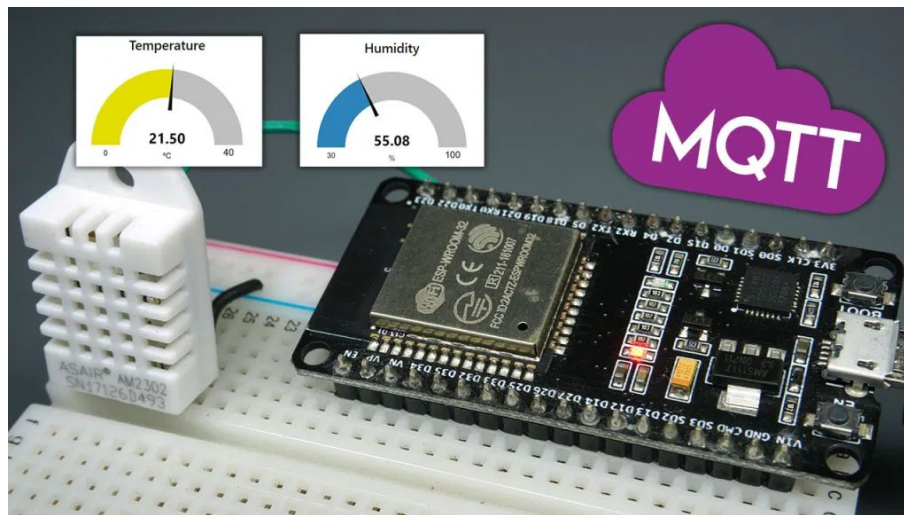


Internet Of Things

Rendiamo “intelligenti” i sensori

Cosa faremo?

In questa sezione del corso creeremo il nostro primo oggetto connesso! Si tratta di una *dashboard* in cui verranno mostrati i valori letti da alcuni sensori



Cosa faremo?



Questo è un workshop di gruppo, ma cosa significa?

- Ci saranno poche slide, andremo passo passo insieme quindi si richiede particolare attenzione alla spiegazione.
- Essendo un workshop, l'interazione con il gruppo e con l'insegnante è fondamentale: non abbiate paura di chiedere maggiori dettagli durante la spiegazione, interrompetemi, collaborate tra di voi.
- Le slide non sono un libro, se durante un esercizio c'è qualcosa che non ho detto/scritto, usate internet! Provate a risolvere il problema da soli, in extremis interviene l'insegnante e si risolve insieme.
- Dividetevi in gruppi di 5!

... prima di cominciare

Quando si sviluppa un progetto è importante tenere traccia dei propri progressi, prendere appunti e note su cosa si sta creando/sviluppando.

In generale, bisogna creare della **documentazione**, una specie di diario di bordo!

Per far ciò utilizzeremo il protocollo *git*, qualcuno lo conosce?



GitHub

1. Andiamo su <https://github.com/> e creiamo un account
2. Scarichiamo ed installiamo [Git](https://git-scm.com/downloads) (<https://git-scm.com/downloads>)
3. Scarichiamo ed installiamo [GitHub Desktop](https://desktop.github.com/) (<https://desktop.github.com/>)
4. Creiamo la nostra prima repository
5. Sincronizziamo la *repo* in locale
6. “*pushiamo*” il nostro primo file!

IN CASE OF FIRE 



1. git commit



2. git push



3. git out!

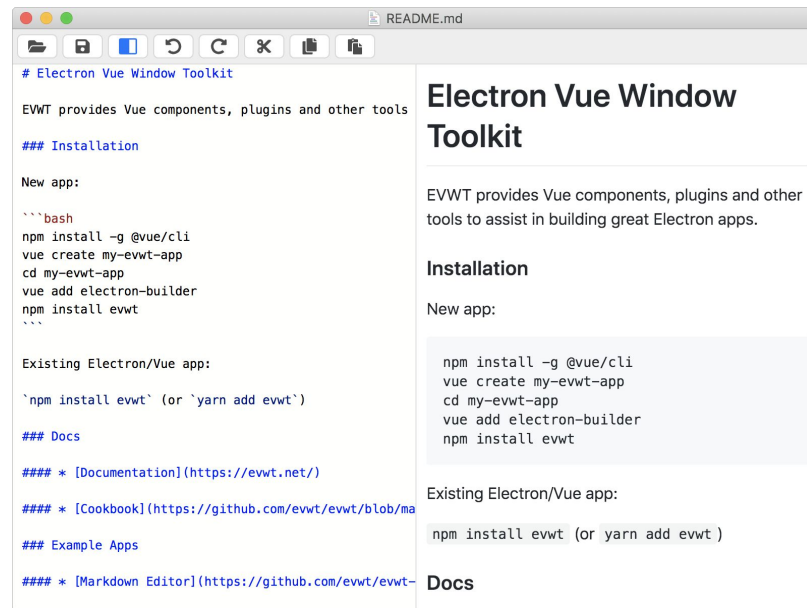
File Markdown

Il Markdown (*.md*) è un linguaggio di *markup*, ovvero serve a creare dei documenti.

Attraverso alcune regole, è possibile stilizzare il testo, creare note, tabelle, link, eccetera.

Senza dover utilizzare Word!

Al seguente [link](https://www.markdownguide.org/basic-syntax/) è presente una lista con degli esempi
(<https://www.markdownguide.org/basic-syntax/>)



Esercizio 1

Durante tutto il progetto produrremo della documentazione, tenendo traccia di tutti i nostri progressi:

1. creiamo un file “**README.md**”, e lasciamo vuoto per il momento.
2. creiamo una cartella “**diaries**” e al suo interno un file chiamamo “**14_07.md**”
3. aggiungiamo la cartella diaries al *.gitignore*
4. dentro questo file, un membro del team si occuperà di riportare tutti i progressi del gruppo: i dubbi che sono sorti, come sono stati risolti, il codice scritto, cosa fa il codice, eccetera.

Per ogni giornata del corso, l'incaricato alla documentazione svolgerà questo compito. Non dimenticate a fine giornata di pushare il file con i progressi!

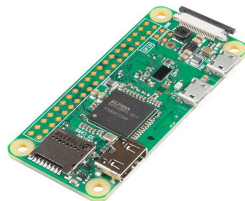
Schede utilizzate

Per svolgere il nostro progetto useremo due “schede”:

- [Esp32](#)



- [Raspberry Pi Zero W](#)



Configurazione Raspberry

Abbiamo scoperto che la raspberry in realtà è un computer in miniatura! Quindi come interagiamo con lei?

Idee?

Configurazione Raspberry

All'interno di questo workshop mostreremo due modalità di comunicazione:

- *SSH*
- *VNC*

Possiamo anche utilizzare monitor, tastiera e mouse come se fosse un pc normale. Ma non lo faremo per diversi motivi:

- La raspberry pi zero è molto poco potente (Cpu 1 Ghz, RAM 512 mb)
- Nell'IoT è importante comunicare con il server da remoto
- Non avevamo i fondi per comprare 3 monitor

Configurazione Raspberry - SSH

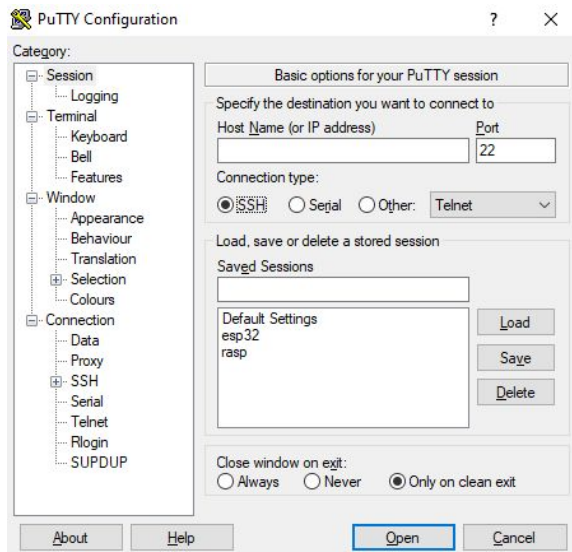
1. Inseriamo la scheda memoria nel nostro pc (chiudete tutte le varie schermate di “errore”)
2. Andiamo nella partizione “*boot*” e creiamo un file chiamato *ssh*
3. Creiamo un file “*wpa_supplicant.conf*” con il seguente contenuto

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="WIFI_SSID"
    scan_ssid=1
    psk="WIFI_PASSWORD"
    key_mgmt=WPA-PSK
}
```

Configurazione Raspberry - SSH

4. Scarichiamo [PuTTY](https://www.putty.org/) (<https://www.putty.org/>)
5. Colleghiamoci da remoto (username: pi, password: raspberry)



Configurazione Raspberry - VNC

1. Colleghiamoci da remoto con PuTTY
2. Digitiamo: *sudo raspi-config*
3. Dal menù comparso, selezioniamo “*Interface Options*”
4. Dal sottomenù comparso, selezioniamo “VNC”
5. Digitiamo: *sudo reboot*
6. Scarichiamo RealVNC Viewer (<https://www.realvnc.com/en/connect/download/viewer/>)
7. Inseriamo l'ip della nostra raspberry, username, password e connettiamoci!

Configurazione Esp32



1. Scarichiamo [Arduino IDE](https://www.arduino.cc/en/software/) (https://www.arduino.cc/en/software/)
2. Esercizio 2: seguite [questa](https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/) guida!
(https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/)

Configurazione Esp32

Problemi incontrati?

Può capitare che Windows non riconosca i driver, installiamoli dal seguente [link](https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers) e poi riavviamo. (<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>)

Testiamo l'esp32

Scriviamo il nostro primo codice e carichiamolo sull'esp32 per testarne il funzionamento

```
int LED_PIN = 2;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

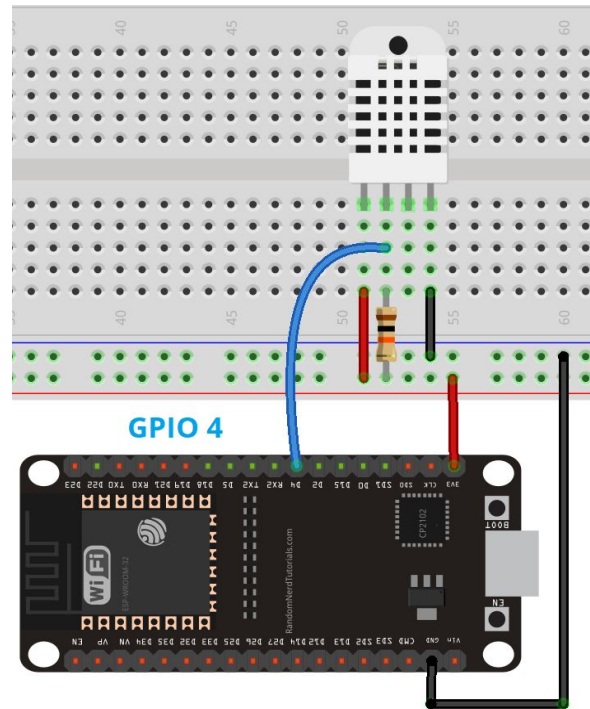

Configurazione Esp32

3. Installiamo le librerie del sensore di umidità DHT11
 - a. Andiamo nel gestore delle librerie (lo stesso da cui avete scaricato le librerie per l'esp32)
 - b. cerchiamo ed installiamo “DHT sensor library” by Adafruit
 - c. cerchiamo ed installiamo “Adafruit Unified Sensor” by Adafruit
4. Riavviamo Arduino IDE

Testiamo il sensore di umidità

Collegiamo il sensore di umidità all'esp32:

- GND (-) nel negativo della breadboard
- VCC (+) nel positivo della breadboard
- DATA (S) al pin G4 dell'esp
- la resistenza da 4.7 kΩ tra DATA (S) e il positivo della breadboard
- GND dell'esp32 al negativo della breadboard
- 3V3 dell'esp32 al positivo della breadboard



Testiamo il sensore di umidità

Scriviamo il codice per testare il sensore di umidità:

```
#include "DHT.h";
#include <WiFi.h>;

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

float temp;
float hum;

void setup() {
    ...
}
```

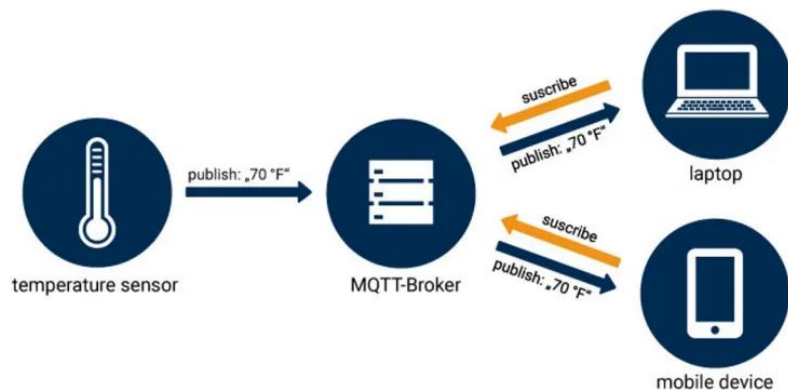
Testiamo il sensore di umidità

```
void setup() {  
  Serial.begin(9600);  
  Serial.println();  
  dht.begin();  
}  
  
void loop() {  
  hum = dht.readHumidity();  
  temp = dht.readTemperature();  
  if (isnan(temp) || isnan(hum)) {  
    Serial.println(F("Error: Failed to read from DHT sensor!"));  
  } else {  
    Serial.printf("Temp: %.2f \tHum: %.2f\n", temp, hum);  
  }  
  delay(2000);  
}
```

Comunicazione tra Raspberry ed Esp32

Entrambe le schede sono dotate di Bluetooth e WiFi, ma in generale quando si parla di IoT si intendono dispositivi connessi tramite una rete internet.

Per far comunicare queste due schede utilizzeremo il protocollo MQTT

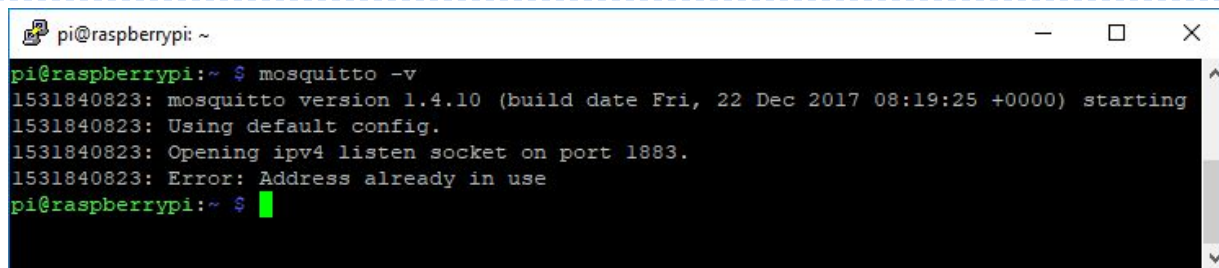


Comunicazione tra Raspberry ed Esp32

Per far ciò, dobbiamo prima installare il server sulla raspberry, e le librerie sull'esp32.

Raspberry

1. Connettiamoci tramite PuTTY alla raspberry
2. Digitiamo:
 - a. `sudo apt update`
 - b. `sudo apt install -y mosquitto mosquitto-clients`
 - c. `sudo systemctl enable mosquitto.service`
3. Testiamo digitando: `mosquitto -v`



```
pi@raspberrypi: ~  
pi@raspberrypi:~$ mosquitto -v  
1531840823: mosquitto version 1.4.10 (build date Fri, 22 Dec 2017 08:19:25 +0000) starting  
1531840823: Using default config.  
1531840823: Opening ipv4 listen socket on port 1883.  
1531840823: Error: Address already in use  
pi@raspberrypi:~$
```

Comunicazione tra Raspberry ed Esp32

Esp32

1. Scarichiamo la libreria dal seguente [link](https://github.com/marvinroger/async-mqtt-client/archive/master.zip)
(<https://github.com/marvinroger/async-mqtt-client/archive/master.zip>)
2. Estraiamo il file `.zip` e rinominiamolo, togliendo “master”: `async-mqtt-client-master` -> `async_mqtt_client`
3. spostiamo la cartella appena generata in > Documenti > Arduino > libraries
4. Riavviamo Arduino IDE

Comunicazione tra Raspberry ed Esp32

Esp32

5. Ripetiamo la stessa procedura con questa [libreria](https://github.com/me-no-dev/AsyncTCP/archive/master.zip)
(<https://github.com/me-no-dev/AsyncTCP/archive/master.zip>)

Codice client MQTT su Esp32

Nelle prossime slide scriveremo un codice da caricare sull'esp32, che si occuperà di inviare i dati al server mqtt (raspberry)

1. Importiamo tutte le librerie necessarie

```
#include "DHT.h";  
#include <WiFi.h>;  
  
extern "C" {  
    #include "freertos/FreeRTOS.h"  
    #include "freertos/timers.h"  
}  
#include <AsyncMqttClient.h>
```

Codice client MQTT su Esp32

2. Impostiamo le credenziali WiFi

```
#define WIFI_SSID "WIFI_SSID"  
#define WIFI_PASSWORD "WIFI_PASSWORD"
```

3. ???

```
#define MQTT_HOST IPAddress(192, 168, 1, 250)  
#define MQTT_PORT 1883
```

Codice client MQTT su Esp32

4. Impostiamo i *topics*, ovvero i canali in cui verranno pubblicati i dati dei sensori

```
#define MQTT_PUB_TEMP "esp32/dht/temperature"  
#define MQTT_PUB_HUM "esp32/dht/humidity"
```

5. ???

```
#define DHTPIN 4  
#define DHTTYPE DHT11  
DHT dht(DHTPIN, DHTTYPE);
```

Codice client MQTT su Esp32

6. Creiamo le variabili in cui leggeremo i dati dei sensori

```
float temp;  
float hum;
```

7. Creiamo le variabili per tenere traccia del tempo (invieremo dati aggiornati ogni 10 secondi)

```
unsigned long previousMillis = 0;  
const long interval = 10000;
```

Codice client MQTT su Esp32

8. Creiamo un oggetto Mqtt che si occuperà di gestire le richieste verso il server

```
AsyncMqttClient mqttClient;  
TimerHandle_t mqttReconnectTimer;  
TimerHandle_t wifiReconnectTimer;
```

9. Adesso creeremo due *funzioni* legate al wifi

```
void connectToWifi() {  
    Serial.println("Connecting to Wi-Fi...");  
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
}
```

Codice client MQTT su Esp32

```
void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case SYSTEM_EVENT_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            connectToMqtt();
            break;
        case SYSTEM_EVENT_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            xTimerStop(mqttReconnectTimer, 0);
            xTimerStart(wifiReconnectTimer, 0);
            break;
    }
}
```

Codice client MQTT su Esp32

10. Adesso invece, una serie di funzioni legate al protocollo MQTT

```
void connectToMqtt() {  
    Serial.println("Connecting to MQTT...");  
    mqttClient.connect();  
}
```

```
void onMqttPublish(uint16_t packetId) {  
    Serial.print("Publish acknowledged.");  
    Serial.print("  packetId: ");  
    Serial.println(packetId);  
}
```

Codice client MQTT su Esp32

```
void onMqttConnect(bool sessionPresent) {  
    Serial.println("Connected to MQTT.");  
    Serial.print("Session present: ");  
    Serial.println(sessionPresent);  
}
```

```
void onMqttPublish(uint16_t packetId) {  
    Serial.print("Publish acknowledged.");  
    Serial.print("  packetId: ");  
    Serial.println(packetId);  
}
```


Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `setup() { ... }`

11. Inizializziamo il monitor seriale

```
void setup() {  
  Serial.begin(9600);  
  Serial.println();  
  ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `setup() { ... }`

12. Inizializziamo il sensore di umidità

```
void setup() {  
  ...  
  dht.begin();  
  ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà
DENTRO la funzione `setup() { ... }`

14. Queste “brutte” righe di codice permettono all’esp32 di riconnettersi nel caso ci sia un errore di rete

```
void setup() {  
    ...  
    mqttReconnectTimer = xTimerCreate(  
        "mqttTimer",  
        pdMS_TO_TICKS(2000),  
        pdFALSE, (void*)0,  
        reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt)  
    );  
  
    wifiReconnectTimer = xTimerCreate(  
        "wifiTimer",  
        pdMS_TO_TICKS(2000),  
        pdFALSE, (void*)0,  
        reinterpret_cast<TimerCallbackFunction_t>(connectToWifi)  
    );  
    ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `setup() { ... }`

15. Diciamo all'esp32 che quando succede un evento wifi (qualsiasi azione della scheda legata alla rete) deve eseguire la nostra funzione creata al punto 9

```
void setup() {  
    ...  
    WiFi.onEvent(WiFiEvent);  
    ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `setup() { ... }`

16. Effettuiamo la stessa cosa, ma con gli eventi MQTT

```
void setup() {  
    ...  
    mqttClient.onConnect(onMqttConnect);  
    mqttClient.onDisconnect(onMqttDisconnect);  
    mqttClient.onPublish(onMqttPublish);  
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);  
    ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `setup() { ... }`

17. Infine, ci colleghiamo al WiFi

```
void setup() {  
    ...  
    connectToWifi();  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `loop() { ... }`

18. Prendiamo il tempo corrente

```
void loop() {  
    unsigned long currentMillis = millis();  
    ...  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `loop() { ... }`

19. Prendiamo il tempo corrente, se la differenza con l'istante precedente è maggiore dell'intervallo (10 sec) effettuiamo le altre operazioni

```
void loop() {  
    ...  
    if (currentMillis - previousMillis >= interval) {  
        previousMillis = currentMillis;  
        ...  
    }  
}
```


Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `loop() { ... }`

20. Leggiamo i valori dal sensore

```
void loop() {  
    ...  
    hum = dht.readHumidity();  
    temp = dht.readTemperature();  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `loop() { ... }`

21. Leggiamo i valori dal sensore, se sono dei numeri reali (questo indica che non ci sono stati errori) stampiamo un errore, altrimenti...

```
void loop() {  
    ...  
    if (isnan(temp) || isnan(hum)) {  
        Serial.println(F("Error: Failed to read from DHT sensor!"));  
    } else {  
        ...  
    }  
}
```

Codice client MQTT su Esp32

NB: Tutto il codice seguente sarà **DENTRO** la funzione `loop() { ... }`

22. Leggiamo i valori dal sensore, se sono dei numeri reali (questo indica che non ci sono stati errori) stampiamo un errore, altrimenti...

```
void loop() {  
    ...  
    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true, String(temp).c_str());  
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_TEMP, packetIdPub1);  
    Serial.printf("Message: %.2f \n", temp);  
}
```

Codice client MQTT su Esp32

void loop() {

```
void loop() {  
    ...  
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true, String(hum).c_str());  
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM, packetIdPub2);  
    Serial.printf("Message: %.2f \n", hum);  
}
```

NB: Tutto il codice seguente sarà **DENTRO** la funzione *loop()* { ... }

23. Chiudiamo tutti i blocchi

```
void loop() {  
    ...  
}  
  
}
```

Codice client MQTT su Esp32

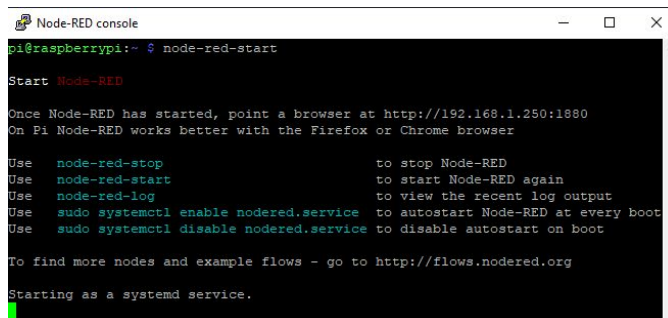
A questo punto il nostro Esp32 invia i dati dei sensori nei canali

- "esp32/dht/temperature"
- "esp32/dht/humidity"

Adesso non ci resta che configurare la raspberry in modo che possa “intercettare” questi dati e mostrarli a schermo!

Visualizzazione dati su Raspberry con NodeRED

1. Connettiamoci tramite PuTTY alla raspberry
2. Digitiamo:
 - a. `cd ~/.node-red`
 - b. `npm install node-red-dashboard`
 - c. `sudo reboot`
3. Dopo essersi riavviato, digitiamo: `node-red-start`



```
Node-RED console
pi@raspberrypi:~$ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.250:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use  node-red-stop          to stop Node-RED
Use  node-red-start         to start Node-RED again
Use  node-red-log           to view the recent log output
Use  sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use  sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
```

Visualizzazione dati su Raspberry con NodeRED

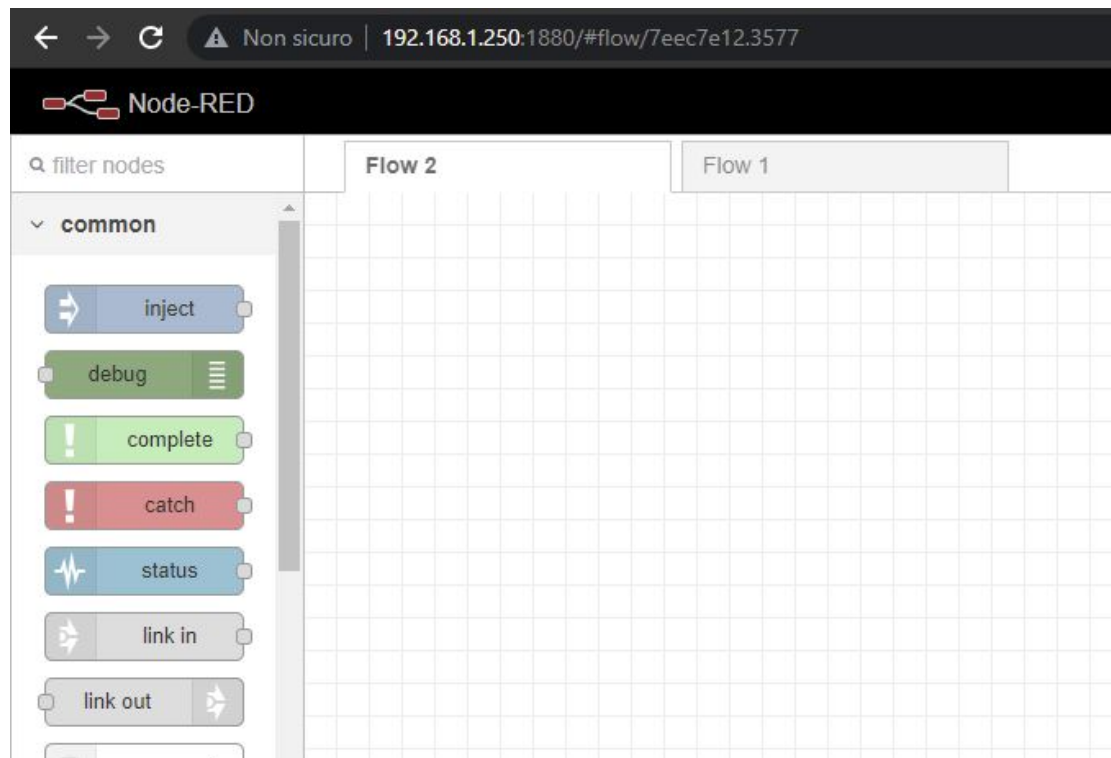
Facciamo un recap:

- L'esp32 invia ogni 10 secondi i dati al server
- La raspberry (server) riceve i dati, ma ancora non li visualizza
- Con Node-RED creeremo una *dashboard* per leggere i dati

È tutto connesso in rete quindi basta accedere, da browser, da qualsiasi altro dispositivo all'indirizzo: <http://raspberry-ip-address:1880>

Apriete Chrome (o Mozilla) da un pc e recatevi al link sopra (ad esempio 192.168.1.250:1880)

Visualizzazione dati su Raspberry con NodeRED



Visualizzazione dati su Raspberry con NodeRED

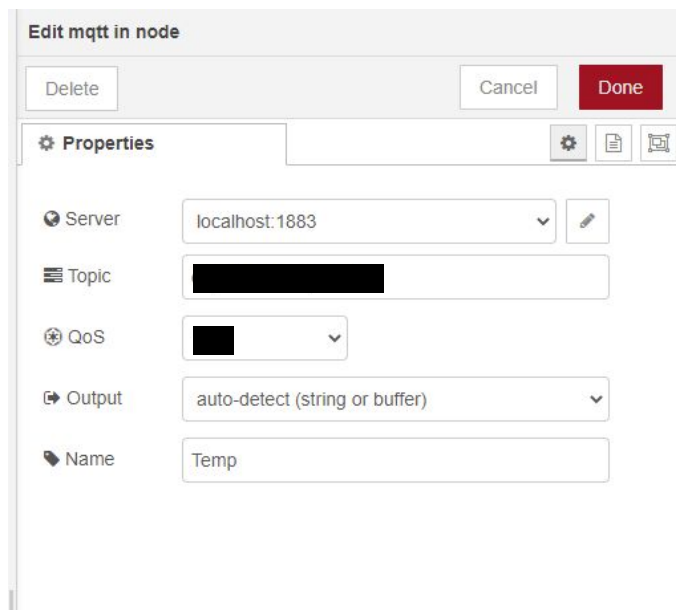
1. Spostiamo nell'area di lavoro due nodi "MQTT"



2. Facciamo doppio click su uno di essi

Visualizzazione dati su Raspberry con NodeRED

3. Che parametri mettiamo negli spazi censurati?



The screenshot shows the 'Edit mqtt in node' configuration window in NodeRED. It features a title bar with 'Edit mqtt in node', a 'Delete' button, and 'Cancel' and 'Done' buttons. Below the title bar is a 'Properties' tab with icons for settings, a document, and a monitor. The configuration fields are as follows:

- Server:** A dropdown menu showing 'localhost:1883' with a pencil icon to its right.
- Topic:** A text input field containing a blacked-out (redacted) string.
- QoS:** A dropdown menu showing a blacked-out (redacted) value.
- Output:** A dropdown menu showing 'auto-detect (string or buffer)'.
- Name:** A text input field containing 'Temp'.

Visualizzazione dati su Raspberry con NodeRED

4. Impostare anche il secondo nodo MQTT
5. Spostiamo nell'area di lavoro due nodi "gauge"



Visualizzazione dati su Raspberry con NodeRED

6. Configuriamoli come segue

Edit gauge node

Delete Cancel Done

Properties

Group [Home] DHT

Size auto

Type Gauge

Label Temperature

Value format {{value}}

Units °C

Range min 0 max 40

Colour gradient

Sectors 0 optional optional 40

Name Temp

Edit gauge node

Delete Cancel Done

Properties

Group [Home] DHT

Size auto

Type Gauge

Label Humidity

Value format {{value}}

Units %

Range min 0 max 100

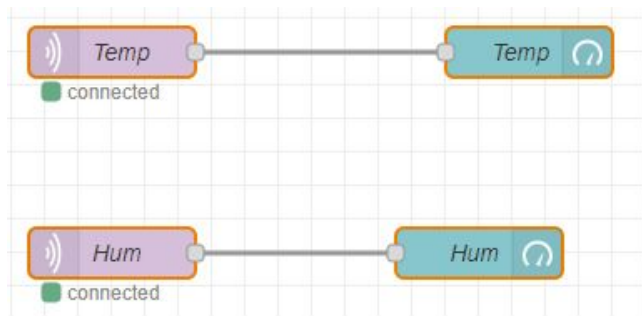
Colour gradient

Sectors 0 optional optional 100

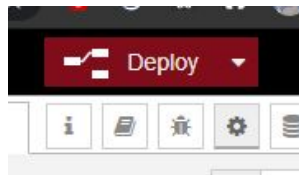
Name Hum

Visualizzazione dati su Raspberry con NodeRED

7. Colleghiamo i rispettivi nodi



8. “Deployamo”!



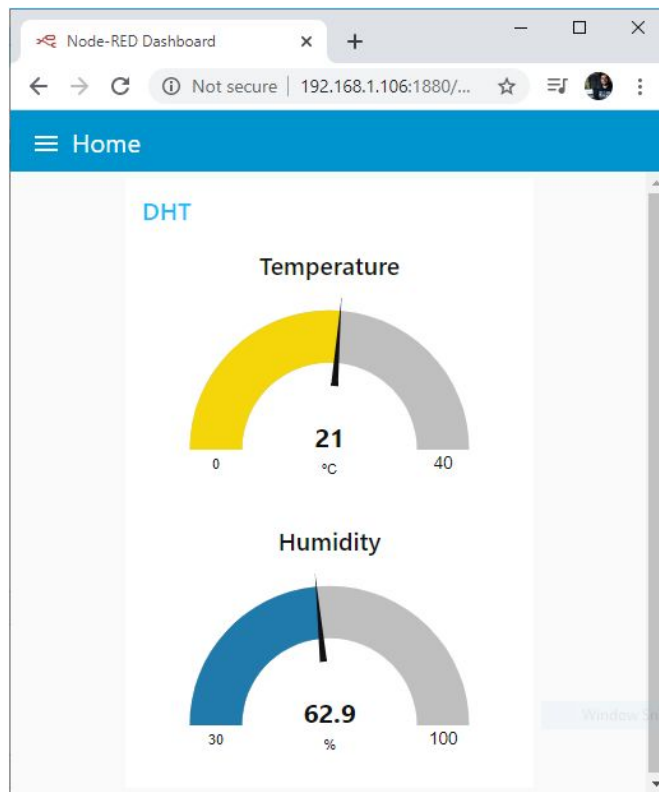
Visualizzazione dati su Raspberry con NodeRED

A questo punto, la situazione è questa:

- L'esp32 invia ogni 10 secondi i dati al server
- La raspberry (server) riceve i dati, ma ancora non li visualizza
- Abbiamo creato la dashboard

Non ci resta che visualizzare i dati. Prendete un vostro smartphone e andate all'indirizzo:
<http://raspberry-ip-address:1880/ui> (ad esempio 192.168.1.250:1880/ui)

Visualizzazione dati su Raspberry con NodeRED



Conclusioni

Siamo arrivati alla fine del progetto, adesso parliamone (non è una minaccia...)

Quali sono secondo voi i limiti del nostro progetto?

Conclusioni

Siamo arrivati alla fine del progetto, adesso parliamone (non è una minaccia...)

Come potremo risolverli?

Fonti

Repository del Fablab: <https://github.com/fablabws/iot-humidity-sensor>

Client SSH su windows: <https://docs.microsoft.com/it-it/windows/iot-core/connect-your-device/ssh>

SSH: <https://www.losant.com/blog/getting-started-with-the-raspberry-pi-zero-w-without-a-monitor>

VNC: <https://www.circuitbasics.com/access-raspberry-pi-desktop-remote-connection/>

Esp32:

<https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Installare MQTT su Raspberry:

<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>

MQTT: <https://randomnerdtutorials.com/esp32-mqtt-publish-dht11-dht22-arduino/>