

# Applicazione di integrazioni IoT ad un sistema di monitoraggio della qualità dell'aria

Andrea Cantarutti (141808)  
Lorenzo Bellina (142544)

20 ottobre 2021

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Il sensore VINDRIKTNING di Ikea</b>	<b>4</b>
<b>3</b>	<b>Obiettivi e Architettura del sistema</b>	<b>4</b>
3.1	Caratterizzazione dei requisiti . . . . .	4
3.2	Definizione dei principali servizi . . . . .	4
<b>4</b>	<b>Modifiche e personalizzazioni apportate a VINDRIKTNING</b>	<b>5</b>
4.1	Obiettivi . . . . .	5
4.2	Personalizzazione dell'hardware . . . . .	5
4.3	Costo di realizzazione . . . . .	7
4.4	Implementazione di un firmware ad-hoc . . . . .	7
4.4.1	Decodifica del payload . . . . .	8
4.4.2	Salvataggio e recupero dei parametri di configurazione . . . . .	8
4.4.3	Aggiornamento del firmware da remoto . . . . .	8
4.4.4	Configurazione di parametri personalizzati . . . . .	9
4.4.5	Comunicazione con il Broker MQTT . . . . .	10
<b>5</b>	<b>Definizione del Broker MQTT</b>	<b>11</b>
5.1	Containerizzazione di Eclipse Mosquitto . . . . .	11
<b>6</b>	<b>Ricezione dei dati</b>	<b>12</b>
6.1	Engine . . . . .	12
6.2	Containerizzazione . . . . .	12
6.3	Ricezione dei dati . . . . .	13
6.4	Trasmissione dei dati ricevuti . . . . .	13
<b>7</b>	<b>Definizione di un database per lo storage dei dati</b>	<b>14</b>
7.1	Scelta del DBMS . . . . .	14
7.2	Containerizzazione . . . . .	14

# 1 Introduzione

Il seguente elaborato espone lo sviluppo di un'**integrazione IoT** atta a conferire ad un preesistente strumento per la rilevazione della qualità dell'aria caratteristiche “smart”, prevalentemente rivolte allo storage dei dati raccolti, al loro monitoraggio e all'interazione con l'utilizzatore. Il progetto si basa su quanto è stato osservato dallo sviluppatore Sören Beye (@Hypfer) che, a seguito di un'attività di *reverse engineering*, ha descritto un procedimento per l'installazione di un modulo ESP8266 all'interno del rilevatore originale, permettendo la raccolta e il processing dei dati rilevati senza alterare le funzionalità di base del sistema.

## 2 Il sensore VINDRIKTNING di Ikea

Lo strumento di rilevazione di qualità dell'aria utilizzato è un articolo commercializzato da Ikea sotto il nome di **VINDRIKTNING**. Quest'ultimo è in grado di rilevare la quantità di polveri sottili presenti in ambienti chiusi o all'aperto (se contenuti) in base alla classificazione PM 2.5. Sulla base di specifici threshold riportati nel manuale d'uso del sensore, il valore assoluto rilevato permette, rispettivamente, l'accensione di:

- Una luce a led di colore verde atta ad indicare un buon livello di qualità dell'aria
- Una luce a led di colore giallo atta ad indicare un degradamento della qualità dell'aria
- Una luce a led di colore rosso atta ad indicare un pessimo livello di qualità dell'aria.



Il sensore è costituito da un parallelepipedo in **ABS** di dimensioni pari a 6x6x9 cm. Al suo interno contiene:

- Un sensore **Cubic PM1006**
- Un microcontrollore che si occupa dell'accensione dei led sulla base dei dati rilevati dal sensore
- Una ventola azionata al fine di favorire il ricircolo dell'aria in prossimità del sensore

VINDRIKTNING non dispone, tuttavia, di ulteriori funzionalità, assestandosi in una fascia di prezzo inferiore a 10€ e venendo spesso proposto a supporto del purificatore d'aria FÖRNUFTIG.

## 3 Obiettivi e Architettura del sistema

### 3.1 Caratterizzazione dei requisiti

In seguito all'acquisto di due unità VINDRIKTNING e ad un breve periodo di utilizzo, sulla base delle necessità individuate sono stati descritti i seguenti requisiti:

- Possibilità di osservare e analizzare l'andamento della qualità dell'aria in un determinato lasso di tempo
- Possibilità di aggregare i dati provenienti da più sensori collocati in diverse stanze e/o diverse abitazioni
- Facoltà di interrogare i sensori da remoto, ricevendo notifiche nel caso del superamento dei threshold specificati.

### 3.2 Definizione dei principali servizi

Al fine di poter attuare gli obiettivi preposti, sono stati delineati i principali servizi necessari all'implementazione di un sistema di supporto ad un **flusso di dati** generato da **più sensori** connessi alla stessa **rete locale**. Si rende, di conseguenza, necessario lo sviluppo di:

- Un **firmware personalizzato** in grado di permettere ai sensori di comunicare via rete le rilevazioni effettuate
- Un servizio per la **ricezione centralizzata dei dati**
- Un **database** adibito allo storage e all'interrogazione dei dati raccolti
- Un applicativo rivolto alla **fruizione** dei dati raccolti e alla **configurazione** del sistema
- Un servizio per la **notifica di uno o più utenti** in caso di cambiamenti notevoli.

Al fine di favorire in partenza lo sviluppo **indipendente** di ognuno dei servizi sopracitati, è stata adottata una strategia implementativa basata sull'organizzazione e la coordinazione di molteplici container. Tramite quest'ultimi, infatti, le risorse possono essere isolate e i processi avviati e gestiti separatamente. Personalizzazioni e modifiche possono, inoltre, essere apportate senza compromettere il funzionamento complessivo del sistema.

Si descrivono, di seguito, le tecnologie adottate e le implementazioni svolte al fine di attuare l'architettura descritta.

## 4 Modifiche e personalizzazioni apportate a VINDRIKTNING

### 4.1 Obiettivi

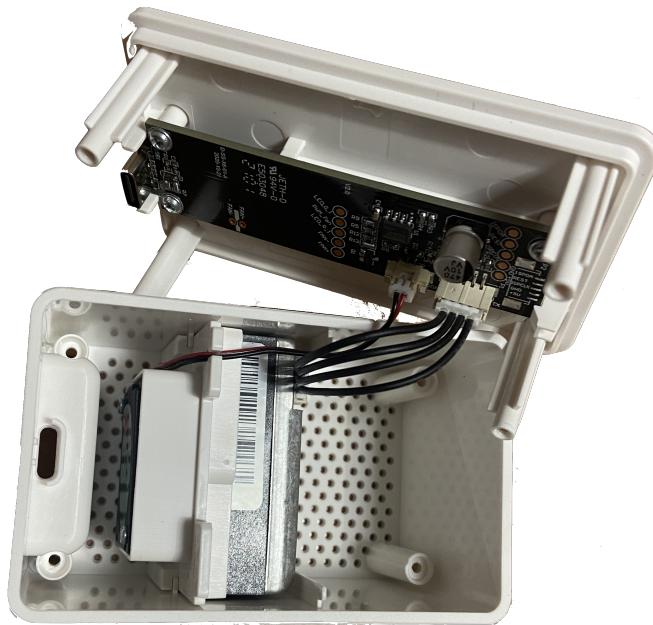
Al fine di permettere a VINDRKTNING una regolare comunicazione via rete della qualità dell'aria rilevata, è stata adottata una strategia basata su **protocollo MQTT**. In questo modo, ogni sensore connesso alla rete comunica, in qualità di **publisher**, aggiornamenti costanti ad un **broker** appositamente predisposto. Tale soluzione permette, inoltre, l'introduzione di nuovi sensori senza richiedere specifiche configurazioni e/o il riavvio del sistema.

### 4.2 Personalizzazione dell'hardware

Come precedentemente specificato, le modifiche apportate all'hardware del sensore si basano sull'attività di reverse engineering svolta dallo sviluppatore Sören Beye (@Hypfer) e accuratamente documentata su GitHub.

Una volta aperto il contenitore di VINDRIKTNING svitando le quattro viti che lo mantengono chiuso, è immediatamente visibile un microcontrollore adibito all'accensione dei led, al quale sono connessi, per mezzo di appositi connettori:

- Il sensore **Cubic PM1006** (la cui rilevazione viene letta tramite protocollo seriale)
- La sottostante ventola per il ricircolo dell'aria.



Si osserva, inoltre, come la breakout board del microcontrollore presenti svariati pin inutilizzati, fra cui:

- +5V e GND (passthru per l'alimentazione ricevuta tramite cavo USB)
- ISPDA e ISPCLK (che forniscono una connessione ai pin SCL e SDA per la comunicazione tramite protocollo I2C)
- REST (che fornisce un test point per il pin seriale RX)
- LED\_G\_1 e LED\_R\_1 (che forniscono un punto d'accesso per la comunicazione con led appositi)
- PWM\_Fan, FAN- e FAN+ (che permettono l'alimentazione e il controllo della velocità della ventola tramite segnali PWM)



Risulta, di conseguenza, possibile la connessione di un'unità esterna che, una volta saldata ai pin di alimentazione (+5V e GND) e al test point seriale REST, permette l'acquisizione del valore rilevato dal sensore **PM1006** e letto dal microcontrollore originale. In particolare, è stata selezionata un'unità **ESP8266** (nello specifico, un clone del **D1 Mini by Wemos** prodotto da AZDelivery), che in dimensioni estremamente ridotte fornisce:

- La possibilità di essere alimentata a 5V grazie al regolatore di tensione built-in (e quindi di ricevere la corrente di alimentazione direttamente dalla porta USB di VINDRIKTNING)
- La connettività via Wi-Fi in modalità Access Point e Station Mode, entrambe necessarie allo sviluppo previsto.
- La possibilità di essere programmato tramite il framework **Arduino**, con il conseguente accesso alla moltitudine di librerie disponibili in rete

Cavi dupont appositamente modificati sono stati saldati ai punti di accesso citati in precedenza, ottenendo il seguente risultato.



Infine, i seguenti pin del modulo D1 mini sono stati impiegati per effettuare la connessione:

D1 Mini	Punto di Accesso
+5V	+5V
GND	GND
D2	REST

VINDRIKTNING permette, infine, l'alloggiamento del modulo al suo interno, grazie alla moltitudine di spazio disponibile.



### 4.3 Costo di realizzazione

Le modifiche apportate hanno coinvolto l'utilizzo dei seguenti materiali:

- Unità VINDRIKTNING originale
- Cavi dupont
- Cacciavite di tipo PH0 per l'apertura del vano posteriore
- Strumentazione per la saldatura
- Modulo ESP8266 (D1 Mini by Wemos)

Al netto dell'attrezzatura già in possesso, il costo necessario all'apporto delle modifiche sopracitate viene di seguito descritto:

Componente	Costo Individuale	Quantità acquistate
VINDRIKTNING	9,95 €	2
D1 Mini	4,00 €	2
Cavi Dupont	3,00 €	1

Il costo coinvolto nella personalizzazione di una singola unità VINDRIKTNING corrisponde, quindi, alla cifra di **16,95 €**. La personalizzazione di due unità richiede, invece, una spesa complessiva pari a **30,90 €**.

### 4.4 Implementazione di un firmware ad-hoc

A seguito dell'installazione del modulo D1 Mini è stato sviluppato un firmware parzialmente ispirato a quello proposto da Sören Beye, con l'obiettivo di fornire le seguenti funzionalità:

- Lettura e decodifica del payload inviato dal sensore sulla porta seriale
- Semplice procedura per la configurazione e la connessione di VINDRIKTNING alla rete Wi-Fi
- Persistenza dei parametri di configurazione anche in caso di riavvio del dispositivo
- Possibilità di eseguire aggiornamenti del firmware da remoto, senza richiedere la riapertura del contenitore
- Invio regolare dei dati ad un broker MQTT appositamente configurato

#### 4.4.1 Decodifica del payload

Al fine di memorizzare efficacemente ogni singola rilevazione, viene mantenuta la struttura dati di seguito descritta.

```
struct particleSensorState_t {  
    uint16_t avgPM25 = 0;  
    uint16_t measurements[5] = {0, 0, 0, 0, 0};  
    uint8_t measurementIdx = 0;  
    boolean valid = false;  
    uint8_t status = 0;  
};
```

Quest'ultima incapsula un insieme di cinque misurazioni (svolte consecutive per aumentare la precisione della rilevazione), la loro media e ulteriori flag che indicano la classe di qualità rilevata dal sensore e la validità della misurazione. Il sensore viene regolarmente interrogato dal microcontrollore originale, inviando in risposta un payload di 20 byte. Di questi:

- I primi tre sono costanti (in caso di payload valido) e costituiscono l'**header**
- I byte 5 e 6 codificano il valore di qualità dell'aria rilevato

Il namespace **SerialCom** contenuto all'interno dell'header file **Utils.h** fornisce le funzionalità per:

- Leggere i dati dalla porta seriale (configurata tramite SoftwareSerial sul pin 2D del modulo ESP8266) all'interno di un buffer
- Effettuare cinque letture consecutive del valore di qualità dell'aria, calcolandone la media
- Individuare la classe di qualità alla quale appartiene il valore medio rilevato
- Verificare la validità dell'header (i cui tre byte devono corrispondere a 0x16 0x11 0x0B)
- Verificare la validità del checksum (la somma dei venti byte deve essere pari a 0)

In particolare, l'ottenimento del numero intero (codificato da due byte) relativo alla misurazione di PM2.5 da parte del sensore viene permesso dalla seguente operazione bitwise, che applica un padding destro di 8 bit al primo byte e, successivamente, effettua un **OR** tra il primo e secondo byte. Il risultato, una volta codificato come dato di tipo **uint16\_t**, viene salvato nell'apposita struttura dati **struct particleSensorState\_t**.

```
const uint16_t pm25 = (serialRxBuf[5] << 8 | serialRxBuf[6]);
```

#### 4.4.2 Salvataggio e recupero dei parametri di configurazione

Al fine di rendere disponibile il salvataggio dei parametri di configurazione personalizzabili e il loro successivo recupero in seguito ad un eventuale riavvio del microcontrollore, all'interno dell'headerfile **Utils.h** il namespace **Config** definisce due funzioni che permettono, rispettivamente, di serializzare i parametri di configurazione in un file JSON all'interno della memoria flash del dispositivo e di leggere i parametri a partire da un eventuale file già presente in memoria.

Ciò è reso possibile dalla libreria **LittleFS**, che permette l'indicizzazione di un filesystem all'interno della memoria flash del dispositivo (la quale risulta avere una capienza pari a 4MB) tramite tecniche di wear levelling dinamico che ne limitano l'usura.

#### 4.4.3 Aggiornamento del firmware da remoto

L'aggiornamento via rete del firmware è reso possibile dalla libreria **ArduinoOTA** (On The Air), che permette di istruire il microcontrollore alla ricezione di nuovi binari precompilati tramite una socket TCP appositamente aperta.

Successivamente, è stato possibile inoltrare aggiornamenti al microcontrollore tramite il seguente comando disponibile nel framework offerto da **PlatformIO**:

```
pio run --target upload --upload-port [VINDRIKTNING-IP]
```

La risposta ottenuta dipende dalla configurazione di ArduinoOTA all'interno della funzione di setup. L'aggiornamento di VINDRIKTNING presenta il seguente output:

```

CURRENT: upload_protocol = esptool
Uploading .pio/build/d1_mini_lite/firmware.bin
00:15:42 [DEBUG]: Options: {'esp_ip': '192.168.1.10', 'host_ip': '0.0.0.0', 'esp_port': 8266, 'host_port': 19571, 'auth': 'ikea'}
00:15:42 [INFO]: Starting on 0.0.0.0:19571
00:15:42 [INFO]: Upload size: 393296
00:15:42 [INFO]: Sending invitation to: 192.168.1.10
Authenticating...OK
00:15:42 [INFO]: Waiting for device...

Uploading: [                                         ]  0%
Uploading: [                                         ]  0%
Uploading: [                                         ]  0%
Uploading: [=                                       ]  1%
Uploading: [=                                       ]  1%
Uploading: [=                                       ]  1%
Uploading: [=                                       ]  2%
Uploading: [==                                      ]  2%
Uploading: [==                                      ]  2%

```

Al fine di prevenire upload accidentali, è stata definita una password che viene richiesta per completare la procedura di aggiornamento.

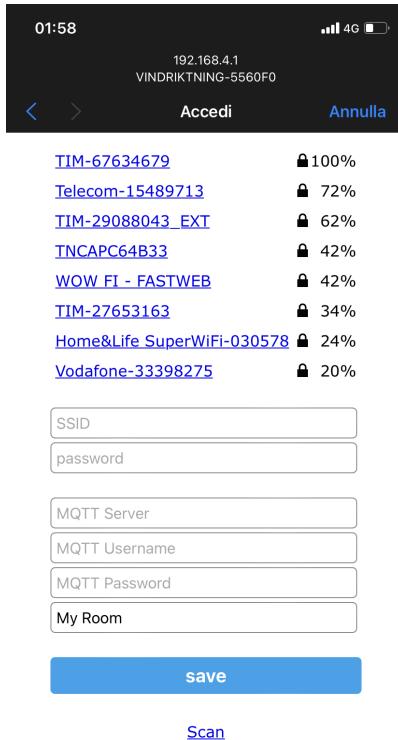
#### 4.4.4 Configurazione di parametri personalizzati

Per permettere all'utilizzatore il collegamento alla propria rete e la configurazione di parametri per la comunicazione con il Broker MQTT, è stata impiegata la libreria **WiFiManager**. Quest'ultima permette, sulla base della presenza o assenza di una rete WiFi alla quale il microcontrollore è in grado di connettersi, la conversione automatica del modulo ESP8266 da modalità **SoftAccessPoint** a **Station** e viceversa.

In modalità SoftAccessPoint, è possibile connettersi direttamente al microcontrollore, che espone una pagina web la quale permette all'utente di effettuare lo scan delle reti disponibili, la connessione ad una rete specifica e la configurazione dei parametri personalizzabili. Nel caso di VINDRIKTNING, l'utente ha la facoltà di specificare:

- L'indirizzo IP del Broker MQTT
- La porta del Broker MQTT
- Il nome utente e la password per inviare messaggi al Broker
- Un nome da assegnare allo specifico sensore (ad esempio, *Cucina*)

La schermata di configurazione visualizzato da uno smartphone è la seguente:



Nel caso in cui la connessione alla rete WiFi selezionata vada a buon fine, il modulo passa automaticamente a modalità Station ed entra a far parte degli host connessi alla rete specificata. I dati ottenuti da WiFiManager vengono, infine, serializzati in memoria flash tramite le funzionalità precedentemente descritte. In questo modo, non si rende necessario ripetere la procedura a fronte di un semplice riavvio del sistema.

#### 4.4.5 Comunicazione con il Broker MQTT

La gestione della connessione e dell'invio di messaggi al Broker MQTT è stata, invece, affidata alla libreria **PubSubClient**. Ad intervalli specificati dalla macro **MQTT\_PUBLISH\_INTERVAL\_MS**, un'eventuale rilevazione valida viene inviata al Broker sull'apposito topic di aggiornamento dello stato di qualità dell'aria. Nel caso di assenza di connettività, invece, il sistema tenta regolarmente una riconnessione finché questa non risulta avvenuta.

Due ulteriori messaggi vengono, infine, comunicati al broker al momento della connessione:

- Dichiarazione di **connessione** da parte di VINDRIKTNING al Broker
- Dichiarazione di **testamento** da parte di VINDRIKTNING al Broker

Quest'ultima permette la definizione di uno specifico messaggio con flag di ritenzione attiva che viene inviato dal broker a tutti i **subscriber** in caso di una brusca ed imprevista disconnessione da parte del sensore.

Al fine di fornire ad ogni sensore un identificatore univoco per agevolarne la comunicazione con il broker, viene descritto un apposito **sensorID** costituito dalla stringa **VINDRIKTNING-[chip-id]**, dove **chip-id** rappresenta l'**UUID** del modulo ESP8266. Di conseguenza, nonostante il nome a livello utente sia quello dichiarato nell'apposito campo "Name", un ulteriore identificativo viene reso disponibile al fine di poter individuare rapidamente il sensore nell'insieme di quelli connessi alla rete.

Sulla base dell'identificatore univoco di ogni sensore, i topic coinvolti risultano, quindi, i seguenti:

1. **airquality/[sensorID]/online** (Connessione del sensore identificato da **[sensorID]**)
2. **airquality/[sensorID]/offline** (Disconnessione del sensore identificato da **[sensorID]**)
3. **airquality/[sensorID]/status** (Nuova rilevazione dal sensore identificato da **[sensorID]**)

Il codice sorgente del firmware implementato risulta accessibile all'interno del progetto PlatformIO contenuto all'interno della directory **firmware**.

## 5 Definizione del Broker MQTT

Al fine di poter comunicare i dati sulla rete, VINDRIKTNING necessita di un indirizzo valido che identifichi un **Broker MQTT** adibito alla ricezione all'eventuale ritenzione dei messaggi. A tal fine, è stata scelta l'adozione del software open source **Eclipse Mosquitto**.

### 5.1 Containerizzazione di Eclipse Mosquitto

Sulla base delle decisioni architetturali riportate in precedenza, è stato scelto di eseguire Eclipse Mosquitto all'interno di un container **Docker**, sfruttando l'immagine ufficiale.

```
FROM eclipse-mosquitto

ADD ./broker/broker-entrypoint.sh /

ENTRYPOINT ["sh", "./broker-entrypoint.sh"]

CMD ["/usr/sbin/mosquitto", "-c", "/mosquitto/config/mosquitto.conf"]
```

A partire dall'immagine disponibile dal repository di *Docker Hub*, è stato aggiunto uno script shell per la configurazione del broker (eseguito come entrypoint all'avvio). L'esecuzione dello script (riportato di seguito) comporta:

- L'attribuzione dei corretti permessi alle cartelle relative alla configurazione di Mosquitto
- La verifica della presenza di un nome utente e una password all'avvio del container
- La definizione delle credenziali tramite l'utilità `mosquitto_passwd`

```
set -e

# Fix write permissions for mosquitto directories
chown --no-dereference --recursive mosquitto /mosquitto/log
chown --no-dereference --recursive mosquitto /mosquitto/data

mkdir -p /var/run/mosquitto \
&& chown --no-dereference --recursive mosquitto /var/run/mosquitto

if ( [ -z "${MOSQUITTO_USERNAME}" ] || [ -z "${MOSQUITTO_PASSWORD}" ] ); then
  echo "MOSQUITTO_USERNAME or MOSQUITTO_PASSWORD not defined"
  exit 1
fi

# create mosquitto passwordfile
touch passwordfile
mosquitto_passwd -b passwordfile $MOSQUITTO_USERNAME $MOSQUITTO_PASSWORD

exec "$@"
```

Una volta eseguito, il servizio risulterà accessibile alla porta 1883 del container.

## 6 Ricezione dei dati

### 6.1 Engine

La ricezione dei dati trasmessi dalle unità VINDRIKTNING è stata affidata ad un servizio denominato **Engine**. Quest'ultimo, una volta collegatosi al Broker MQTT in qualità di **subscriber** ed iscrittosi ai topic di interesse, interagisce con i dati ottenuti dai sensori connessi svolgendo le seguenti attività:

- Trasmissione dei dati al servizio centralizzato di storage
- Esecuzione di opportune routine per la notifica del superamento di determinati threshold

### 6.2 Containerizzazione

Il servizio è stato implementato in **Python** sfruttando le librerie **Requests** e **Paho MQTT Python Client** ed è stato, successivamente, containerizzato a partire dall'immagine **Alpine** tramite il Dockerfile di seguito riportato. Quest'ultimo prevede:

- L'installazione delle librerie necessarie
- La creazione di una cartella che andrà a contenere il file di log prodotto dal programma nel corso della sua esecuzione
- L'aggiunta del programma `engine.py` e dello script `entrypoint.sh` per l'inizializzazione del container

```
FROM alpine

RUN apk add --update --no-cache python3
RUN python3 -m ensurepip
RUN pip3 install --no-cache --upgrade pip setuptools
RUN pip3 install --no-cache --upgrade paho-mqtt
RUN pip3 install --no-cache --upgrade requests
RUN mkdir /log

ADD ./engine/engine.py /
ADD ./engine/entrypoint.sh /

ENTRYPOINT ["sh", "/entrypoint.sh"]

CMD ["python3", "/engine.py"]
```

Lo script `entrypoint.sh` esegue le seguenti istruzioni al fine di mantenere disponibile il logfile **antecedente** all'ultimo riavvio del sistema nel caso in cui questo sia disponibile grazie all'impiego di un volume persistente. Questo permette all'utilizzatore di individuare le motivazioni che hanno provocato la più recente interruzione del programma.

```
cp /log/logfile.log /log/report.log
cat /dev/null > /log/logfile.log
python3 /engine.py
```

Il codice sorgente dell'applicativo è consultabile al path `engine/engine.py` della repository.

### 6.3 Ricezione dei dati

La ricezione dei dati trasmessi dalle unità VINDRIKTNING al Broker MQTT sui topic d'interesse richiede l'autenticazione di Engine. A tal fine, il container prevede la definizione delle variabili d'ambiente `MOSQUITTO_USERNAME` e `MOSQUITTO_PASSWORD` in riferimento, rispettivamente, al nome utente e alla password previsti da Eclipse Mosquitto a seguito della configurazione sopradescritta.

### 6.4 Trasmissione dei dati ricevuti

Ad ogni nuovo messaggio ricevuto, Engine comunica la nuova osservazione o l'eventuale aggiornamento di stato di un sensore ad un servizio denominato **AirPI** (descritto nel dettaglio nel corso dei capitoli successivi). La completa comunicazione fra Engine ed AirPI richiede la presenza di:

- Un endpoint per la comunicazione di **aggiornamenti di stato** di un'unità VINDRIKTNING
- Un endpoint per la comunicazione di **nuove rilevazioni** da parte di un sensore
- Un endpoint per l'innesco di un **evento di notifica** all'atto del superamento di specifici threshold

Questi ultimi vengono rispettivamente codificati, all'interno del programma, dalle seguenti variabili d'ambiente:

- `SENSOR_STATUS_ENDPOINT`
- `DATA_ENDPOINT`
- `NOTIFICATION_ENDPOINT`

La comunicazione prevede, infine, un'autenticazione di Engine presso AirAPI basata su Json Web Tokens. A tal fine, l'applicativo necessita della presenza di:

- Un endpoint per l'**autenticazione** del servizio
- Un **identificativo** e un **secret** al fine di poter richiedere un token di autenticazione

Tali parametri vengono codificati dalle seguenti variabili d'ambiente:

- `AUTH_ENDPOINT`
- `AUTH_APPNAME`
- `AUTH_APPPASS`

## 7 Definizione di un database per lo storage dei dati

### 7.1 Scelta del DBMS

Al fine di permettere il salvataggio e la successiva fruizione dei dati, è stata impiegata una base di dati organizzata come servizio indipendente, anch'esso containerizzato. In particolare, è stata scelta l'adozione del DBMS **InfluxDB** sulla base della risposta alle seguenti necessità:

- Organizzazione dei dati orientata ai **timestamp**
- Definizione di una **retention policy** al fine di permettere l'eliminazione dei dati al di fuori del periodo di interesse

### 7.2 Containerizzazione

La containerizzazione del servizio ha richiesto la sola definizione del seguente Dockerfile che, nello specifico, prevede l'inclusione di un opportuno script di inizializzazione all'interno dell'immagine **InfluxDB** originale al fine di inizializzare il database utilizzato e la retention policy.

```
FROM influxdb:1.8
ADD ./influxdb/createdb.iql /docker-entrypoint-initdb.d/
```

In particolare, lo script `createdb.iql` (il cui contenuto è di seguito riportato) prevede la definizione di:

- Un database denominato `airquality` con retention-policy pari a 7 giorni
- Un utente con permessi di scrittura e lettura sulla base di dati
- Un utente con permessi di sola lettura sulla base di dati

```
CREATE DATABASE airquality WITH DURATION 7d
CREATE USER api WITH PASSWORD 'apisecret'
CREATE USER reader WITH PASSWORD 'read'
GRANT READ ON airquality TO api
GRANT READ ON airquality TO reader
GRANT WRITE ON airquality TO api
```

Una volta avviato, il servizio risulterà disponibile all'uso e accessibile tramite la porta **8086** del container.