTTHandler() {
    temperatureDevice = new TemperatureDevice();
    lightIntensityDevice = new LightIntensityDevice();
    humidityDevice = new HumidityDevice();

    do {
        try {
            mqttClient = new MqttClient(BROKER, CLIENT_ID);
            System.out.println("Connecting to the broker: " + BROKER);
            mqttClient.setCallback(this);
            connectBroker();
        }
        catch (MqttException me)
        {
            System.out.println(x:"Connection error");
        }
    }while(!mqttClient.isConnected());
}
```

The class initializes an MQTT client `mqttClient` and connects to the MQTT broker specified by the `BROKER` constant. It also creates instances of the `TemperatureDevice`, `HumidityDevice`, and `LightIntensityDevice` classes. The `TemperatureDevice` represents a temperature sensor, the `HumidityDevice` represents a humidity sensor, and the `LightIntensityDevice` represents a light intensity sensor.

```
private void connectBroker () throws MqttException {
    mqttClient.connect();
    mqttClient.subscribe(humidityDevice.HUMIDITY_TOPIC);
    System.out.println("Subscribed to: " + humidityDevice.HUMIDITY_TOPIC);
    mqttClient.subscribe(lightIntensityDevice.LIGHTINTENSITY_TOPIC);
    System.out.println("Subscribed to: " + lightIntensityDevice.LIGHTINTENSITY_TOPIC);
    mqttClient.subscribe(temperatureDevice.TEMPERATURE_TOPIC);
    System.out.println("Subscribed to: " + temperatureDevice.TEMPERATURE_TOPIC);
}
```



The *connectBroker()* method is responsible for connecting the MQTT client to the broker and subscribing to relevant topics. It subscribes to topics related to humidity, light intensity, and temperature measurements.

*messageArrived()* method is an overridden method from the *MqttCallback* interface. It is called when a new MQTT message arrives on a subscribed topic.

*messageArrived()* method processes the incoming messages for temperature, humidity, and light intensity topics.

#### d. DATABASE

It is essential to store the data collected with the sensors, also to be able to analyze them through Grafana. The database (smart\_greenhouse) is particularly simple, we have defined a table for each type of measurement (humidity, temperature, etc.), and there are no dependencies between the tables. For both humidity, temperature and light 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. Also, a table is defined for the CoAP actuators that register to the application.

<table> <tr> <th colspan="2">actuators</th></tr> <tr> <td>PK</td><td><u>IP : VARCHAR NOT NULL</u></td></tr> <tr> <td></td><td>name: VARCHAR DEFAULT NULL</td></tr> <tr> <td></td><td>timestamp: timestamp CURRENT_TIMESTAMP</td></tr> <tr> <td></td><td>state: VARCHAR DEFAULT NULL</td></tr> </table>	actuators		PK	<u>IP : VARCHAR NOT NULL</u>		name: VARCHAR DEFAULT NULL		timestamp: timestamp CURRENT_TIMESTAMP		state: VARCHAR DEFAULT NULL	<table> <tr> <th colspan="2">temperature</th></tr> <tr> <td>PK</td><td><u>ID : int NOT NULL</u></td></tr> <tr> <td></td><td>nodeId: int NOT NULL</td></tr> <tr> <td></td><td>timestamp: timestamp CURRENT_TIMESTAMP</td></tr> <tr> <td></td><td>degrees: int NOT NULL</td></tr> </table>	temperature		PK	<u>ID : int NOT NULL</u>		nodeId: int NOT NULL		timestamp: timestamp CURRENT_TIMESTAMP		degrees: int NOT NULL
actuators																					
PK	<u>IP : VARCHAR NOT NULL</u>																				
	name: VARCHAR DEFAULT NULL																				
	timestamp: timestamp CURRENT_TIMESTAMP																				
	state: VARCHAR DEFAULT NULL																				
temperature																					
PK	<u>ID : int NOT NULL</u>																				
	nodeId: int NOT NULL																				
	timestamp: timestamp CURRENT_TIMESTAMP																				
	degrees: int NOT NULL																				
<table> <tr> <th colspan="2">light</th></tr> <tr> <td>PK</td><td><u>ID : int NOT NULL</u></td></tr> <tr> <td></td><td>nodeId: int NOT NULL</td></tr> <tr> <td></td><td>timestamp: timestamp CURRENT_TIMESTAMP</td></tr> <tr> <td></td><td>level: int NOT NULL</td></tr> </table>	light		PK	<u>ID : int NOT NULL</u>		nodeId: int NOT NULL		timestamp: timestamp CURRENT_TIMESTAMP		level: int NOT NULL	<table> <tr> <th colspan="2">humidity</th></tr> <tr> <td>PK</td><td><u>ID : int NOT NULL</u></td></tr> <tr> <td></td><td>nodeId: int NOT NULL</td></tr> <tr> <td></td><td>timestamp: timestamp CURRENT_TIMESTAMP</td></tr> <tr> <td></td><td>level: int NOT NULL</td></tr> </table>	humidity		PK	<u>ID : int NOT NULL</u>		nodeId: int NOT NULL		timestamp: timestamp CURRENT_TIMESTAMP		level: int NOT NULL
light																					
PK	<u>ID : int NOT NULL</u>																				
	nodeId: int NOT NULL																				
	timestamp: timestamp CURRENT_TIMESTAMP																				
	level: int NOT NULL																				
humidity																					
PK	<u>ID : int NOT NULL</u>																				
	nodeId: int NOT NULL																				
	timestamp: timestamp CURRENT_TIMESTAMP																				
	level: int NOT NULL																				

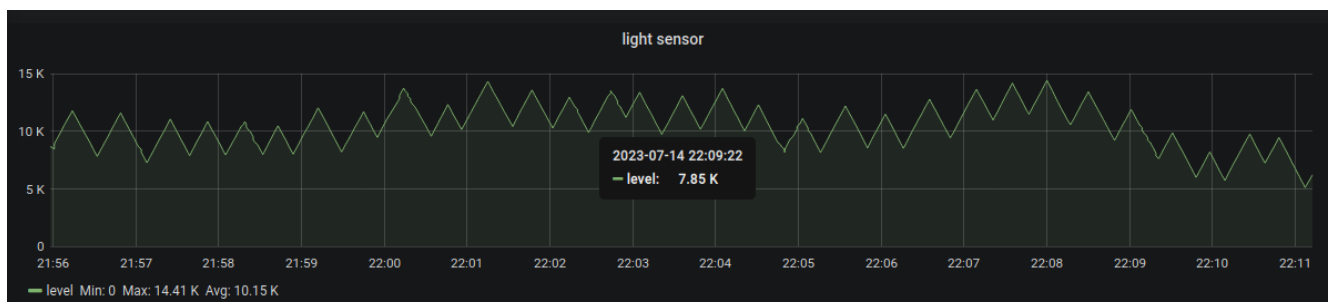
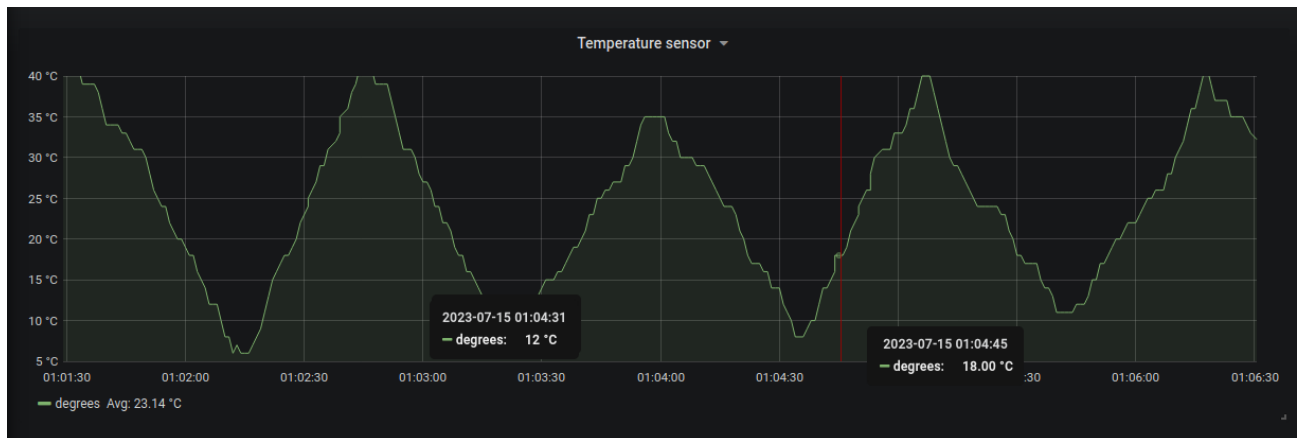
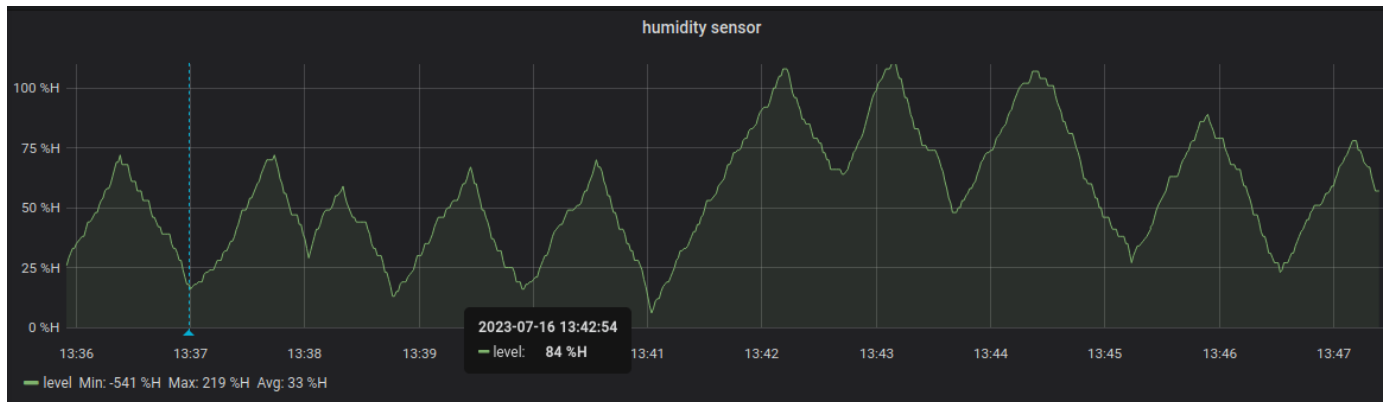
#### e. DATA ENCODING

To facilitate the exchange of measurements in both CoAP and MQTT, we opted to employ JSON messages. This decision was motivated by the need to include multiple fields within the message itself, which is conveniently accomplished using JSON. When the Collector needs to transmit a message to an actuator within the CoAP Network, it will utilize a POST request with the payload containing the parameter in plain text. This approach facilitates parsing within the actuator when utilizing CoAP. Conversely, when the Collector sends a message to an actuator in the MQTT Network, JSON format is employed. All sensor-generated data is transmitted to the collector as JSON objects. This choice is driven

by the limited resources of the sensors, as XML's structure proves overly complex for our requirements. JSON offers greater flexibility and a more concise representation, resulting in reduced overhead.

## f. GRAFANA

We have implemented a dashboard on Grafana in order to be able to monitor in real time the data we store in the database, and therefore to be able to view the trend of the monitored parameters. More precisely, we have created a panel for humidity, one for temperature and one for light. Through this dashboard you can see how the measured values established ranges remain within the established ranges.



### g. COLLECTOR

The collector serves as the central component within our architecture, responsible for receiving data, interacting with the database to store received samples, and making decisions regarding the appropriate actions to be taken. Additionally, it incorporates a Command-Line Interface (CLI) to enhance user management capabilities. To implement the collector, we opted for Java and leveraged the Paho and Californium libraries for added functionality.

#### CLI:

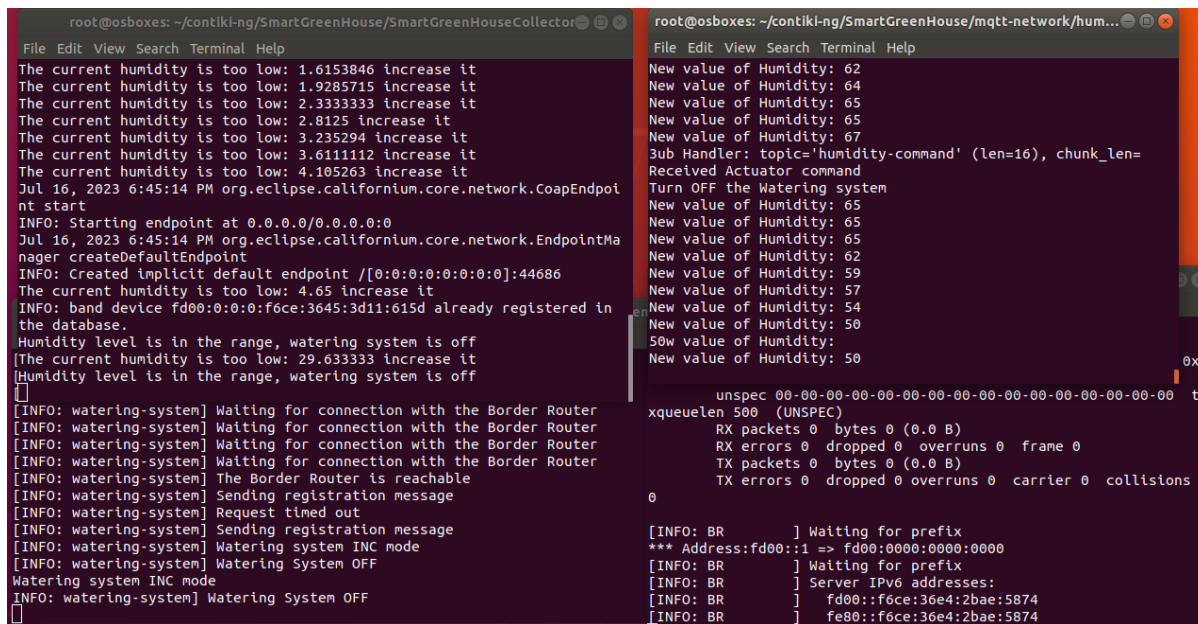
- 1) **!help <command> -->** shows the details of a command
- 2) **!get\_humidity -->** recovers the last humidity measurement
- 3) **!set\_humidity <lower bound> <upper bound> -->** sets the range within which the humidity must stay
- 4) **!start\_watering -->** starts manually the watering system
- 5) **!stop\_watering -->** stops manually the watering system
- 6) **!get\_temperature -->** recovers the last temperature measurement
- 7) **!set\_temperature <lower bound> <upper bound> -->** sets the range within which the temperature must stay
- 8) **!active\_ventilation <mode> -->** starts ventilation system manually in INC or DEC mode
- 9) **!deactivate\_ventilation -->** stops manually the ventilation system
- 10) **!get\_light\_intensity -->** recovers the last light intensity measurement
- 11) **!set\_light\_intensity <lower bound> <upper bound> -->** sets the range within which the light intensity must stay
- 12) **!activate\_lightning -->** starts manually the lightning system
- 13) **!stop\_lightning -->** stops manually the lightning system
- 14) **!exit -->** terminates the program

## 4. DEPLOYMENT

The Smart Green House has been tested with 6 nRF52840 Dongles.

In this example for simplicity only with 3 dongles.

On Top left the Collector Application, on top right the MQTT device plugged on ttyACM2 port, on bottom left the COAP device plugged on ttyACM1 port and on bottom right the RPL border router plugged on ttyACM0 port.



```

root@osboxes: ~/contiki-ng/SmartGreenHouse/SmartGreenHouseCollector
File Edit View Search Terminal Help
The current humidity is too low: 1.6153846 increase it
The current humidity is too low: 1.9285715 increase it
The current humidity is too low: 2.3333333 increase it
The current humidity is too low: 2.8125 increase it
The current humidity is too low: 3.235294 increase it
The current humidity is too low: 3.6111112 increase it
The current humidity is too low: 4.105263 increase it
Jul 16, 2023 6:45:14 PM org.eclipse.californium.core.network.CoapEndpoint
nt start
INFO: Starting endpoint at 0.0.0.0/0.0.0.0
Jul 16, 2023 6:45:14 PM org.eclipse.californium.core.network.EndpointMa
nager createDefaultEndpoint
INFO: Created implicit default endpoint [/0:0:0:0:0:0:0:0]:44686
The current humidity is too low: 4.65 increase it
INFO: band device fd00:0:0:0:f6ce:3645:3d11:615d already registered in
the database.
Humidity level is in the range, watering system is off
[The current humidity is too low: 29.633333 increase it
Humidity level is in the range, watering system is off
[
[INFO: watering-system] Waiting for connection with the Border Router
[INFO: watering-system] Waiting for connection with the Border Router
[INFO: watering-system] Waiting for connection with the Border Router
[INFO: watering-system] Waiting for connection with the Border Router
[INFO: watering-system] The Border Router is reachable
[INFO: watering-system] Sending registration message
[INFO: watering-system] Request timed out
[INFO: watering-system] Sending registration message
[INFO: watering-system] Watering system INC mode
[INFO: watering-system] Watering System OFF
Watering system INC mode
INFO: watering-system] Watering System OFF
[

root@osboxes: ~/contiki-ng/SmartGreenHouse/mqtt-network/hum...
File Edit View Search Terminal Help
New value of Humidity: 62
New value of Humidity: 64
New value of Humidity: 65
New value of Humidity: 65
New value of Humidity: 65
New value of Humidity: 67
3ub Handler: topic='humidity-command' (len=16), chunk_len=
Received Actuator command
Turn OFF the Watering system
New value of Humidity: 65
New value of Humidity: 65
New value of Humidity: 65
New value of Humidity: 62
New value of Humidity: 59
New value of Humidity: 57
New value of Humidity: 54
New value of Humidity: 50
50w value of Humidity:
New value of Humidity: 50
0x
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 t
xqueueLen 500 (UNSPEC)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions
0
[INFO: BR      ] Waiting for prefix
*** Address:fd00::1 => fd00:0000:0000:0000
[INFO: BR      ] Waiting for prefix
[INFO: BR      ] Server IPv6 addresses:
[INFO: BR      ] fd00::f6ce:36e4:2bae:5874
[INFO: BR      ] fe80::f6ce:36e4:2bae:5874

```

