MSc. Music and Acoustic Engineering

Internet of Things - Final Project

Environmental Carbon Dioxide Management System

Author:
Zaccaria Eliseo Carrettoni
[10504151]

Link to the repository: https://github.com/IronZack95/IoT_Network

Contents

1 Introduction

The goal of this project is to develop a sensor system, on hardware, with IoT technologies for monitoring various environmental parameters in my home's cellar, where we produce and store wine for family use every year. The main environmental parameters that I want to observe are humidity, temperature and percentage saturation of the CO_2 level in the air, which is the gas that can most cause asphyxiation problems during the fermentation phase of wine in a closed place. In this project I want both to seek efficient and economical solutions but also to be complementary to the current state of my home computer network. The long-term aim of this project are not limited to the collection of raw data of environmental parameters, but would like to make the whole system automatic and able to act on the environment through specific actuators autonomously, such as turning on the automatic ventilation system, heating or dehumidification of the cellar.

1.1 Network Map

As you can see in the Figure 1, the physical structure of the network is divided between two WiFi routers connected in LAN (20m away from each other); the first one works as a main router connected to the internet and the second one works as an Access Point. The sensors in the cellar are connected via WiFi to the main router (10m away in a straight line). The sensors and actuators communicate over the network using the MQTT protocol to my Homelab server, which will manage the logic and communications of the entire local network, and acts like a database.

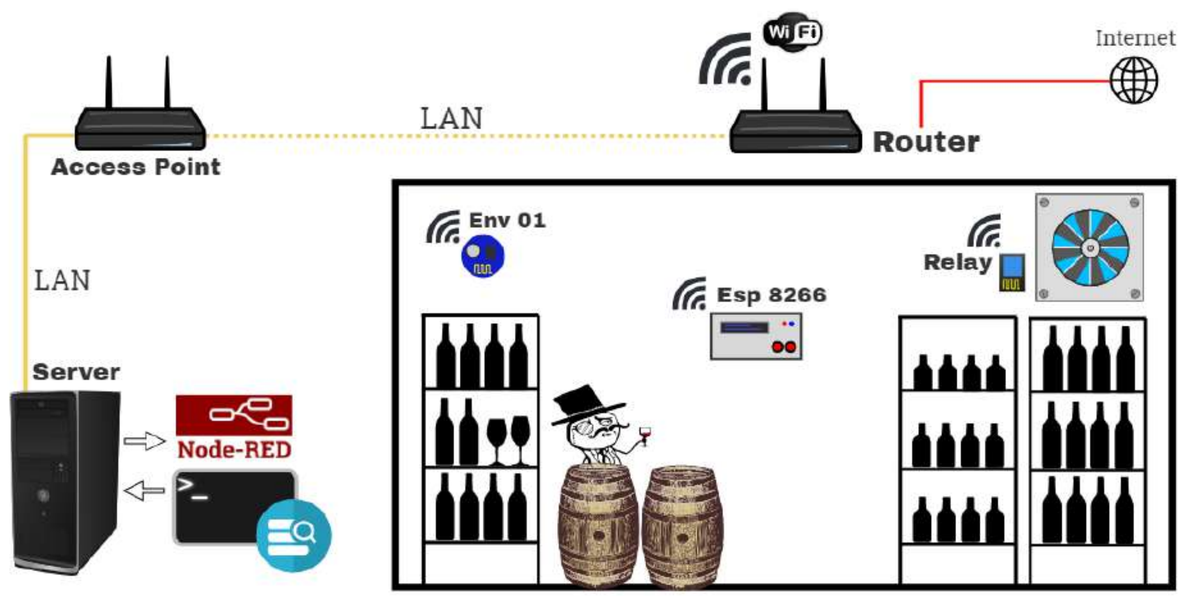

Figure 1: Architecture Schematic

2 Server Side

2.1 Architecture

I have a DELL workstation that I use as a Homelab Server, only from SSH command line, running a Linux Ubuntu Server 20.04 LTS operating system. To meet all the needs of optimizations and compatibility, the best solution was to use Docker container. Dockerization allows programs to run bypassing the SO specific linux distribution and using only the linux kernel, this allows a better isolation of the softwares and a valid alternative to the virtual machine, Figure 2.
First of all, it was necessary to create a network in docker that acts as a bridge between the two main applications placed in the containers, Node-Red and Mosquitto Broker. The bridge allows to collect all the ports shown by the programs in a single virtual environment and show them outside of docker in the network interface. In particular, 1883 and 9001 for MQTT messages, 1880 for Node-Red and 1880/ui for the dashboard. It was also necessary to enable port forwarding from the main router and assign static IP to the server (my case 192.168.1.100).

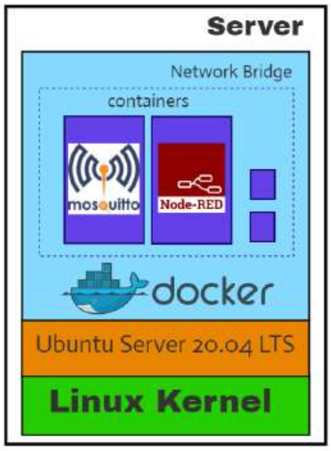

Figure 2: Server Graph

2.2 Mosquitto Broker

Mosquitto Docker version broker is responsible for handling all MQTT message traffic in the local network. Mainly it will act as an intermediary in the communication between the mote and Node-Red. In the initial configuration of Docker-compose I established that the Mosquitto container must start working when the server starts up or be restarted autonomously in case of failure.

2.3 Node-Red

Node red, on the other hand, takes care of managing all the logic of the system. Each environmental parameter is mapped into a specific MQTT topic depending on the mote that generated it. Node-Red has two flows and a dashboard. The first flow deals with subscribing to the topics and tracing an average of the values of each parameter that comes from the various motes, after which it prints those values on the dashboard (Figure 7) and publishes them on the topics of the average values. Subsequently, the average values are sent to the second flow which takes care of saving them on the hard disk in a CSV file which is automatically named with the current date. In order not to open the file for writing too many times, the second flow waits for a queue of 10 values before saving the data, Figure 8.

3 Mote Side - Firmware

As anticipated before, the sensor network is composed of two boards that use ESP8266 as a WiFi module. The firmware that I wrote, in C ++ with Arduino IDE, on both boards has a similar structure in the connection algorithm, but it differs in the management of the sensors mounted on each board. The connection algorithm follows the diagram, Figure 6. First, the WiFi connection to the network of the first router is attempted. Once successful, the connection is attempted with the server and in particular with the MQTT broker. After that the sensor data transmission can begin. For managing MQTT messages I use PubSubClient library for the motes. At any time, if the connection fails or if the microcontroller is interrupted it goes into deep sleep mode for a few seconds minutes or hours depending on the type of error. This is intended for continuous use of the device.

Figure 3: NodeMCU Esp8266

3.1 ESP8266 Board

The main board based on NodeMCU module, whose schematic is shown in Figure 5, represents the heart of the system. It mounts two different sensors for measuring the ambient temperature, one analog through a thermistor, another digital, through a DHT11 module, which also collects humidity. The CSS811 CO_2 sensor is connected to the I^2C interface and works at a working voltage of 5V instead of 3.3V like the rest of this board. Provides a CO_2 saturation value in parts per million. An OLED Display is always available as a user interface via the I^2C, it can be waked up for few seconds through an handy pushbutton which triggered an interrupt in the shield.

3.2 Environment Board

Figure 4: Envy Esp8266

As a second sensor board I use an AZ-Envy ESP8266 module. The board already integrates a digital temperature and humidity sensor and an MQ-2 air quality sensor. It is a robust Gas sensor suitable for sensing LPG, Smoke, Alcohol, Propane, Hydrogen, Methane and Carbon Monoxide concentrations in the air.

3.3 Relay

In the future I will use an ESP-01s relay module connected to the ventilation system, specifically a 220/230VAC single-phase motor fan in a vent. In the working mode as web server it is possible to toggle it by means of a specific request in the HTTP format. The specific codes I wrote are available on the GitHub page of this project.

4 Conclusions

This project allowed me to touch many disciplines, including electronics, physics, C ++ firmware programming and IT. The most complicated part was that of the setup of the medium-high level linux infrastructure and the design the breadboard for the sensors keeping costs as low as possible. Not counting shipping costs and spare parts for my electronic lab, this project cost me around 140 euros. It is a lot, but it must be considered that most of the expenses are related to prototyping. The use of container technology has made the whole project much more stable and allows greater scalability, even if it hides traps in approaching linux for the first time like I did. I enclose in the appendices the diagrams mentioned previously and the images of the implementation. The link to the repository is on the first page.

A Table of Figures

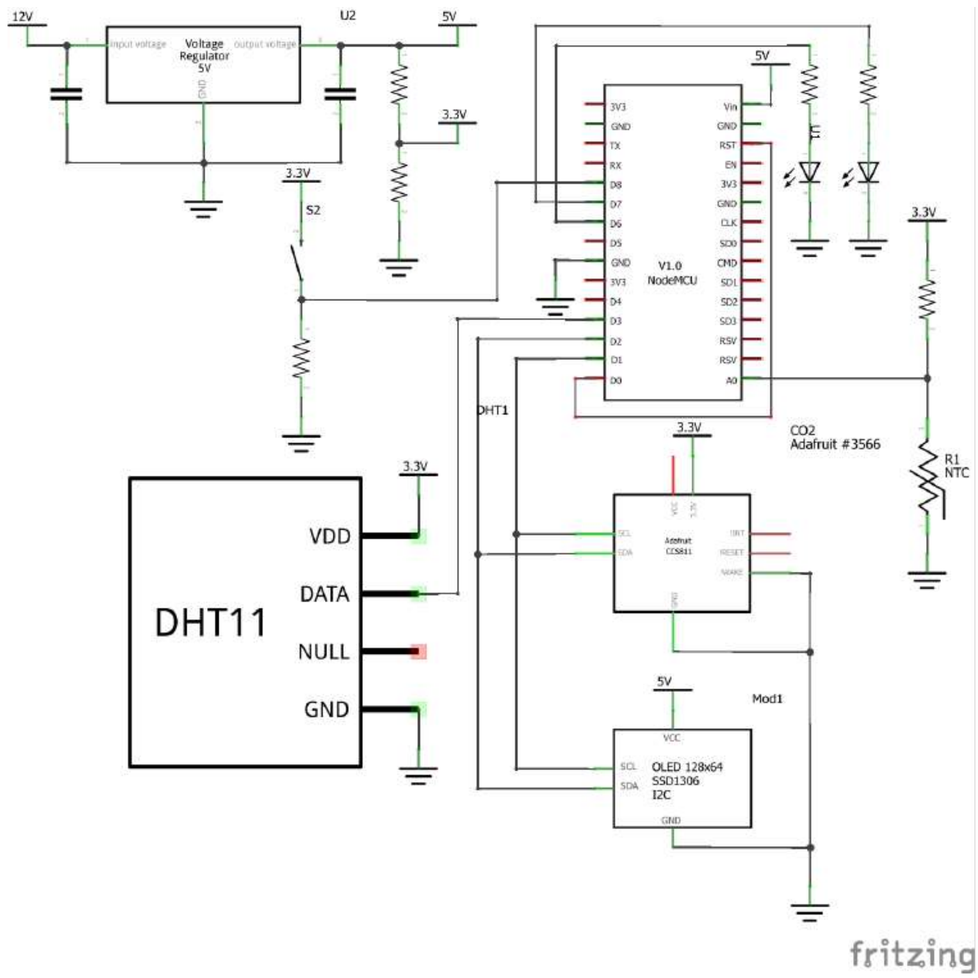

Figure 5: Schematic

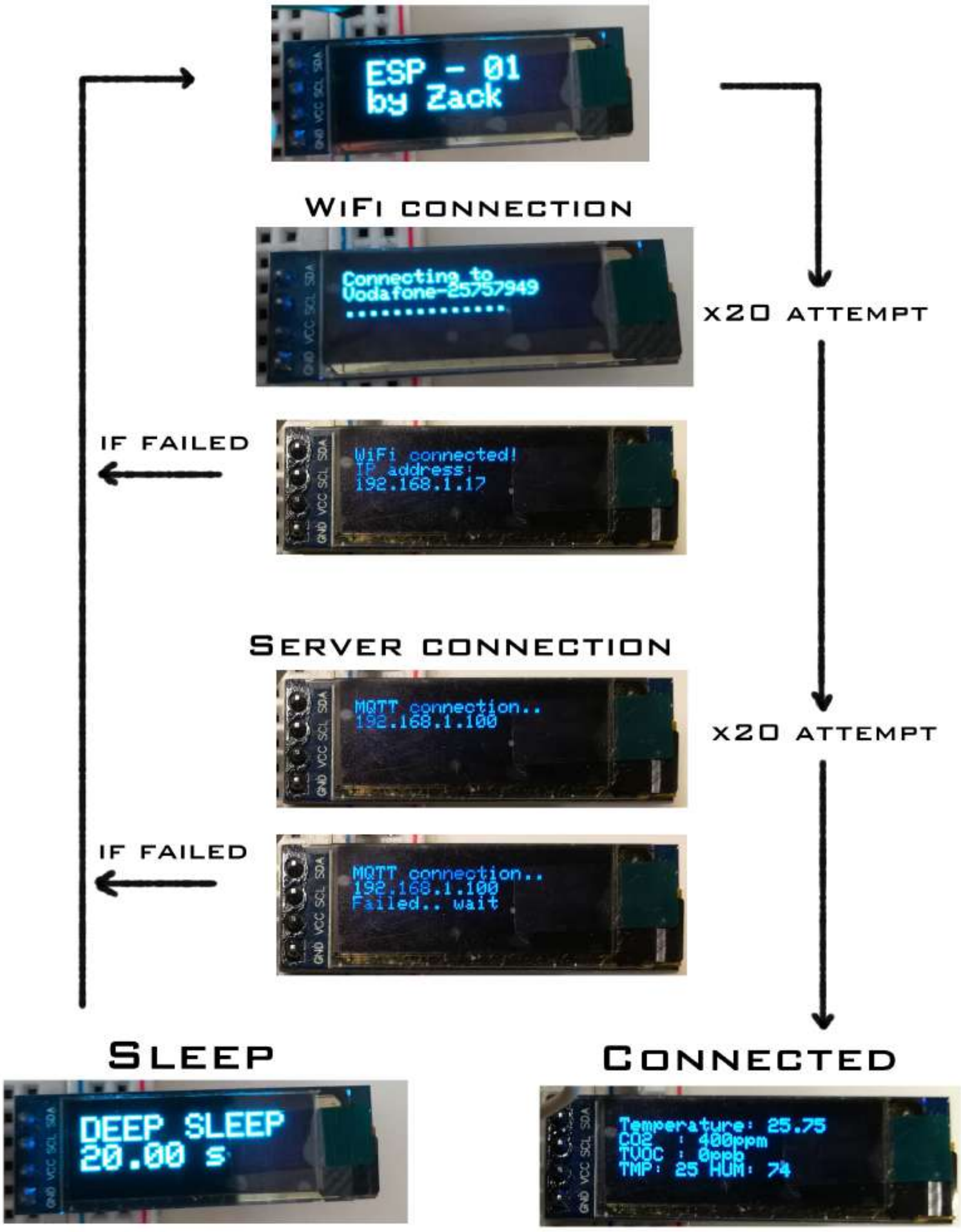

Figure 6: WiFi connection Algorithm

Figure 7: Node-Red Dashboard "http://192.168.1.100:1880/ui"

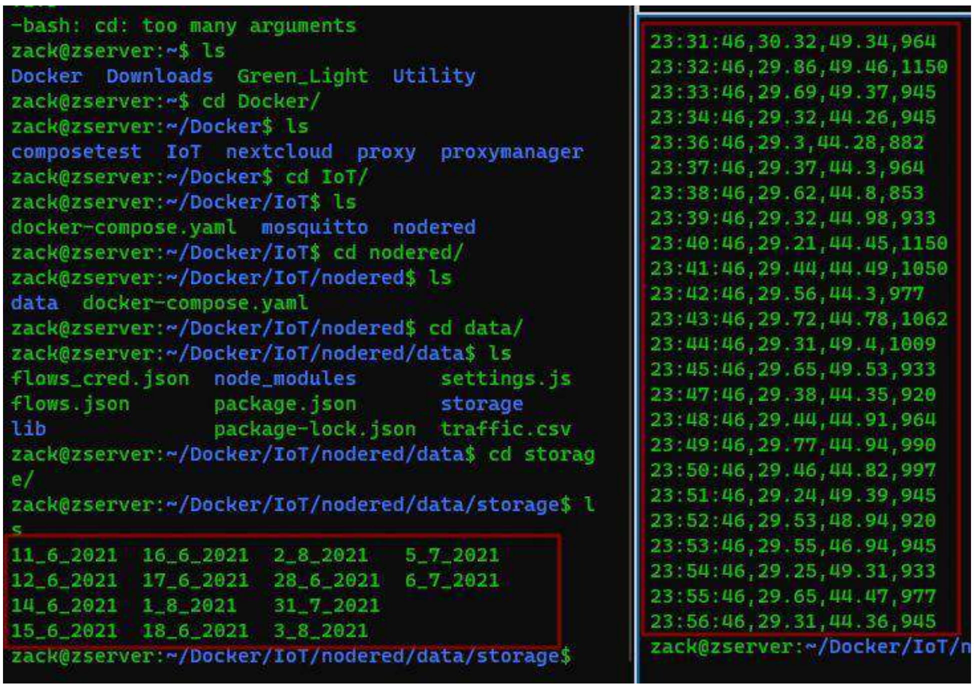

Figure 8: Terminal SSH view of CSV files daily divided

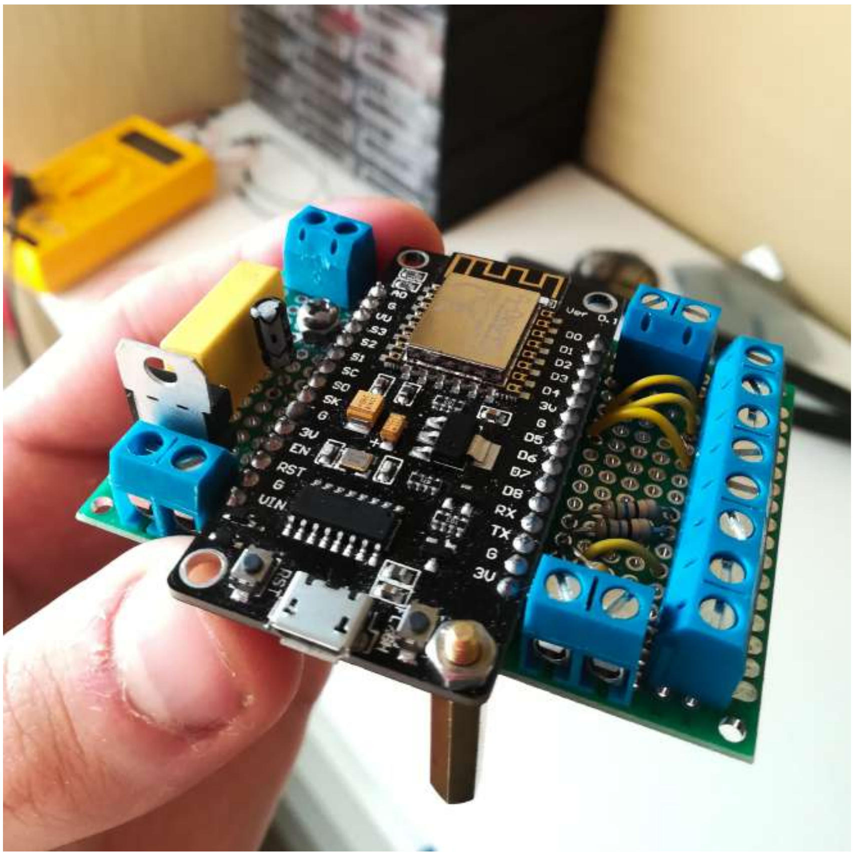

Figure 9: Custom sensor's PCB of ESP8266

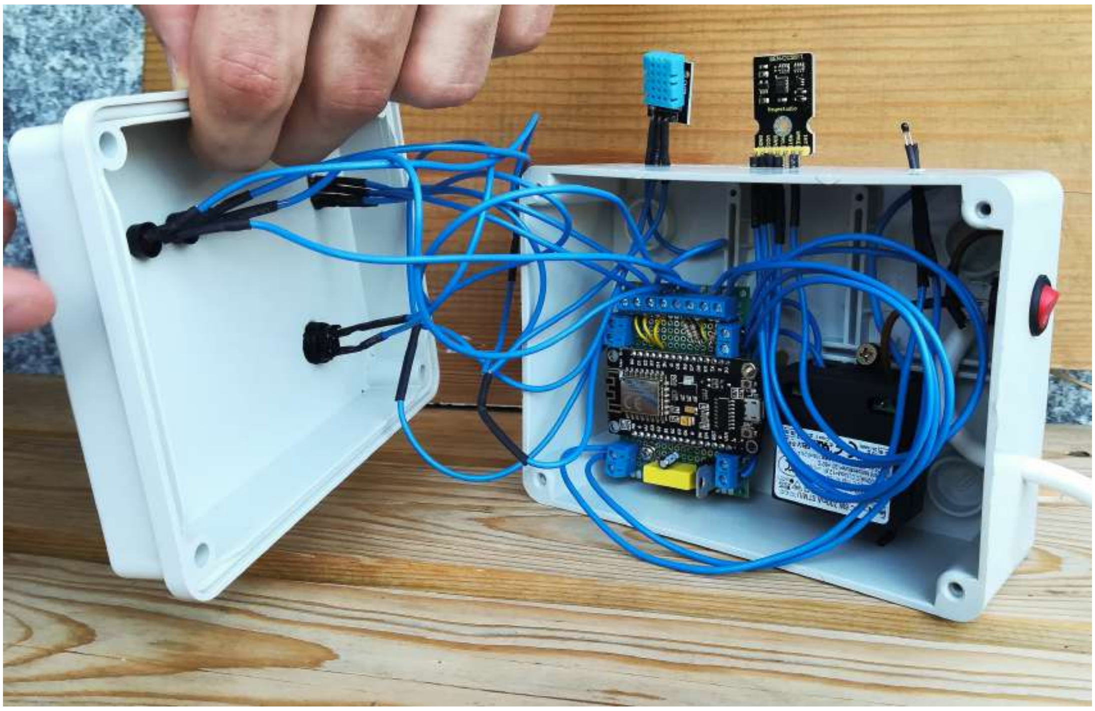

Figure 10: Internal wiring

Figure 11: Final Result

Figure 12: Envy Board installation

Figure 13: ESP8266 Board installation