Serra «smart» con Arduino MKR

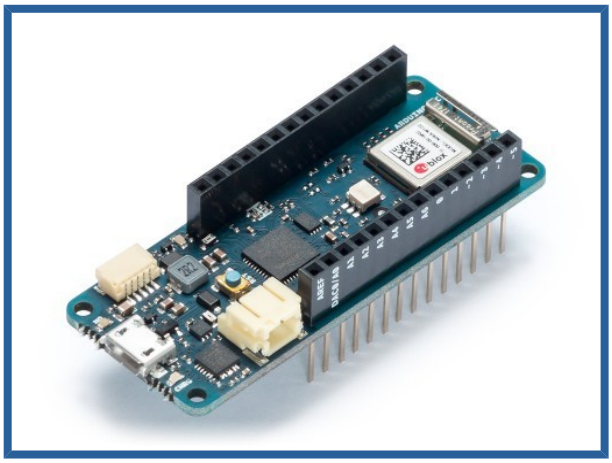
Speciale «Trento Smart City Week 2019»

Versione 1.0
09/03/2020

0. Introduzione

In questo tutorial utilizzeremo le schede della famiglia **Arduino MKR** (che si pronuncia “maker”) per costruire una serra *smart*, pronta per l’***Internet delle cose*** (***Internet of Things***, in breve: ***IoT***): la MKR WiFi 1010 che assicura la connettività (fondamentale, per poter parlare di *IoT*) e -opzionalmente- lo shield MKR ENV che mette a disposizione tutta una gamma di sensori utili per monitorare le caratteristiche ambientali (temperatura, umidità, pressione, luminosità presente, UVA e UVB, ed indice UV).

Arduino MKR WiFi 1010



Shield Arduino MKR ENV

N.B. Utilizzando queste schede, dobbiamo fare bene attenzione a due cose: la prima è l’estrema **fragilità dell’antenna** del modulo WiFi (sulla scheda si trova all’estremità opposta rispetto alla porta micro USB e si può notare che in qualche modo è stata protetta con una goccia di colla a caldo).
L’altra cosa è la tensione di lavoro delle porte delle schede MKR: contrariamente ai “soliti” Arduini (che lavorano sui 5 Volt), queste schede utilizzano una tensione massima pari a 3.3 Volt. Ciò significa che possono pilotare in uscita altre porte a 5 Volt o i canonici LED, ma **non possono accettare in ingresso segnali di 5 Volt**, pena la bruciatura della porta.

N.B.2 Puoi trovare questo tutorial, assieme a tutti gli sketch di Arduino, sul nostro sito all’indirizzo:

coderdojotrento.it/arduino4

Questi i passi che seguiremo per completare il nostro progetto:

- creazione di un account sul sito di Arduino (vedi capitoli 1 e 2),
- registrazione dell'hardware e gestione di connettività e sicurezza (cap. 3 e 4),
- descrizione della nostra "cosa" (cap. 5),
- implementazione (codifica) dell'oggetto (cap. 6),
- due passi nella "nuvola" (cap. 7),
- utilizzo dello shield ENV con i sensori ambientali (cap. 8).

1. Creazione account ed installazione agent

Colleghiamoci al sito **`create.arduino.cc`** e **creiamo un nostro account**: qui non solo potremo scrivere i nostri sketch per Arduino, ma potremo anche realizzare i nostri progetti *IoT* (descrivendo i componenti che ne fanno parte e gestendo i cruscotti che ci permetteranno di interagire con essi via web).

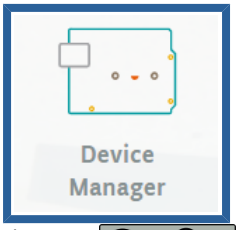
Il **Web Editor** che troviamo alla pagina **`create.arduino.cc/editor`** (al quale possiamo **accedere utilizzando l'icona qui a destra**) può sostituire del tutto l'IDE Arduino che di solito installiamo sui nostri computer: i suoi vantaggi sono molti (file sempre automaticamente salvati e disponibili, costantemente aggiornata con le ultime versioni delle librerie, possibilità di condividere i progetti nella community di Arduino), ma non è in grado di caricare i programmi sulle schede collegate al nostro PC; per questo bisogna **installare il plugin "Arduino Create Agent"** come suggerito dal pannello giallo che ci avverte appunto che manca la connessione con il plugin.
N.B. Dopo l'installazione può essere necessario verificare che antivirus e firewall del computer non blocchino la comunicazione tra l'Agent e l'editor nel browser.

2. Utilizzo di Arduino offline, solo per fare una prova

La 1010 è una scheda con funzionalità WiFi, ma nessuno ci impedisce di utilizzarla offline, come fosse un caro vecchio Arduino UNO: **colleghiamo la MKR WiFi 1010 ad una porta USB del computer** ed attendiamo che l'Agent la metta in comunicazione con il Web Editor.

Appena stabilito il contatto, il Web Editor riconoscerà in automatico il tipo di scheda e la porta alla quale è collegata. Siamo pronti per far girare l' "Hello, world!" di Arduino: dal menù di sinistra **navighiamo le voci** "Examples" → "Build in" → "01.BASICS" → "Blink" e carichiamo lo sketch sulla nostra scheda utilizzando il **tasto di "Upload"** (icona qui a destra).
Se il caricamento va a buon fine, il piccolo LED montato sulle scheda comincerà a lampeggiare: funziona tutto… forse! A dire il vero lo sketch "blink" è pre-caricato su ogni scheda, quindi per assicurarsi che funzioni tutto davvero, cambiamo il valore dell'attesa (numero di millisecondi nella chiamata a `delay()`), colleghiamo un secondo LED tra GND ed un pin digitale diverso da quello proposto (il LED_BUILTIN corrisponde al pin 6), oppure aggiungiamo anche qualche tracciamento con la classe Serial (che potremo poi controllare dal menù "Monitor" dell'interfaccia web).

3. Arduino IoT – Devices: descrizione dell'hardware connesso

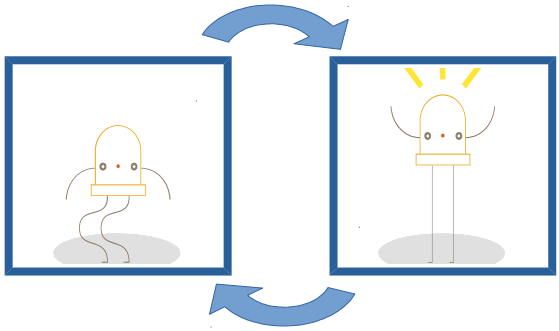


Siamo pronti per cominciare il nostro progetto da *Internet of Things* e per prima cosa dobbiamo **descrivere la scheda fisica** che ci permetterà di connetterci alla nuvola. Dalla schermata principale del sito Arduino Create, colleghiamoci alla pagina **`create.arduino.cc/devices`** utilizzando l'icona qui a fianco e

prepariamoci a creare un “nuovo device”:
a) scegliamo la scheda in nostro possesso (Arduino MKR WiFi 1010);
b) se già non è attivo, facciamo partire il plugin “Arduino Create Agent”;
c) colleghiamo la scheda alla porta USB del PC;
d) diamo un nome alla scheda (ad esempio “MKR1010_VERDE”);
e) completiamo il passo di configurazione della scheda per abilitare il chip del WiFi alla connessione in rete (questo passo può impiegare qualche minuto). Siamo quindi pronti per la prima prova di utilizzo “nel cloud”.

4. Arduino IoT – Devices: connessione alla rete e test nel cloud

Il wizard ci suggerisce di caricare un progetto che possiamo studiare in seguito ma che -lo notiamo subito- è costituito da uno sketch di Arduino (mostrato nel tab “WiFi_Cloud_Blink.ino”) e da un altro **tab dal nome “Secret”**, abbastanza esplicito: qui dobbiamo infatti immettere il **nome e la password della rete WiFi alla quale il nostro Arduino si aggancerà**.



Se il programma caricato riuscirà a connettersi alla rete, la procedura di installazione guidata ci proporrà una pagina grazie alla quale possiamo **inviare dei comandi alla scheda MKR**: possiamo scrivere “ON” oppure “OFF” nella casella di testo ed inviarli alla scheda, oppure possiamo cliccare sul disegno del LED. Se tutto è in ordine, **il piccolo LED sulla scheda** si accenderà e spegnerà al nostro comando, e l’icona mostrata rispecchierà lo stato di accensione del LED on board.

N.B. La scheda Arduino è ancora attaccata al nostro PC tramite il cavo USB, è vero; i comandi che sottomettiamo dall’interfaccia web, però, **seguono tutt’altro giro**: il nostro browser interagisce col server `create.arduino.cc`, il quale comunica con `mqtts-sa.iot.arduino.cc` (un altro server che svolge il ruolo di broker MQTT, un protocollo di comunicazione usato nell’IoT), il quale a sua volta dispaccia i comandi ON/OFF al nostro Arduino collegato sul WiFi prima configurato.
Staccando il cavo ed alimentando la scheda con una batteria, avremmo lo stesso risultato: insomma, il nostro primo esperimento nella nuvola funziona senza trucchi e senza inganni[1]!

Terminata la fase di setup ed il test, troveremo la scheda così predisposta tra i nostri device.

5. Arduino IoT – Things: modellazione della “cosa”

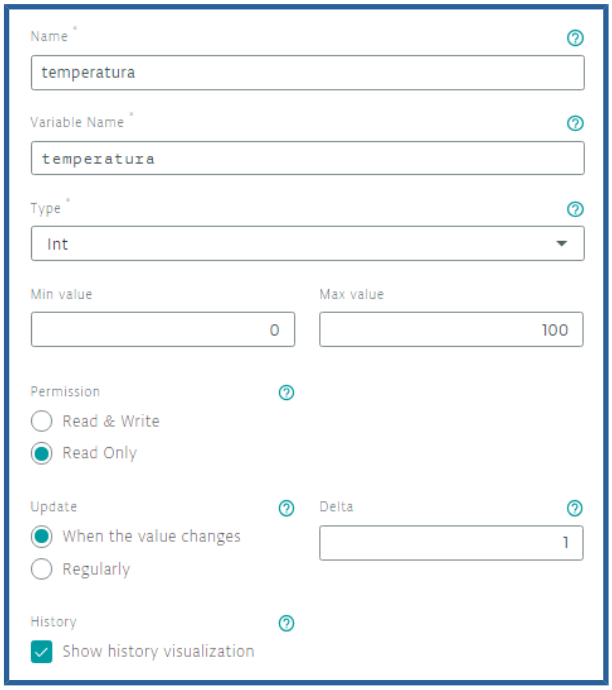
Una volta sistemati i device, dobbiamo modellare la “thing” attivando l’icona qui a destra (al link **`create.arduino.cc/iot/things`**). Questa “cosa” è la rappresentazione logica dell’hardware sottostante: per poter far *entrare nella nuvola* il nostro **oggetto connesso**, pensiamo a quali sono **gli attributi che lo caratterizzano**. Se parliamo di una serra, possiamo ad esempio individuare la temperatura dell’ambiente e lo stato di accensione di una lampada ad infrarossi da impiegare per scaldare le nostre piante (e anche se una lampada ad infrarossi non ce l’abbiamo, un LED sarà perfetto per simularla).

1 Una volta configurato il chip del WiFi, infatti, possiamo programmare il nostro Arduino anche senza appoggiarci all’editor web: possiamo scaricare lo sketch ed aprirlo con Arduino IDE. Per la corretta compilazione/esecuzione è necessario includere le librerie ArduinoIoTCloud, ArduinoMqttClient, ArduinoIoTCloudBearSSL, ArduinoCloudThing, ArduinoECCX08 e RTCZero. Trovi il sorgente a questo link.

Utilizziamo il tasto qui a destra per creare la nostra nuova "cosa": possiamo dare un nome alla thing (ad esempio "SERRA_IOT") ed associarla alla scheda hardware prima configurata ("MKR1010_VERDE").

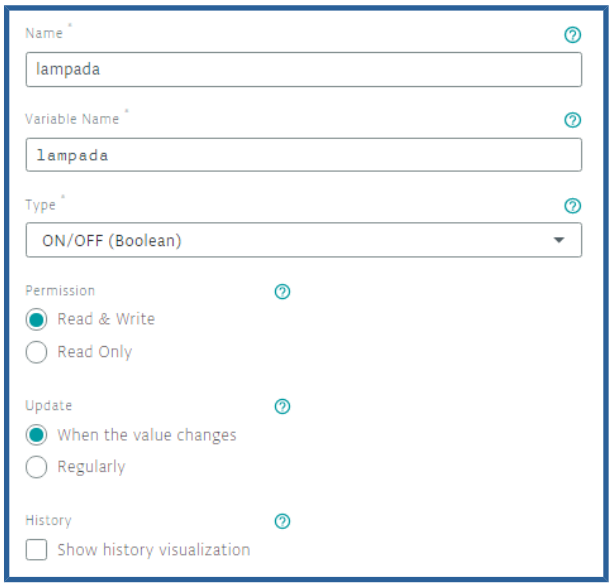
Nella prossima schermata lavoriamo sul **tab "Properties"** e definiamo due nuove proprietà descrivendo i due attributi individuati sopra:

- nome proprietà: **temperatura**
  - nome variabile: temperatura
  - tipo: intero
  - valore minimo: 0
  - valore massimo: 100
  - permessi: "Sola lettura" (RO = "Read only")
  - aggiornamento: "Quando il valore cambia"
  - delta: 1
  - visualizza storia: sì



La spiegazione è semplice: ci serve un attributo di tipo intero (assumiamo che il sensore non fornisca misure con i decimali), che varia al massimo tra 0 e 100 gradi; il sensore si limiterà a leggere la temperatura (non potremo deciderla noi, in sostanza) e chiediamo alla nostra "cosa" di informarci del valore misurato solo quando vengono rilevate delle variazioni di almeno un grado (il "delta").

- nome proprietà: **lampada**
  - nome variabile: lampada
  - tipo: ACCESO/SPENTO (booleano)
  - permessi: "Lettura e scrittura" (R&W = "Read & write")
  - aggiornamento: "Quando il valore cambia"
  - visualizza storia: no



A differenza del primo attributo, questo può solo assumere i valori ACCESO/SPENTO (quindi sarà rappresentato da una variabile booleana vero/falso) ed è R&W perché -oltre a consultarne il valore- possiamo anche intervenire per cambiarlo.

Diamo ora un'occhiata al **tab "Dashboard"**: qui troviamo un semplice cruscotto nel quale sono riportati degli elementi grafici che corrispondono alle properties che abbiamo appena modellato (in base al tipo scelto, avremo modi diversi per rappresentare le misure corrispondenti: per i booleani verranno usati degli interruttori ON/OFF, per i parametri reali dei grafici o dei visualizzatori a lancetta, etc.).

6. Arduino IoT – Things: implementazione (codifica) della "cosa"

È il momento di implementare il comportamento della nostra "cosa": una volta descritte le proprietà che verranno esposte nella nuvola, bisogna **decidere come il nostro hardware fornirà queste misure**. Lo realizziamo scrivendo un po' di codice!

Cliccando sul tasto “Edit sketch” si viene indirizzati sul Web Editor: qui troveremo un nuovo **progetto generato automaticamente** che ha lo stesso nome della nostra “cosa”, abbinato alla data odierna (ad esempio: “SERRA_IOT_aug20a”).

Al suo interno troviamo:

- **un ReadMe.doc** nel quale possiamo descrivere il progetto e che potrebbe anche servire se decidessimo di pubblicare il nostro progetto nella community di Arduino;
- **il tab “Secret”** del quale abbiamo già fatto conoscenza durante la configurazione ed il test: sappiamo già che qui dovremo indicare la rete WiFi e la password che la nostra “cosa” utilizzerà per connettersi alla nuvola;
- **un file thingProperties.h** che descrive le variabili usate per modellare le nostre properties (non abbiamo esigenza di modificarlo, ma possiamo aprirlo per vedere che contiene alcune istruzioni necessarie alla connessione alla nuvola, oltre alla definizione delle variabili e dei metodi che saranno il “corpo” della nostra thing:
  - `void onLampadaChange();`
  - `bool lampada;`
  - `int temperatura;`
- **uno sketch di Arduino (il file .ino)** che contiene il codice della nostra “cosa”.

Quest’ultimo è il file che andremo a modificare: se scorriamo lo sketch principale del progetto vediamo l’istruzione `#include` che incorpora thingProperties.h, il metodo di `setup()` nel quale vengono preparate le properties e viene effettuata la connessione alla nuvola, quindi i due unici metodi sui quali dobbiamo intervenire:

```
void loop() {
  ArduinoCloud.update();
  // Your code here

}

void onLampadaChange() {
  // Do something
}
```

Ma cosa ci dicono i suggerimenti contenuti nei commenti? All’interno del metodo `loop()` -che viene costantemente invocato in maniera ciclica dopo che il nostro Arduino sarà avviato ed inizializzato- subito sotto la scritta “`// Your code here`” aggiungeremo i comandi per **valorizzare le due variabili `temperatura` e `lampada` che poi verranno *automagicamente* inviate nella nuvola** dal framework di Arduino.
Questa rappresenta la parte di ***lettura* delle properties**: l’ambiente (oppure il nostro codice) cambia le proprietà e Arduino propaga nella nuvola grazie al metodo `ArduinoCloud.update()`.

Per quanto riguarda invece la risposta del nostro oggetto ad una variazione che arriva dalla nuvola, ovvero alla ***scrittura* delle properties** che abbiamo modellato come di tipo “read & write”, dobbiamo agire sul metodo `void onLampadaChange()`: quando qualcuno o qualcosa nella nuvola modifica la variabile lampada, `ArduinoCloud` invoca in automatico il metodo `onLampadaChange()` e noi dobbiamo scrivere di conseguenza il codice per rispondere a questa variazione.

Nello sketch qui riportato, abbiamo una continua variazione della `temperatura` che parte dal valore 30 (vedi riga 14) e quindi aumenta (o diminuisce) dell'ammontare `variazioneTemperatura` (riga 23); questa variazione avviene soltanto quando sono passati almeno `INTERVALLO` millisecondi dall'ultimo aggiornamento (riga 22). Per gestire la tempistica vengono effettuati dei controlli sul tempo trascorso con la funzione `millis()` ed inoltre -per evitare di inondare la nuvola con un aggiornamento dei dati troppo frequente- vien introdotto un ritardo forzato di qualche decimo di secondo (riga 32).
Inoltre possiamo ottenere una rappresentazione visiva dello stato di accensione/spegnimento della nostra lampada della serra controllando in maniera opportuna il LED on board del nostro Arduino MKR WiFi 1010: prepariamo il pin come canale di output (riga 16 nel `setup()`) e nel metodo che risponde alle variazioni dell'attributo, andremo ad accendere o spegnere il LED in base al valore assunto in quell'istante (riga 37 del metodo `onLampadaChange()`).
Per evitare che la temperatura aumenti indefinitamente, effettuiamo un controllo che inverte la rampa (riga 26) se la temperatura supera i 45 gradi oppure scende sotto i 10 (riga 25). Quando avvengono questi superamenti di soglia, imponiamo anche un cambio di stato nella lampada (riga 27) ed invochiamo di conseguenza il metodo che aggiorna il LED (riga 28).

```
1  #include "thingProperties.h"
2  unsigned long tsUltimoAggiornamento;       // blocco A
3  #define INTERVALLO 1000
4  int variazioneTemperatura = 2;
5
6  void setup() {
7    Serial.begin(9600);
8    delay(1500);
9    initProperties();
10   ArduinoCloud.begin(ArduinoIoTPreferredConnection);
11   setDebugMessageLevel(2);
12   ArduinoCloud.printDebugInfo();
13
14   temperatura = 30;
15   tsUltimoAggiornamento = millis();         // blocco B
16   pinMode(LED_BUILTIN, OUTPUT);
17 }
18
19 void loop() {
20   ArduinoCloud.update();
21   // Your code here
22   if (millis() - tsUltimoAggiornamento > INTERVALLO) {
23     temperatura += variazioneTemperatura;
24     tsUltimoAggiornamento = millis();
25     if ((temperatura > 45) || (temperatura < 10)) {
26       variazioneTemperatura *= -1;
27       lampada = HIGH - lampada;
28       onLampadaChange();                    // blocco C
29     }
30   }
31
32   delay(200);
33 }
34
35 void onLampadaChange() {
36   // Do something
37   digitalWrite(LED_BUILTIN, lampada);       // blocco D
38   if (lampada == HIGH)
39     Serial.println("La lampada e' accesa.");
40   else
41     Serial.println("La lampada e' spenta.");
42 }
```

Le parti evidenziate in giallo sono quelle che dobbiamo aggiungere allo sketch generato automaticamente dal sito:

A) c'è il blocco iniziale delle definizioni delle variabili di supporto (righe 2 ... 4);

B) l'inizializzazione delle stesse e la predisposizione del pin 6 come canale di output (nel `setup()`: righe 14 … 16);

C) nel `loop()` c'è il blocco di codice che -trascorso il lasso di tempo di `INTERVALLO` millisecondi- cambia la temperatura, controlla i superamenti di soglia ed eventualmente inverte la rampa e cambia lo stato della lampada; infine rallenta un poco l'esecuzione del ciclo (in `loop()`: righe 22 ... 32);

D) nel metodo che reagisce al cambio del valore di lampada, accendiamo o spegniamo il LED e tracciamo un messaggio sulla seriale (in `onLampadaChange()`: righe 37 ... 41).

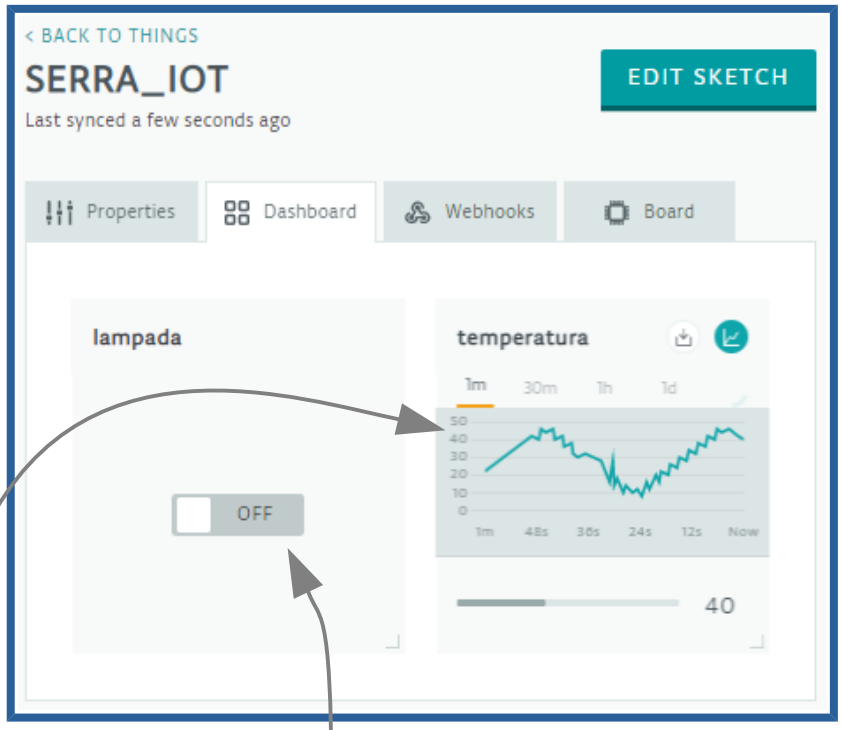
Siamo pronti: carichiamo lo sketch sulla nostra scheda con il tasto a forma di freccia!

7. Arduino IoT – Cloud: ma cos'è questa "nuvola", alla fine?

Nelle pagine precedenti abbiamo usato espressioni come "la nuvola scrive dei valori" o "Arduino invia delle misure alla nuvola", ma cosa sarebbe -in fondo in fondo- questa "nuvola"?
Nulla di magico, già prima abbiamo fatto conoscenza con almeno una delle sue parti, quella terminale: si tratta del **cruscotto con i widget che ci mostrano i valori della nostra "thing"**.

Dalla pagina del Web Editor di Arduino, possiamo tornare alla pagina della nostra "cosa" cliccando il tasto qui a destra; quindi accediamo al cruscotto selezionando **il tab "Dashboard"**.

La **dashboard** è un'applicazione web che visualizza i dati raccolti dalla nostra scheda Arduino: è integrata all'interno del sito `create.arduino.cc` ed è configurata per "**stare in ascolto**" per eventuali dati prodotti dalla nostra serra: grazie ad un meccanismo di "**iscrizione ad una topic**" (l'argomento in questione sono appunto i dati prodotti dalla nostra "thing"), viene informata dall'infrastruttura di Arduino ogni volta che un nuovo dato è resto disponibile dalla scheda. Il dato viene quindi raccolto e mostrato grazie ai **widget grafici** presenti nel cruscotto.



Con il programma della pagina precedente, vediamo un cruscotto simile a quello qui a destra: la temperatura ha un grafico a doppia rampa tra 10 e 45 gradi e ad ogni inversione della pendenza, cambia lo stato ON/OFF della lampada.

E fin qui stiamo solo **leggendo dei parametri**. Ma dato che abbiamo configurato `lampada` come un **attributo di tipo "read & write" possiamo anche cambiarne il valore** agendo sul componente a forma di interruttore.

8. Utilizzo dello Shield Arduino MKR ENV: misure ambientali reali

Nello sketch precedente abbiamo simulato programmaticamente la variazione della temperatura ma -se disponiamo di uno shield Arduino MKR ENV- possiamo **misurare la temperatura ambientale reale**. Stacchiamo quindi il cavo USB, eventuali componenti presenti sui pin della MKR WiFi 1010, applichiamo lo shield ENV, aggiungiamo un LED sul pin 2 e ricolleghiamo le schede al PC tramite il cavo USB.

Una volta predisposto l'hardware, dobbiamo **adeguare il software** per effettuare la lettura dei parametri e la condivisione con la nuvola. Possiamo partire dal progetto già creato e modificare lo sketch principale; in alternativa possiamo creare un nuovo progetto e ricopiare il file thingProperties.h e replicare i dati per la connessione al WiFi presenti nel tab "Secret" (infatti **l'interfaccia** della nostra "cosa" rimane invariata, mentre cambia il modo in cui produce i dati).

Lo script finale sarà come quello riportato nella pagina seguente; anche stavolta abbiamo evidenziato in giallo le novità.

```
1  #include <Arduino_MKRENV.h>
2  #include "thingProperties.h"
3
4  #define PIN_LAMPADA 2
5
6  void setup() {
7    Serial.begin(9600);
8    delay(1500);
9    initProperties();
10   ArduinoCloud.begin(ArduinoIoTPreferredConnection);
11   setDebugMessageLevel(2);
12   ArduinoCloud.printDebugInfo();
13
14   if (!ENV.begin()) {
15     Serial.println("Errore nello shield MKR ENV!");
16     while (1);
17   }
18   Serial.println("pinMode");
19   pinMode(PIN_LAMPADA, OUTPUT);
20 }
21
22 void loop() {
23   ArduinoCloud.update();
24   // Your code here
25
26   float temperature = ENV.readTemperature();
27   float humidity    = ENV.readHumidity();
28   float pressure    = ENV.readPressure();
29   float illuminance = ENV.readIlluminance();
30   float uva         = ENV.readUVA();
31   float uvb         = ENV.readUVB();
32   float uvIndex     = ENV.readUVIndex();
33
34   temperatura = int(temperature);
35   Serial.print("temperatura = ");
36   Serial.println(temperatura);
37
38   delay(200);
39 }
40
41 void onLampadaChange() {
42   // Do something
43   digitalWrite(PIN_LAMPADA, lampada);
44   if (lampada == HIGH)
45     Serial.println("La lampada e' accesa.");
46   else
47     Serial.println("La lampada e' spenta.");
48 }
```

Per prima cosa includiamo la **libreria** che permette di accedere alle **funzionalità dello shield MKR ENV** (riga 1) e definiamo il nuovo pin con attaccato il LED (riga 4).

Inizializziamo l'oggetto `ENV` e tracciamo eventuali errori (righe 14 … 17).
Sempre nel `setup()` configuriamo il `PIN_LAMPADA` come pin di output (righe 18 … 19)

Nel metodo `loop()` effettuiamo la **lettura di tutti i parametri che lo shield è in grado di misurare** e li memorizziamo in opportune variabili (righe 26 … 32).

Di queste utilizziamo in realtà solo la temperatura: la convertiamo ad intero e la **associamo alla property omonima**; quindi tracciamo il valore in console (righe 34 … 36).

Il metodo `onLampadaChange()` è quasi invariato: l'unica differenza è che lavoriamo non più su `LED_BUILTIN` ma sul pin `PIN_LAMPADA` (riga 43).

Come prima **carichiamo lo sketch** sulla scheda **utilizzando il tasto a forma di freccia**; controlliamo che Arduino stia lavorando come atteso attivando **il menù "Monitor"** (qui ritroviamo tutti i tracciamenti effettuati nel codice grazie alla classe `Serial`).

Attivando quindi il tasto GO TO IOT CLOUD possiamo tornare sul cruscotto per vedere l'andamento della temperatura reale e per accendere/spegnere la lampada (implementata fisicamente dal LED collegato al pin 2). **La nostra serra *smart* è infine *up & running*!**

Note e ringraziamenti

Questo tutorial:
* fa parte delle "risorse" di CoderDojo Trento
* è stato scritto con il supporto di CoderDolomiti

Si ringrazia per l'ospitalità

Puoi trovare il codice descritto nel tutorial **coderdojotrento.it/arduino4**