PROGETTO PCTO

Dal Sensore al Web: il Nostro Progetto IoT con Arduino e Rete Lepida

Descrizione

Lo scopo principale di questo progetto è quello di inviare dati relativi alla temperatura, umidità e stato della batteria alla rete PaIoT di Lepida basata su LoRaWAN utilizzando la scheda Arduino MKR WAN 1310 e il sensore DHT22.

Cos'è la rete PAIOT

La **Rete PAIoT di Lepida** è una rete pubblica IoT (Internet of Things) realizzata dalla società **Lepida S.c.p.A**. per la **Pubblica Amministrazione (PA)**, che consente l'integrazione di sensori distribuiti sul territorio per il monitoraggio istituzionale e l'uso pubblico.

Caratteristiche principali

1. **Raccolta e condivisione dei dati:**
    - I sensori appartenenti alla PA, ai cittadini e alle aziende raccolgono dati sul territorio.
    - La rete trasporta questi dati, rendendoli disponibili sia ai proprietari dei sensori sia alla PA per scopi istituzionali e pubblici.
2. **Obiettivi del progetto:**
    - Permettere alla PA di avere una visione completa dei dati raccolti dai sensori sul territorio.

        > Ottimizzare l'uso delle frequenze attraverso una **rete unica gestita dalla PA**, migliorando l'efficienza delle risorse.

    - Creare una **mappatura centralizzata dei sensori**, costituendo un **Catasto dei sensori** con dettagli descrittivi e informazioni sui proprietari.
    - Integrare sensori privati (cittadini e aziende) nella rete pubblica, consentendo il **ritorno dei dati ai proprietari** tramite interfacce applicative o un portale ad accesso sicuro con **SPID**.
3. **Tecnologia utilizzata:**
    - Utilizza la tecnologia **LoRaWAN**, integrando i **LoRa Gateway** nella **Rete Lepida**, una rete pubblica ad alta affidabilità e velocità (fino a 100 Gbps), basata su fibra ottica e collegamenti radio su frequenze licenziate (26 GHz).

Collegare un sensore alla rete

Per collegare un sensore alla rete PAIoT è necessario recarsi al sito https://retepaiot.it ed effettuare login con **SPID**, **CIE** o **SmartCard**

Funzionamento di LoRaWAN

Struttura

L'architettura di LoRaWAN è strutturata su tre livelli principali:

Dispositivi End-node:
- Sensori e attuatori che raccolgono o trasmettono dati (es. sensori di temperatura, contatori intelligenti).
- Comunicano via radio utilizzando la tecnologia **LoRa** (modulazione a spettro espanso).

Gateway:
- Ricevono i segnali LoRa da più dispositivi e li inoltrano al **Network Server** tramite connessione IP (tipicamente Ethernet o 4G).
- Agiscono da **bridge** tra la rete LoRa e internet.

Network Server:
- Smista i pacchetti ricevuti e gestisce il traffico, eliminando i duplicati e verificando l'autenticità.
- Invia i dati validi al **Application Server**, che li elabora e li rende disponibili agli utenti finali.

Trasmissione dei dati

Fase di join

Questa fase consente al dispositivo di autenticarsi sulla rete LoRaWAN e ottenere le chiavi di sessione necessarie per la comunicazione sicura.
Ci sono due modi principali per connettersi a qualsiasi rete LoRaWAN:

Join tramite OTAA (Over The Air Activation)**:**

Il dispositivo avvia il processo di autenticazione inviando un **Join Request** al gateway.

Join Request:

- Contiene:
  - **DevEUI:** Identificatore univoco del dispositivo.
  - **AppEUI:** Identificatore dell'applicazione.
  - **AppKey:** Serve a criptare i dati via AES-128 durante la fase di join
- Il **gateway** inoltra il pacchetto al **Network Server**.

Validazione e Risposta:

- Il Network Server verifica l'identità e genera le chiavi di sessione:
  - **AppSKey (Application Session Key):** Per la crittografia dei dati.
  - **NwkSKey (Network Session Key):** Per autenticare i pacchetti.
- Invia una **Join Accept** al dispositivo, criptata con la **AppKey**.
- La risposta contiene il **DevAddr** (identifica il dispostivo) e altri dati.

Conferma:

- Una volta ricevuto il **Join Accept**, il dispositivo è connesso e può iniziare a trasmettere dati.

Join tramite ABP (Activation By Personalization)**:**

- Le chiavi (**AppSKey** e **NwkSKey**) sono preconfigurate nel dispositivo.
- Non c'è scambio iniziale, quindi è più veloce ma meno sicuro rispetto a **OTAA** (In OTAA, le chiavi sono generate dinamicamente ad ogni join, aumentando la sicurezza).

Trasmissione dati

Dopo il join, il dispositivo inizia a trasmettere i dati raccolti (downlink, da dispostivo a server).

Struttura del Pacchetto
Ogni pacchetto dati include:

- **Intestazione (Header):** Informazioni sul tipo e l'indirizzo.
- **Payload:** I dati effettivi, crittografati con **AppSKey**.
- **MIC (Message Integrity Code):** Verifica l'integrità del pacchetto (usando **NwkSKey**).

Procedura di Trasmissione (Uplink):

1. **Preparazione del messaggio:**
    - Il dispositivo codifica i dati e aggiunge il **MIC** per garantire l'integrità.
2. **Trasmissione:**
    - Il pacchetto viene trasmesso via LoRa a uno o più gateway.
3. **Ricezione dal Gateway:**
    - Il gateway inoltra i dati al **Network Server** tramite connessione IP.
4. **Verifica:**
    - Il server controlla il **MIC** per autenticare il dispositivo.
5. **Invio ai Servizi Applicativi:**
    - I dati validati vengono inoltrati al **Application Server**.

Procedimento

Step 1: Acquisizione dell’hardware

Arduino MKR WAN 1310

> L’Arduino MKR WAN 1310 è una scheda compatta,avanzata e potente progettata per applicazioni LoRa ideale per progetti IoT che richiedono una comunicazione wireless a lunga portata e a basso consumo energetico
>
> **APPLICAZIONI PRINCIPALI**
> - Monitoraggio ambientale
> - Agricoltura intelligente

> **CARATTERISTICHE PRINCIPALI**
> - Microcontrollore:SAMD21 Cortex-M0+ a 32 bit
> - Memoria: 256 KB di flash e 32 KB di ram
> - Modulo LoRa: Murata CMWX1ZZABZ,Chip SemtechSX1276
> - Interfacce: UART,SPI,I2C,8 ingressi analogici e 7 uscite digitali
> - Alimentazione: funzionamento tramite USB
> - Ottimizzato per dispositivi a basso consumo
> - Compatibile con il protocollo LoRaWAN, ideale per trasmissioni a lunga distanza

> **VANTAGGI**
> - Facile da programmare con l’IDE di Arduino
> - Ideale per applicazione IoT
> - Consumo minore di energia
> - Compatibile con infrastrutture LoRaWAN

Sensore DHT22

> Il **DHT22** è un sensore digitale che misura temperatura e umidità con alta precisione, utilizzando un sensore capacitivo per l'umidità e un termistore per la temperatura.
>
> **ALIMENTAZIONE**
> 3,3 - 6 VCC
> **CONSUMO**
> In misura: 1,15 mA
> In standby: 50 µA

Step 2: Test dell'hardware

2.1: Comunicazione LoRa tra due dispositivi

Seguendo una guida nel sito di Arduino (https://docs.arduino.cc/tutorials/mkr-wan-1310/lora-send-and-receive/) abbiamo messo in comunicazione 2 dispositivi: uno che invia un messaggio con "Hello World!" e l'altro che lo riceve.
Con questo passaggio ci siamo assicurati che l'hardware per la comunicazione funzioni correttamente.

2.2: Trovare i dati per la connessione alla rete di Lepida

Ci sono due modi principali per connettersi a qualsiasi rete LoRaWAN, descritti in precedenza, **OTAA** e **ABP**.
Noi abbiamo deciso di utilizzare il primo metodo (OTAA), di conseguenza, come riportato precedentemente, abbiamo dovuto:

- trovare il **DevEUI** (Device Extended Unique Identifier): è un identificativo che viene assegnato a ogni dispositivo durante la fabbricazione e non può essere modificato.
- generare **AppKey**, una sequenza di 128 bit (32 caratteri esadecimali) che serve al dispositivo per generare un messaggio di join firmato da inviare al server. Il server successivamente verifica questo messaggio con l'AppKey dato in precedenza e se tutto va bene, vengono generate le chiavi di sessione.
- trovare l'**AppEUI**, un identificatore a 64 bit (8 byte) che rappresenta in modo univoco l'applicazione a cui il dispositivo appartiene.

Per trovare il **DevEUI**, abbiamo seguito una guida sul sito di Arduino (https://docs.arduino.cc/tutorials/mkr-wan-1310/the-things-network/) e abbiamo caricato uno sketch che mostra il DevEUI. Successivamente abbiamo proseguito autonomamente.
Per quanto riguarda l'**AppKey**, visto che è una chiave segreta che deve essere salvata sia nel client che nel server, l' abbiamo generata utilizzando il seguente sito: https://loratools.nl/#/keys.
Infine, per quanto concerne l'**AppEUI**, visto che non ci è stata data nessuna informazione riguardo questo dato, ne abbiamo utilizzato uno generico: una sequenza di zeri lunga 8 bytes o 16 caratteri (**0000000000000000**).

Questi dati vanno inseriti nella dashboard di Lepida

Nuovo dispositivo

Configura un nuovo dispositivo LoRa.

Nel caso in cui il modello non sia previsto nell'elenco attuale, si chiede di inviare una mail a retepaiot@lepida.it indicando marca e modello da aggiungere

Marca *

ARDUINO

Esempio: Arduino.

Modello *

MKRWAN1310

Esempio: MKRWAN1300TUCB.

SENSORE DI UMIDITÀ

SENSORE CARICA BATTERIA

SENSORE DI TEMPERATURA

Connessione *

OTAA (Over-the-Air Activation)

ABP (Activation by Personalization)

EUI *

End-device Identifier.

AppKey *

Application Key.

Latitudine *

Longitudine *

Descrizione *

Descrizione breve

2.3: Invio di dati di test

Una volta trovati tutti i dati necessari, ci siamo recati in un’area coperta dalla rete (la scuola) e abbiamo iniziato a inviare dei valori.
Abbiamo inizialmente provato a inviare dei semplici numeri ma non venivano registrati in modo corretto visto che per ogni carattere compariva un “3” davanti.

28/03/2025 09:26:01	39
28/03/2025 08:59:08	3132
28/03/2025 07:20:39	35

Come si può vedere, abbiamo provato a inviare “5”, “12”, “9”.
Quindi abbiamo compreso che fosse meglio inviare il tutto come un’ unica stringa in esadecimale, e così facendo ora i dati vengono registrati correttamente.

03/04/2025 10:33:09	7012500a
03/04/2025 10:31:33	1013500a
03/04/2025 10:29:49	7012280a
03/04/2025 10:28:10	de12280a
03/04/2025 10:28:02	d412280a
03/04/2025 10:21:31	ca12320a
03/04/2025 10:21:05	7e13280a
03/04/2025 10:19:45	f0

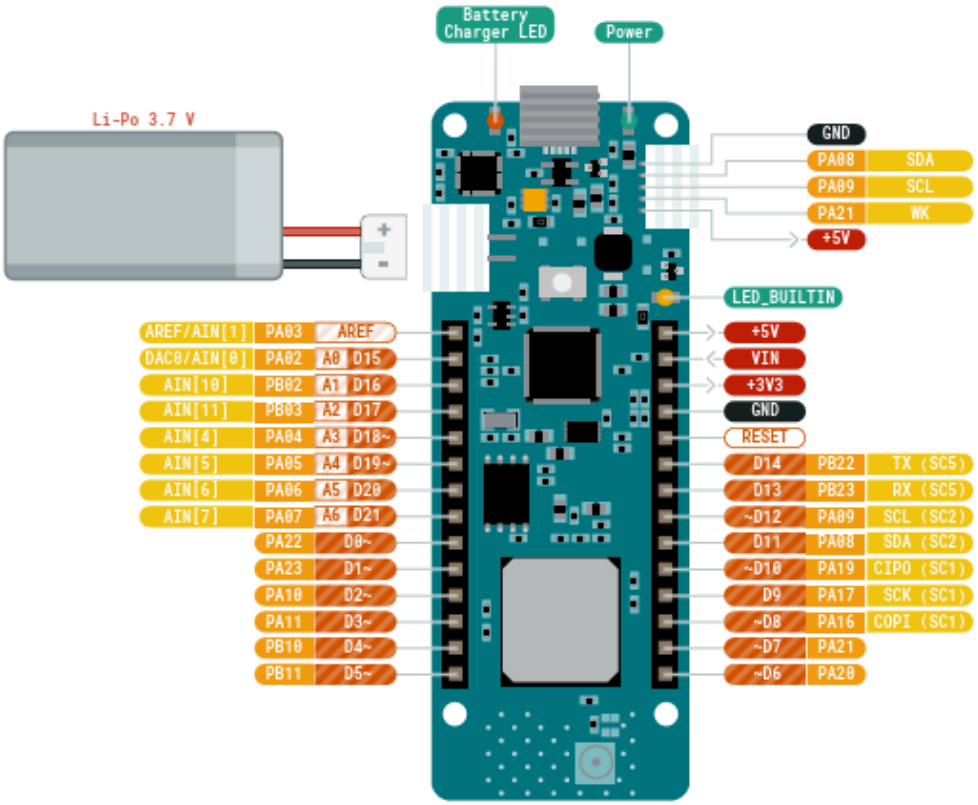
Step 3: Collegamento del sensore & invio dei dati

Una volta verificato che tutto fosse funzionante, abbiamo collegato il sensore e avviato l'invio dei dati.
Poiché gli output del sensore DHT22 sono numeri di tipo float (decimali) che occupano 4 byte, prima di inviarli a Lepida li moltiplichiamo per 100.
Questo passaggio consente di convertirli in numeri interi, mantenendo la precisione a due decimali e riducendo, così, la dimensione dei dati a soli 2 byte per valore.
In questo modo, siamo riusciti a trasmettere tutti e tre i dati (umidità, temperatura e batteria) all'interno di un payload di appena 6 byte.

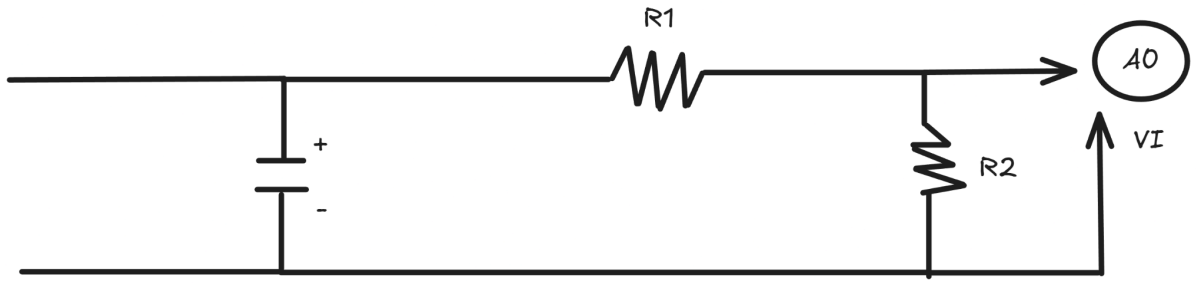
Esempio: **6b1bf3010a0a**
Questo è un esempio di stringa inviata dal nostro Arduino, e tenendo in mente che abbiamo inviato 3 dati da 2 bytes ciascuno, nel formato **umidità** - **batteria** - **temperatura**, possiamo decodificarli nel seguente modo:

- Umidità: **6b1b** → **1b6b** (ruotare la stringa) → **7019**→ **70,19**% (dividi per 100)
- Batteria: **f301** → **01f3** (ruotare la stringa) → **499** → **4.99**V (dividi per 100)
- Temperatura: **0a0a** → **0a0a** (ruotare la stringa) → **2570**-> **25,7°C** (dividi per 100)

Step 4: Collegamento della batteria



L’obiettivo iniziale era quello di collegare una batteria LiPo e leggerne la percentuale di carica. Abbiamo collegato la batteria secondo il seguente schema



Abbiamo partizionato la tensione prodotta dalla batteria e l'abbiamo inviata al pin A0, riuscendo così a leggere correttamente lo stato della batteria. Tuttavia, la durata si è rivelata estremamente limitata, tanto che la batteria non è riuscita a durare nemmeno una notte. Per questo motivo, abbiamo deciso di utilizzare una semplice pila alcalina.

Step 5: Interfaccia Web

Come ultimo passaggio, abbiamo creato un' interfaccia web che utilizza le API di Lepida per ottenere i dati registrati e mostra gli stessi su grafici.
Il sito è accessibile all'indirizzo https://pcto-frontend.vercel.app.
Codice sorgente: https://github.com/le0o0oo/pcto-frontend

Il sito è stato fatto con il framework Nuxt.
I dati inviati dalla scheda vengono reperiti nel backend del sito con la seguente funzione:

```
async function fetchData() {
  consola.info("Fetching data from Lepida API...");
  const res = await $fetch(
    `https://api.retepaiot.it/sensordata/${config.app_eui}`,
    {
      method: "POST",
      body: {
        auth_key: config.lepida_api_key,
        from: config.start_from,
      },
    }
  );

  cache = res;
  consola.success("Data fetched & cached");
}
```

Vengono poi decodificati in questo modo:

```ts
// decoder.ts
function rotateString(str: string, n: number): string {
  n = n % str.length;
  return str.slice(n) + str.slice(0, n);
}

export default (
  input: string
): { temp: number; battery: number; hum: number } => {
  let temp, hum, battery;

  let humHex = input.slice(0, 4);
  let batteryHex = input.slice(4, 8);
  let tempHex = input.slice(8, 12);

  humHex = rotateString(humHex, 2);
  tempHex = rotateString(tempHex, 2);
  batteryHex = rotateString(batteryHex, 2);

  temp = parseInt(tempHex, 16) / 100;
  hum = parseInt(humHex, 16) / 100;
  battery = parseInt(batteryHex, 16);

  return { temp, battery, hum };
};
```