

Dalla Riva Alessandro
Scozzai Samuele
Propedo Demien
Tavano Matteo



Progetto di Internet of Things aa. 2022-2023

Realizzazione di un sistema IoT in grado di acquisire parametri ambientali in real-time e presentarli in un Cloud, attraverso un protocollo di comunicazione MQTT.

Abstract: *Questa relazione dettaglia la progettazione e l'implementazione di un sistema IoT per il monitoraggio ambientale basato su un microcontrollore ESP8266 e un sensore multiparametro BME280. Il sistema è stato sviluppato per acquisire in tempo reale parametri critici come temperatura (massima e minima), pressione atmosferica e umidità. La comunicazione dei dati avviene attraverso il protocollo MQTT tramite connessione WiFi, consentendo una trasmissione efficiente e affidabile. I dati acquisiti vengono successivamente visualizzati in un cloud Thingspeak, fornendo un'interfaccia accessibile e intuitiva per il monitoraggio remoto.*

Introduzione:

Nell'era sempre più interconnessa dell'**Internet of Things**, il monitoraggio dei parametri ambientali all'interno di un sistema assume un ruolo cruciale per garantire il corretto funzionamento, la sicurezza e l'efficienza delle operazioni. La capacità di **raccogliere** e **analizzare** dati in tempo reale provenienti da sensori ambientali consente una risposta immediata a variazioni critiche, contribuendo a prevenire guasti e ottimizzare le risorse. In questo contesto, la scelta di un protocollo di comunicazione affidabile è fondamentale: **MQTT**, grazie alla sua leggerezza, efficienza e capacità di gestire la comunicazione tra dispositivi eterogenei, emerge come una soluzione ideale. La trasmissione di dati attraverso **MQTT** consente una comunicazione efficiente, riducendo la latenza e ottimizzando l'utilizzo della larghezza di banda. Inoltre, la **sicurezza** dei dati è prioritaria, e l'adozione di canali sicuri per la visualizzazione di questi parametri ambientali fornisce un ulteriore strato di protezione, assicurando che le informazioni sensibili siano trattate in modo sicuro e accessibili solo a coloro che ne hanno l'autorizzazione. Questa sinergia tra monitoraggio ambientale, protocolli affidabili e canali sicuri di comunicazione costituisce la base per sistemi avanzati, pronti a fronteggiare le sfide e sfruttare appieno le opportunità offerte dall'evoluzione dell'Internet of Things.

Analisi dei requisiti

Per condurre un'**analisi dei requisiti** completa per un progetto di monitoraggio ambientale basato su IoT, è fondamentale identificare in modo dettagliato le esigenze e le aspettative degli utenti, definire i requisiti funzionali e non funzionali, nonché considerare le limitazioni e i vincoli del sistema. Di seguito, forniamo un'analisi dei requisiti suddivisa in diverse categorie:

- **Rilevamento dei Parametri Ambientali:**
 - Il sistema deve essere in grado di rilevare parametri quali temperatura, umidità e pressione atmosferica.

- **Comunicazione e Trasmissione Dati:**
 - Il sistema deve utilizzare un protocollo per la trasmissione efficiente e affidabile dei dati tra il sensore e il sistema di gestione.
 - Deve essere prevista la capacità di gestire una quantità moderata di dati in tempo reale.

- **Elaborazione e Analisi dei Dati:**
 - Il sistema deve elaborare i dati ricevuti, tramite grafici cartesiani e linee di tendenza.
 - Deve essere possibile generare report periodici sull'andamento dei parametri ambientali.

- **Affidabilità:**
 - Il sistema deve garantire l'**affidabilità** nella raccolta e trasmissione dei dati, anche in presenza di perdita di connettività temporanea.
 - Deve essere prevista una strategia di riconnessione in grado di poter ripristinare il sistema al più presto.

- **Sicurezza:**
 - Tutti i dati trasmessi devono essere **crittografati** per garantire la sicurezza delle informazioni.
 - L'accesso al sistema deve richiedere autenticazione multi-fattore.

- **Performance:**
 - Il tempo di risposta del sistema deve essere **ottimizzato** per garantire una visualizzazione real-time dei dati.

- Deve essere definito un limite massimo di latenza per la trasmissione dei dati.
 - Il **consumo di energia** deve essere ridotto al minimo indispensabile, al fine di poter essere un sistema valido.
 - Lo **spazio fisico** allocato dal dispositivo deve essere minimale, ovvero occupare la minor superficie possibile.
- **Vincoli e Limitazioni:**
 - Il sistema deve essere compatibile con le **risorse hardware** disponibili nei dispositivi IoT utilizzati per il monitoraggio ambientale.
 - Il sistema deve essere progettato per funzionare su diverse **tipologie di reti**, inclusi scenari con connettività intermittente o limitata.
 - Utilizzo del sistema in ambienti tipicamente interni, non essendo provvisto momentaneamente di una copertura isolante.

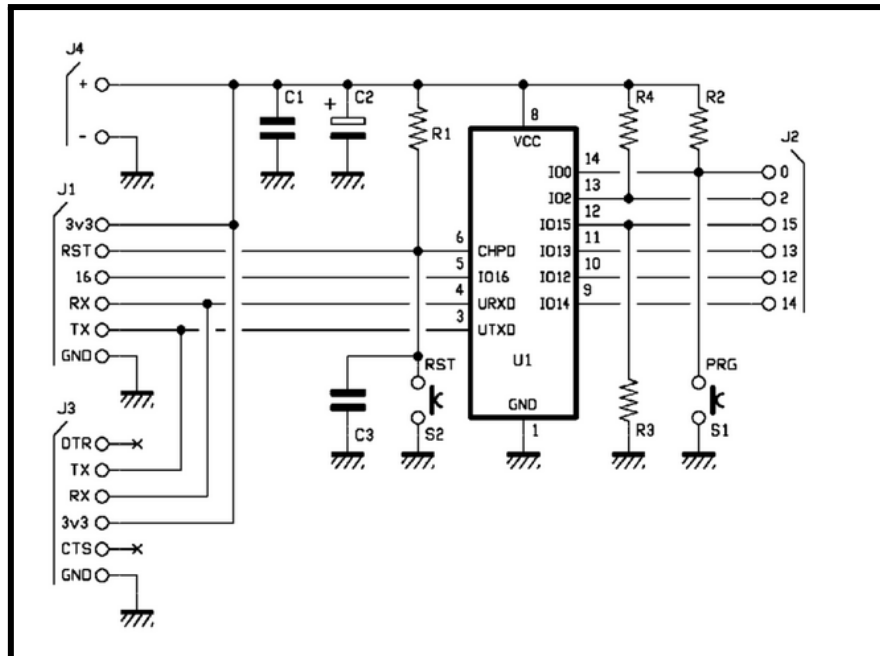
Progettazione

La scelta del microcontrollore **ESP8266** per la realizzazione del sistema di monitoraggio ambientale è stata guidata da diverse considerazioni fondamentali. La versatilità e l'efficienza del chip **ESP8266**, che integra un processore a bassa potenza e un modulo Wi-Fi, si sono rivelate decisive per soddisfare i requisiti specifici del progetto. La presenza di componenti base integrate, quali il processore a 32 bit, la memoria flash e la capacità di connessione *Wi-Fi*, ha consentito di ridurre notevolmente il numero di componenti esterni, semplificando il layout del circuito stampato e ottimizzando le dimensioni del dispositivo. Inoltre, l'**ESP8266** offre una vasta comunità di sviluppatori e un ampio supporto di risorse online, facilitando lo sviluppo, la programmazione e la risoluzione di eventuali problematiche. La decisione di adottare un microcontrollore che incorpora le componenti essenziali necessarie al progetto riflette la ricerca di un equilibrio tra prestazioni, efficienza e facilità di implementazione, elementi fondamentali per garantire il successo del sistema di monitoraggio ambientale.

Sfortunatamente questo modulo viene spesso commercializzato su schede di prototipazione che integrano altri componenti, con il risultato di peggiorare sensibilmente il consumo in corrente, tanto da dover pensare da zero una nuova scheda elettronica. Invece di progettare una specifica scheda per questo progetto abbiamo pensato di realizzare una generica scheda basata su **ESP8266** con i soli componenti essenziali.

La scheda è stata reperita su un sito di elettronica on-line.

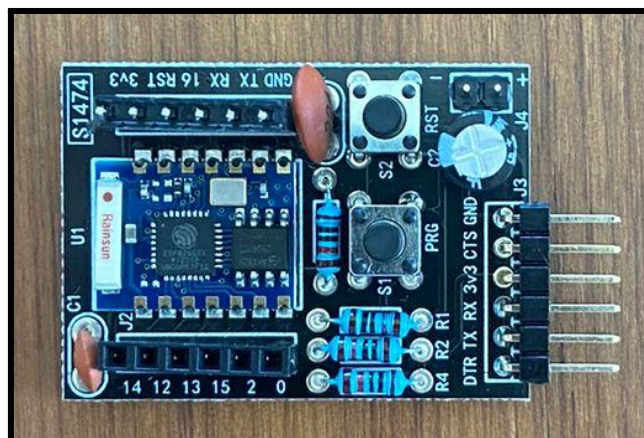
Architettura del sistema elettronico:



Il circuito che utilizzeremo, rappresentato nell'immagine, è molto semplice in confronto ad un generico modulo *ESP8266*, pur mantenendo le stesse caratteristiche di interesse che ci saranno utili per la realizzazione del sistema.

Il resistore *R1* è posto sulla linea del reset, che viene azionata manualmente dal pulsante *S2* quando si desidera resettare il modulo *ESP*. La funzione di reset ci sarà utile sia per caricare lo sketch, che per decifrare sul serial monitor i messaggi di testo.

Il condensatore *C3* è necessario per utilizzare la funzione **Wake on from sleep**. Il resistore *R2* è connesso alla linea di programmazione che viene azionata dal pulsante *S1* quando si desidera programmare il modulo *ESP*.



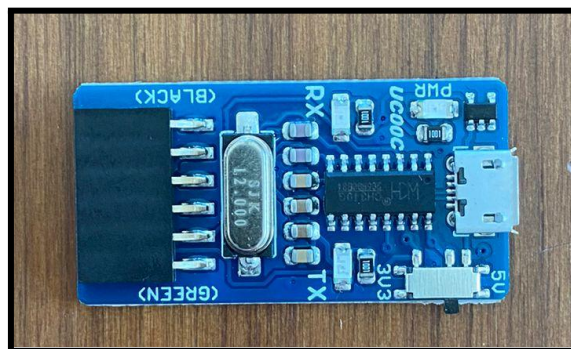
I resistori *R3* e *R4* sono necessari per porre rispettivamente il pin a livello alto ed il pin a livello basso in modo che il modulo ESP esegua all'avvio il corretto boot con l'esecuzione del programma in memoria.

Come già spiegato, il microcontrollore è dotato di due tasti, *RST* e *PRG*, utili per la programmazione e per svolgere varie funzioni, come il decrypt dei messaggi su serial monitor di *Arduino Ide*.

Scelta delle componenti Hardware utilizzate:

Per quanto riguarda le altre componenti elettroniche utilizzate all'interno di questo progetto, descriviamo, il sensore multiparametro *BME280*, un piccolo chip realizzato da Bosch e il convertitore da USB a UART 3.3V e 5V.

Convertitore da USB a UART 3.3V e 5V:



Un **convertitore UART** (Universal Asynchronous Receiver/Transmitter) da 3.3V a 5V è essenziale quando si collegano dispositivi che operano a tensioni diverse. In una comunicazione seriale UART, i dispositivi possono utilizzare livelli di tensione diversi per i segnali di trasmissione (TX) e ricezione (RX). Un convertitore UART 3.3V a 5V si occupa di adattare i segnali tra questi due livelli.

Ad esempio, se un microcontrollore o un sensore opera a 3.3V e deve comunicare con un dispositivo che opera a 5V, il convertitore si interpone tra di essi, garantendo che i segnali UART siano compatibili. Questo previene danni dovuti a una tensione incompatibile e consente una comunicazione affidabile tra dispositivi con diverse tensioni di alimentazione.

Il convertitore tipicamente offre una soluzione plug-and-play, con connettori standard per facilitare l'integrazione nei circuiti. Questo componente è cruciale in applicazioni in cui è necessario garantire una corretta interoperabilità e proteggere i dispositivi collegati da danni causati da differenze di tensione.

Per la realizzazione e la programmazione del circuito, abbiamo utilizzato anche un **cavetto** da USB a Micro-USB, per il collegamento alla seriale del Computer.

Sensore BME280:



Il **sensore BME280** è un componente chiave nell'ambito dell'IoT per il monitoraggio ambientale, in quanto offre la misurazione simultanea di temperatura, umidità e pressione atmosferica. Dotato di un elevato livello di precisione e stabilità, il *BME280* consente una rilevazione affidabile delle condizioni ambientali. Funzionante con tensioni di alimentazione basse e caratterizzato da un basso consumo energetico, è particolarmente adatto per applicazioni IoT alimentate a batteria. La comunicazione con dispositivi di raccolta dati avviene comunemente attraverso interfacce standard come I2C e SPI, facilitando l'integrazione con una vasta gamma di piattaforme IoT. Le sue dimensioni compatte e il design robusto lo rendono ideale per l'implementazione in sensori connessi alla rete, fornendo dati di qualità per applicazioni quali il monitoraggio ambientale, la meteorologia e la gestione energetica nell'ambito dell'Internet of Things.

Scelta delle componenti Software e Cloud utilizzate:

L'adozione della piattaforma cloud **ThingSpeak** e dell'**Arduino IDE** come ambiente di sviluppo per il progetto di monitoraggio ambientale ha dimostrato di offrire un connubio potente di flessibilità e semplicità operativa. *ThingSpeak*, con la sua natura basata su cloud, ha consentito di archiviare e visualizzare agevolmente i dati raccolti dai sensori ambientali in tempo reale. La sua interfaccia intuitiva e la capacità di generare grafici e report personalizzati hanno semplificato notevolmente l'analisi dei dati e la loro interpretazione. Inoltre, ThingSpeak offre funzionalità di notifica e integrazione con altre piattaforme, potenziando la capacità di risposta del sistema. D'altra parte, l'utilizzo dell'Arduino IDE come ambiente di sviluppo ha offerto un'esperienza di programmazione accessibile, grazie alla sua sintassi chiara e alla vasta comunità di supporto online. La piattaforma Arduino IDE è stata scelta anche per la sua compatibilità diretta con il microcontrollore ESP8266, semplificando il processo di programmazione e integrazione del dispositivo nel sistema complessivo. In definitiva, la combinazione di ThingSpeak e Arduino IDE ha fornito una soluzione completa e coerente per il progetto, unendo la potenza del cloud computing alla facilità di sviluppo e programmazione offerta dall'ambiente Arduino.

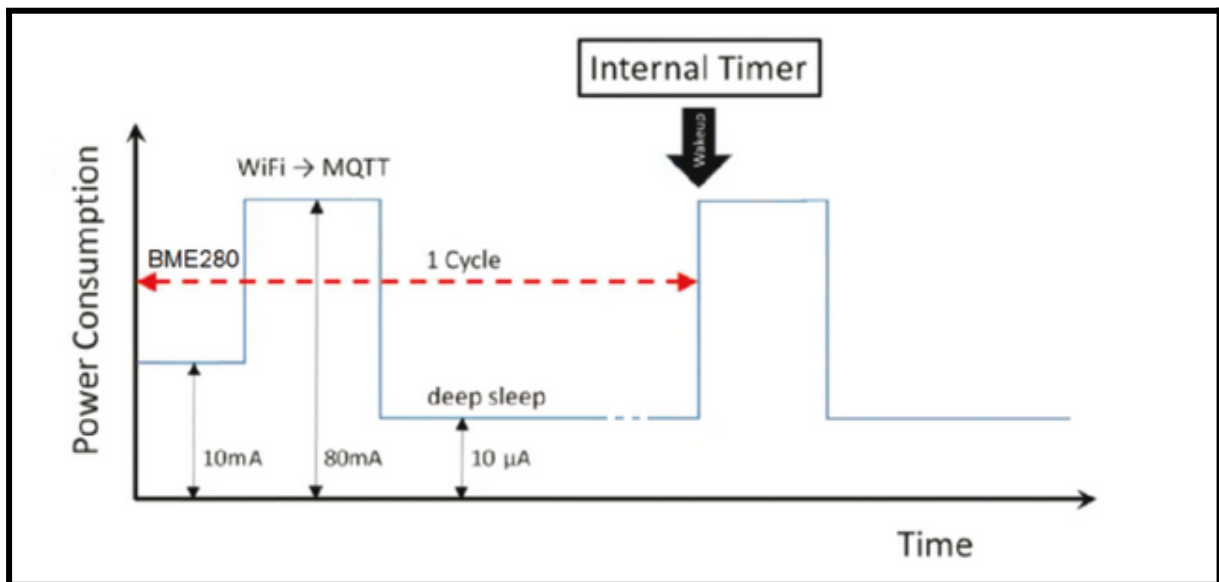
Logica di funzionamento:

Dopo ogni risveglio il modulo ESP interroga il sensore BME280 acquisendo i valori di temperatura, umidità e pressione. Il sensore è configurato per rimanere sempre in uno stato a basso consumo (ma non spento) per cui è in grado di fornire i valori dei parametri ambientali in modo molto veloce senza la necessità di un warm-up.

Questa modalità è chiamata *weather monitoring* e la lettura avviene forzatamente da parte del modulo ESP, terminata la quale il modulo è di nuovo in stand-by. In questa fase la sezione *WiFi* del modulo ESP è spenta e l'assorbimento è di pochi milliampere, il sensore *BME280* assorbe solo una manciata di microampere e solo nel momento in cui avviene la lettura dei dati.

Successivamente viene accesa la sezione radio e si provvede alla connessione alla rete *WiFi* operazione che nel nostro caso si conclude in pochi secondi; nel caso la connessione non avvenisse nell'arco di un tempo massimo di 10 secondi il modulo ritornerebbe in deep sleep e riprovarebbe la connessione al prossimo risveglio.

Appena connessa alla rete la scheda contatta il broker *MQTT* di Thingspeak per inviargli tutti i dati, compresi quello della tensione ai capi della batteria, dopodiché spegne il *WiFi* ed entra nuovamente nello stato di deep sleep.



Nel grafico riportato sopra, possiamo notare come ad ogni ciclo, il consumo di energia (asse y) sia variabile. Nella prima fase, il consumo medio si aggira sui **10 mA** (acquisizione dei parametri).

Nella seconda fase, abbiamo il picco di consumo (**80 mA**) in corrispondenza alla connessione del modulo *WiFi* integrato nel chip, e al passaggio dei valori tramite *MQTT*.

L'ultima fase, corrisponde al passaggio deep sleep, caratterizzata da un consumo minimale, circa **10 µA**. Il processo si ripete ad ogni ciclo.

Per questa applicazione è necessario *cortocircuitare* il **pin 16** con il pin **RST**.

In un uso reale di questo circuito ipotizziamo di aggiornare i dati ad intervalli di un'ora in quanto ventiquattro misure al giorno risultano essere più che sufficienti.

Visto che il circuito rimane inutilizzato per la maggior parte del tempo potremmo sfruttare la modalità *deep sleep* del microcontrollore **ESP** con *wake up* automatico dopo un certo tempo prefissato.

L'istruzione messa a disposizione nell'**IDE** di Arduino si chiama

ESP.deepSleep(TimeToSleep).

Durante la modalità di **deep sleep** tutti i circuiti interni sono disattivati tranne il **modulo RTC**, che rimane in funzione aggiornando un **contatore** interno ad intervalli regolari di 1µsec, l'assorbimento in corrente scende drasticamente a pochi microampere.

Il modulo RTC, allo scadere del tempo impostato dalla variabile *TimeToSleep*, porta il pin 16 a livello basso, che essendo connesso al pin RST del MCU, consente il reset del modulo ed il riavvio programma dall'inizio. Il **condensatore C3** ha lo scopo di prolungare la durata dell'impulso di reset per il microcontrollore.

Una volta risvegliato, il modulo legge i valori forniti dal sensore *BME280*, attiva il *WiFi* e si connette alla rete per inviare i dati al cloud, operazione che impiega circa 6 secondi.

Siccome questa operazione verrà compiuta ad intervalli di un'ora avremmo un circuito che rimane attivo per **6 secondi ogni 3.600 secondi (ogni ora...)** con una rilevante diminuzione dell'energia assorbita. Il protocollo utilizzato per trasferire i dati dal modulo ESP al cloud è MQTT particolarmente apprezzato per la sua efficienza.

Programmazione dell'ESP:

Per la **programmazione** serve il convertitore USB-Seriale esterno pertanto, possiamo utilizzare il convertitore UART, precedentemente citato. Connettendo il **connettore J3** al dispositivo UART, risulta essere perfettamente compatibile. Notiamo che i pin DTR e CTS non vengono utilizzati.

Procedura di programmazione: *Quando si desidera programmare il modulo utilizzando l'IDE di Arduino è necessario abilitare il modulo alla programmazione tramite la seguente sequenza: premere e mantenere premuto il pulsante PRG, premere per un istante il pulsante RST, quindi rilasciare il pulsante PRG.*

Questa procedura deve essere eseguita prima di caricare lo sketch per predisporre il modulo ESP alla ricezione del programma. Se si avvia la programmazione senza aver eseguito la procedura, il debug dell'IDE di Arduino lo segnalerà scrivendo una serie di puntini e linee.

Al termine basta premere brevemente il pulsante di reset.

```
Sto caricando...
Lo sketch usa 257712 byte (51%) dello spazio disponib
Le variabili globali usano 26572 byte (32%) di memori
esptool.py v2.6
2.6
esptool.py v2.6
Serial port COM8
Connecting.....
Chip is ESP8266EX
Features: WiFi
MAC: 5c:cf:7f:34:1a:86
Uploading stub...
Running stub...
```

Procedura di programmazione da Arduino IDE.

Lo sketch:

Per quanto riguarda l'implementazione effettiva del sistema, ci siamo basati sullo sketch, reperibile nella cartella: *ESP8266_thingspeak_MQTT_BME280_deepsleepE*

Lo sketch si appoggia ad alcune librerie famose per arduino IDE, tra cui:

```
#include "PubSubClient.h"
#include <ESP8266WiFi.h>
#include "secrets.h"
#include <Wire.h>
#include <Adafruit_BME280.h>
```

- **PubSubClient**: <https://github.com/knolleary/pubsubclient>, per la comunicazione MQTT;
- **<ESP8266WiFi.h>**: per la connettività Wifi;
- **secrets.h**, un file di configurazione dove andremo ad impostare valori personali.
- **<Wire.h>**, libreria per gestire alcune funzionalità elettriche
- **<Adafruit_BME280.h>**, libreria che ci consente di modificare variabili acquisite dal sensore *BME280*.

La variabile **DEBUG** se impostata su true permette di abilitare i messaggi su Serial Monitor di Arduino, utile nei primi test per verificare che tutto funzioni correttamente come evidenziato in figura sottostante:

```
16:02:59.180 ->
16:02:59.695 ->
16:02:59.695 -> Connecting to WiFi SSID MySSID....
16:03:04.572 -> WiFi connected with IP address: 192.168.1.102
16:03:04.572 -> Attempting MQTT connection...
16:03:05.868 -> Connected
16:03:05.868 -> T= 32.08°C H= 45.13% P=1015.22hPa V=3.34V
16:03:05.868 -> Sending payload: field1=32.08&field2=45.13&field3=1015.22&field4=3.34&stat
16:03:05.868 -> Publish ok
16:03:06.540 -> Go to sleep!
16:03:06.540 ->
16:03:06.540 ->
```

Le credenziali di accesso sono tutte raccolte nel file *secrets.h* esse sono state modificate con le credenziali di accesso alla rete WiFi e con i codici di accesso forniti da Thingspeak.

```
#define SECRET_SSID "MySSID" // replace MySSID with your WiFi network name
#define SECRET_PASS "MyPassword" // replace MyPassword with your WiFi password
#define channelID 123456 // Thingspeak channel number
#define writeAPIKey "ABCDEFGHILMNOPQR" // Thingspeak API Key
```

Per evitare di dover attendere tempi troppo lunghi, in fase di debug è possibile modificare anche il parametro **UPDATE_INTERVAL_SECONDS** per ridurre il tempo di invio a solo un minuto.

Il resto del codice implementa i passaggi logici di cui abbiamo trattato, ed è possibile visualizzarlo in chiaro.

Raccogliamo alcuni spezzoni di codice, che andranno a dividere il codice a seconda delle funzioni implementate.

Le funzioni principali sono commentate.

void() Setup:

```
// Connect BME280 GND TO pin14 OR board's GND
pinMode(14, OUTPUT);
digitalWrite(14, LOW);

Serial.begin(115200);
delay(10);

// BME280 Initialise I2C communication as MASTER
Wire.begin(13, 12); //Wire.begin([SDA], [SCL])

BMESStatus = bme.begin();
if (!BMESStatus)
{
    if (DEBUG) { Serial.println("Could not find BME280!"); }
    //while (1);
}

bme.setSampling(Adafruit_BME280::MODE_FORCED,
                Adafruit_BME280::SAMPLING_X1, // temperature
                Adafruit_BME280::SAMPLING_X1, // pressure
                Adafruit_BME280::SAMPLING_X1, // humidity
                Adafruit_BME280::FILTER_OFF );

// read values from the sensor
float temperature, humidity, pressure;
```

Setup del BME:

```
if (BMESStatus)
{
    temperature = bme.readTemperature();
    humidity = bme.readHumidity();
    pressure = bme.readPressure() / 100.0F;
}
else
{
    if (DEBUG) Serial.println("Could not find BME280!");
    temperature=0;
    humidity=0;
    pressure=0;
}

float voltage = ESP.getVcc();
voltage = voltage/1024.0; //volt

if (DEBUG)
{
    Serial.println("T= " + String(temperature) + "°C H= " + String(humidity) + "%" P= " + String(pressure) + "hPa V=" + voltage + "V");
}
```

Costruzione del payload MQTT:

```
// Construct MQTT payload
payload="field1=";
payload+=temperature;
payload+="&field2=";
payload+=humidity;
payload+="&field3=";
payload+=pressure;
payload+="&field4=";
payload+=voltage;
payload+="&status=MQTTPUBLISH";
```

Connessione Wifi:

```
//Connect to Wifi
if (DEBUG)
{
  Serial.println();
  Serial.print("\nConnecting to WiFi SSID ");
  Serial.print(SECRET_SSID);
}

WiFi.begin(SECRET_SSID, SECRET_PASS);

int timeOut=10; // Time out to connect is 10 seconds
while ((WiFi.status() != WL_CONNECTED) && timeOut>0)
{
  delay(1000);
  if (DEBUG) { Serial.print("."); }
  timeOut--;
}

if (timeOut==0) //No WiFi!
{
  if (DEBUG) Serial.println("\nTimeOut Connection, go to sleep!\n\n");
  ESP.deepSleep(1E6 * UPDATE_INTERVAL_SECONDS);
}

if (DEBUG) // Yes WiFi
{
  Serial.print("\nWiFi connected with IP address: ");
  Serial.println(WiFi.localIP());
}
```

Riconnessione al client MQTT:

```
// Reconnect if MQTT client is not connected.
if (!client.connected())
{
  reconnect();
}

mqttpublish();

delay(200); // Waiting for transmission to complete!!! (ci vuole)
WiFi.disconnect( true );
delay( 1 );

if (DEBUG) { Serial.println("Go to sleep!\n\n"); }
// Sleep ESP and disable wifi at wakeup
ESP.deepSleep( 1E6 * UPDATE_INTERVAL_SECONDS, WAKE_RF_DISABLED );
```

MQTT Publish:

```
void mqttpublish()
{
    // read values from the sensor
    float temperature, humidity, pressure;

    if (DEBUG)
    {
        Serial.print("Sending payload: ");
        Serial.println(payload);
    }

    // Create a topic string and publish data to ThingSpeak channel feed.
    String topicString = "channels/" + String( channelID ) + "/publish/" + String(writeAPIKey);
    unsigned int length=topicString.length();
    char topicBuffer[length];
    topicString.toCharArray(topicBuffer,length+1);

    if (client.publish(topicBuffer, (char*) payload.c_str()))
    {
        // || || if (DEBUG) Serial.println("Publish ok");
    }
    else
    {
        // || || if (DEBUG) Serial.println("Publish failed");
    }
}
```

Riconnessione:

```
void reconnect()
{
    String clientName="MY-ESP";
    // Loop until we're reconnected
    while (!client.connected())
    {
        if (DEBUG) Serial.println("Attempting MQTT connection...");
        // Try to connect to the MQTT broker
        if (client.connect((char*) clientName.c_str()))
        {
            if (DEBUG) Serial.println("Connected");
        }
        else
        {
            if (DEBUG)
            {
                Serial.print("failed, try again");
                // Print to know why the connection failed.
                // See http://pubsubclient.knolleary.net/api.html#state for the failure code explanation.
                Serial.print(client.state());
                Serial.println(" try again in 2 seconds");
            }
            delay(2000);
        }
    }
}
```

Configurazione del Cloud:

Tra i vari cloud a disposizione abbiamo deciso di utilizzare **Thingspeak** perché, come vedremo, è quello che permette una maggiore personalizzazione dei dati raccolti offrendo la possibilità di una post analisi dei dati tramite appositi script.

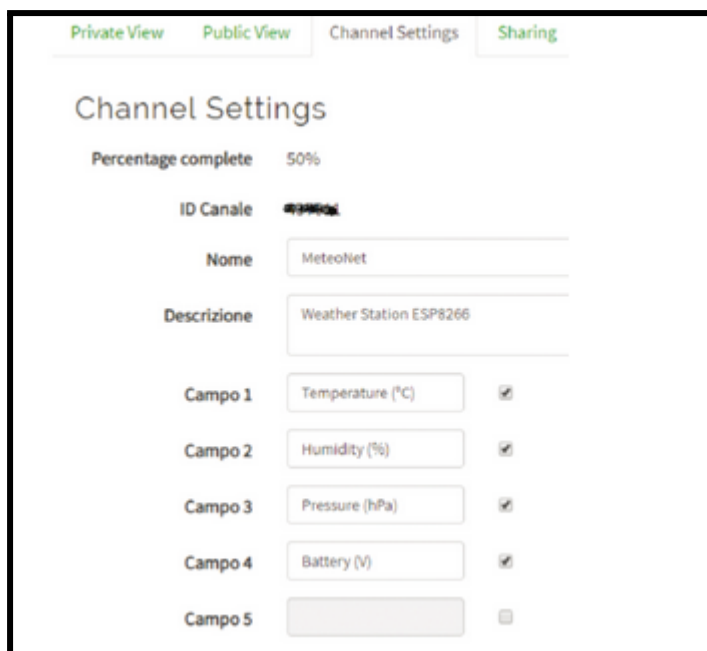
Configurare un canale su ThingSpeak è un processo relativamente semplice. Di seguito, descriviamo la procedura per configurare un canale su ThingSpeak:

1. **Creazione di un Account ThingSpeak:** Andando su <https://thingspeak.com/> è possibile creare un account gratuito.

2. **Accesso al Dashboard:** Accediamo all' account ThingSpeak e andiamo alla sezione dashboard.

3. **Creazione di un Nuovo Canale:**

- Nel menu principale, selezioniamo "Channels" e poi "My Channels".
- Clicchiamo su "New Channel" per creare un nuovo canale.
- L'immagine, ci mostra un'anteprima sulla creazione del nuovo canale.



The screenshot displays the 'Channel Settings' interface on the ThingSpeak platform. At the top, there are tabs for 'Private View', 'Public View', 'Channel Settings' (which is active), and 'Sharing'. Below the tabs, the 'Channel Settings' section is shown with a 'Percentage complete' indicator at 50%. The form includes an 'ID Canale' field with a placeholder image, a 'Nome' field containing 'Meteohet', and a 'Descrizione' field containing 'Weather Station ESP8266'. There are five 'Campo' (Field) settings, each with a text input and a checkbox. Campo 1 is 'Temperature (°C)' with a checked checkbox. Campo 2 is 'Humidity (%)' with a checked checkbox. Campo 3 is 'Pressure (hPa)' with a checked checkbox. Campo 4 is 'Battery (V)' with a checked checkbox. Campo 5 is an empty field with an unchecked checkbox.

4. **Configurazione delle Impostazioni del Canale:**

- Compiliamo ora le informazioni richieste, inclusi il nome del canale, una breve descrizione e l'URL di un'immagine (opzionale) che rappresenti il canale.
- Specifichiamo i campi di interesse che il dispositivo invierà a ThingSpeak.

Nello specifico configureremo:

- Temperatura (*Campo1*);
- Umidità (*Campo2*)
- Pressione atmosferica (*Campo3*)
- Batteria (*Campo4*).

5. Configurazione dei Campi del Canale:

- Per ciascun campo specificato, inseriamo un nome descrittivo (es. "Temperatura", "Umidità", ecc.).
- E' possibile anche impostare le unità di misura corrispondenti a ciascun campo, come "°C" per la temperatura o "% RH" per l'umidità.

6. Salvataggio delle Modifiche:

- Facciamo clic su "Save Channel" per salvare le modifiche apportate al canale.

7. Ottenere le API Keys:

- Nella pagina del tuo canale, andiamo alla scheda "API Keys".
- Troveremo due tipi di chiavi: una per la lettura (**Read API Key**) e una per la scrittura (**Write API Key**).
- La chiave di lettura viene utilizzata per recuperare i dati dal canale, mentre la chiave di scrittura è necessaria per inviare dati al canale. Nel codice, in particolare nel file di configurazione secrets.h, andremo a salvare all'interno di due variabili distinte il numero del canale, e la chiave API di scrittura.



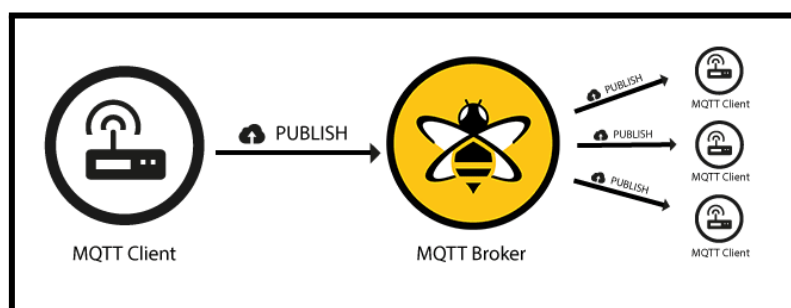
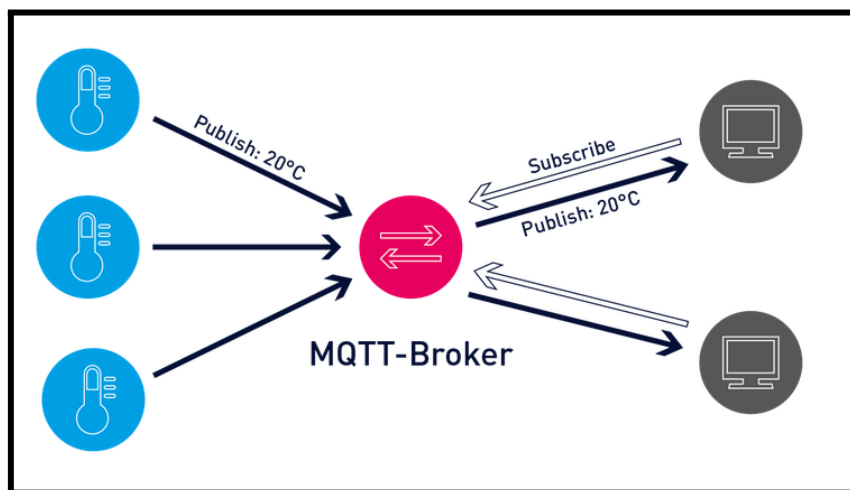
Configurazione Device MQTT:

ThingSpeak offre la flessibilità di configurare un **dispositivo MQTT** direttamente sulla piattaforma, consentendo un'integrazione semplice e potente con dispositivi IoT. La configurazione di un dispositivo MQTT su ThingSpeak consente di acquisire dati da sensori connessi al dispositivo e trasmetterli in tempo reale attraverso il protocollo MQTT. Questo approccio consente una gestione centralizzata dei dati, facilitando la visualizzazione e l'analisi attraverso il dashboard di ThingSpeak. La piattaforma supporta sia la pubblicazione che la sottoscrizione di messaggi MQTT, permettendo ai dispositivi di inviare dati ai propri canali o di ricevere comandi e istruzioni da ThingSpeak. Questa caratteristica si rivela particolarmente utile quando si desidera implementare un sistema di monitoraggio e controllo remoto, consentendo l'interazione bidirezionale tra i dispositivi IoT e la piattaforma cloud. La configurazione di un dispositivo MQTT su ThingSpeak si traduce in un'elevata versatilità e un'integrazione semplificata, consentendo di sfruttare appieno le potenzialità dell'Internet of Things.

Cenni teorici: funzionamento di MQTT

MQTT è l'acronimo di **Message Queuing Telemetry Transport** e indica un protocollo di trasmissione dati TCP/IP basato su un modello di pubblicazione e sottoscrizione che opera attraverso un apposito message broker.

MQTT utilizza un modello di comunicazione **publish/subscribe**, dove i dispositivi pubblicano messaggi su specifici "topic" e altri dispositivi si sottoscrivono a tali topic per ricevere i messaggi di loro interesse. Supporta diversi livelli di qualità del servizio (**QoS**) per garantire la consegna affidabile dei messaggi. MQTT è broker-based, con un broker che funge da intermediario tra i dispositivi, facilitando la comunicazione e la gestione dei messaggi. Grazie alla sua efficienza e flessibilità, MQTT è ampiamente utilizzato nell'implementazione di sistemi IoT, domotici e reti di sensori. La sua capacità di gestire connessioni instabili lo rende uno strumento potente per la comunicazione affidabile tra dispositivi distribuiti.



Risultati

I risultati della **fase di acquisizione** dati sono stati inizialmente promettenti, con il sistema che è riuscito con successo a monitorare e registrare i parametri ambientali per un periodo di alcuni giorni. Tuttavia, è importante sottolineare che successivamente si è verificato un imprevisto che ha portato a una difficoltà nell'ottenere ulteriori dati. Nonostante sforzi significativi per risolvere l'errore, il problema ha presentato delle sfide tecniche che, al momento, non sono state completamente superate.

Questo inconveniente ha evidenziato l'importanza di una robusta fase di test e di un monitoraggio continuo per individuare e affrontare eventuali problematiche tempestivamente. L'esperienza ha fornito preziose lezioni su aspetti critici della progettazione e implementazione del sistema di monitoraggio ambientale, aprendo la strada a possibili miglioramenti e correzioni nella fase successiva dello sviluppo. Nonostante l'ostacolo incontrato, questa fase del progetto ha contribuito a rafforzare la comprensione delle sfide pratiche associate all'implementazione di soluzioni IoT e ha posto le basi per future ottimizzazioni e aggiornamenti.

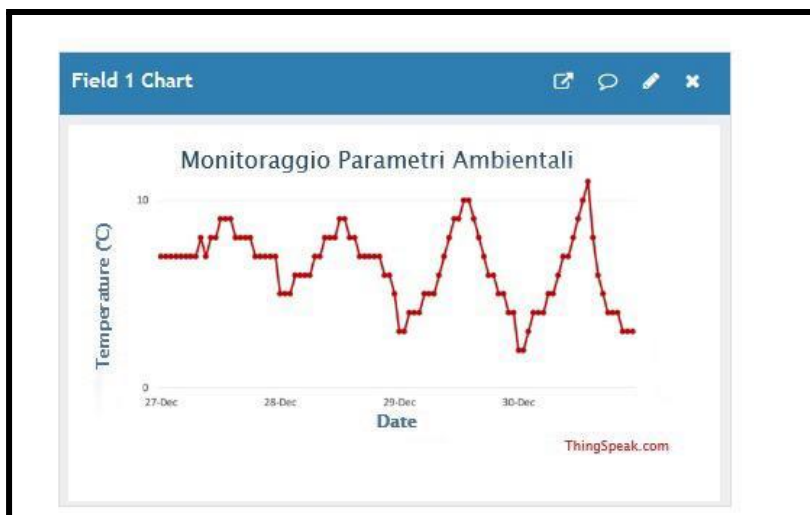
Nello specifico, ci siamo ritrovati dinanzi ad un errore di connessione con il server, ripetuto.

Returns

- -4 : MQTT_CONNECTION_TIMEOUT - the server didn't respond within the keepalive time

Tuttavia, lavorando da remoto siamo riusciti ad acquisire alcuni grafici nelle prime fasi di test. I dati acquisiti sono relativi al periodo 27-30 Dec in cui il sistema riusciva a soddisfare tutte le specifiche definite in precedenza.

Grafico Relativo alla Temperatura:



Possiamo evincere dal grafico una temperatura massima, in prossimità del 30 Dicembre. In questa data, si sono superati i 10 °C. Altri punti di massimo sono prossimi agli orari più caldi della giornata. Per quanto riguarda le temperature minime, esse si sono registrate in

prossimità delle prime ore del mattino. Notiamo il caldo record di questo dicembre, che non ha registrato temperature inferiori ai 0°C

Grafico Relativo alla Pressione Atmosferica:

Per quanto riguarda la pressione atmosferica, essa è misurata in hPa e delinea fluttuazioni un po' più stabili.

Essa mostra un trend oscillante, che inizialmente si concentra attorno ai 1025 hPa, per poi scendere e disporsi attorno al valore di 1020 hPa.

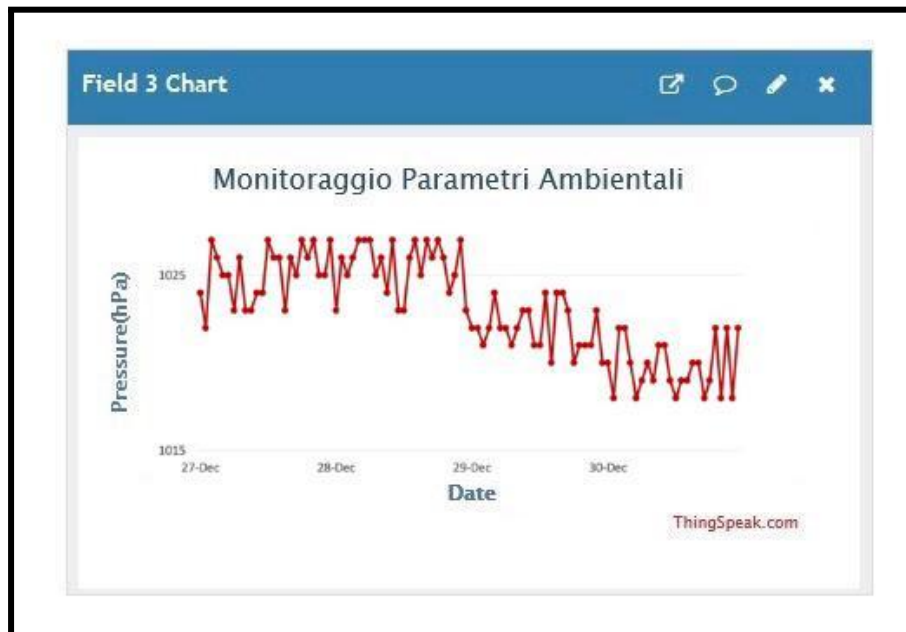


Grafico Relativo all'Umidità nell'aria:



Infine, notiamo che il grafico percentuale dell'umidità, ci delinea un dicembre sostanzialmente molto umido, con valori distribuiti attorno al 90%.

Il grafico mostra dei punti di minimo tra il 27 e il 28 dicembre, e in prossimità del 30 dicembre segno di una diminuzione percentuale dell'umidità.

Implementazioni future

In una prospettiva di sviluppo futuro del progetto, si potrebbe considerare l'implementazione di un'**alimentazione basata su due batterie AA**, fornendo un totale di 3V. Questa scelta potrebbe presentare notevoli vantaggi, in particolare per quanto riguarda la durata e l'autonomia del sistema di monitoraggio ambientale. Le batterie AA sono ampiamente disponibili, economiche e offrono una capacità energetica ragionevole. L'utilizzo di due batterie fornirebbe una tensione costante di 3V, ideale per il funzionamento stabile del microcontrollore e di altri componenti del circuito.

Una delle principali qualità delle batterie AA è la loro capacità di fornire un'alimentazione continua per lunghi periodi. Con una gestione efficiente dell'energia e la possibilità di mettere il microcontrollore in **modalità a basso consumo** durante i periodi di inattività, il sistema potrebbe garantire tempi di funzionamento prolungati che possono superare diversi mesi senza la necessità di sostituire le batterie. Questa autonomia estesa è particolarmente vantaggiosa per applicazioni di monitoraggio ambientale che richiedono una presenza costante e a lungo termine, riducendo al minimo la necessità di interventi manutentivi.

Inoltre, le batterie AA sono facilmente sostituibili, permettendo una gestione agevole delle risorse energetiche. La combinazione di una tensione adeguata, una durata prolungata e la praticità delle batterie AA rappresenterebbe un passo avanti significativo nel rendere il sistema energeticamente efficiente e pratico per applicazioni di monitoraggio ambientale a lungo termine.

Rappresentiamo in un'illustrazione, la possibile idea implementativa del circuito alimentato alle due batterie.



Per l'implementazione, abbiamo realizzato mediante **Stampante 3D**, una scocca di plastica simile a quella rappresentata nell'immagine.

Tuttavia, non siamo stati in grado di ottenere uno slot doppio per contenere le batterie, in quanto i tempi di spedizione superano quelli per la consegna del progetto.

Conclusioni

In conclusione, la realizzazione di questo progetto di monitoraggio ambientale ha coinvolto una serie di passaggi chiave che hanno contribuito a sviluppare un sistema efficiente e affidabile. Inizialmente, sono state selezionate e integrate le componenti fondamentali, tra cui il microcontrollore ESP8266, il protocollo di comunicazione MQTT e la piattaforma cloud ThingSpeak. La scelta di un cablaggio dedicato ha garantito un'implementazione ordinata e stabile del circuito.

Successivamente, è stato configurato un canale su ThingSpeak, definendo i parametri di interesse e le chiavi API necessarie per l'invio sicuro dei dati al cloud. Purtroppo, non siamo riusciti ad ottenere una versione stabile dei dati acquisiti su Cloud, abbiamo provveduto però a presentare una versione preliminare.

Nell'ottica di un possibile sviluppo futuro, si è valutata la fattibilità di alimentare il circuito con due batterie AA, riconoscendo i vantaggi in termini di durata e autonomia che questa scelta comporterebbe.

In sintesi, attraverso l'integrazione di queste fasi, il progetto è giunto a una fase quasi operativa in grado di monitorare i parametri ambientali e trasmettere i dati in modo affidabile al cloud. L'implementazione di questa soluzione non solo soddisfa gli obiettivi iniziali ma apre anche la strada a possibili miglioramenti e ampliamenti, consolidando l'efficacia del sistema nel contesto dell'Internet delle cose e del monitoraggio ambientale.