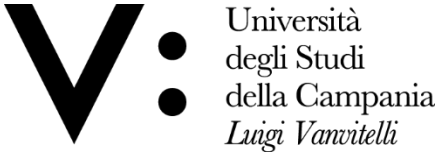


Università di degli Studi della Campania Luigi Vanvitelli - Dipartimento di Ingegneria

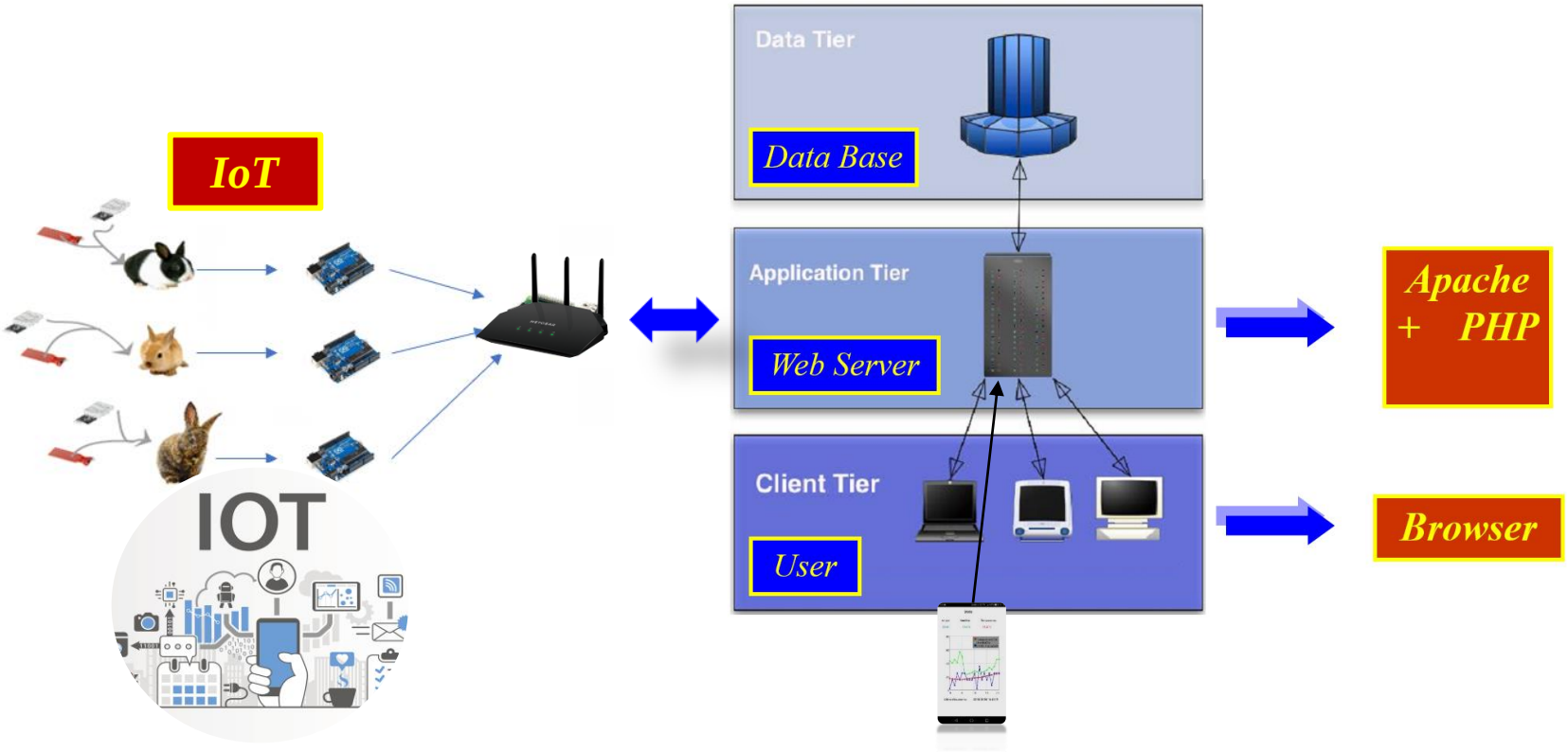
Laurea Triennale in Ingegneria Elettronica e Informatica

Laboratorio di Sviluppo di Applicazioni per IoT
***a.a.* 2023-2024**

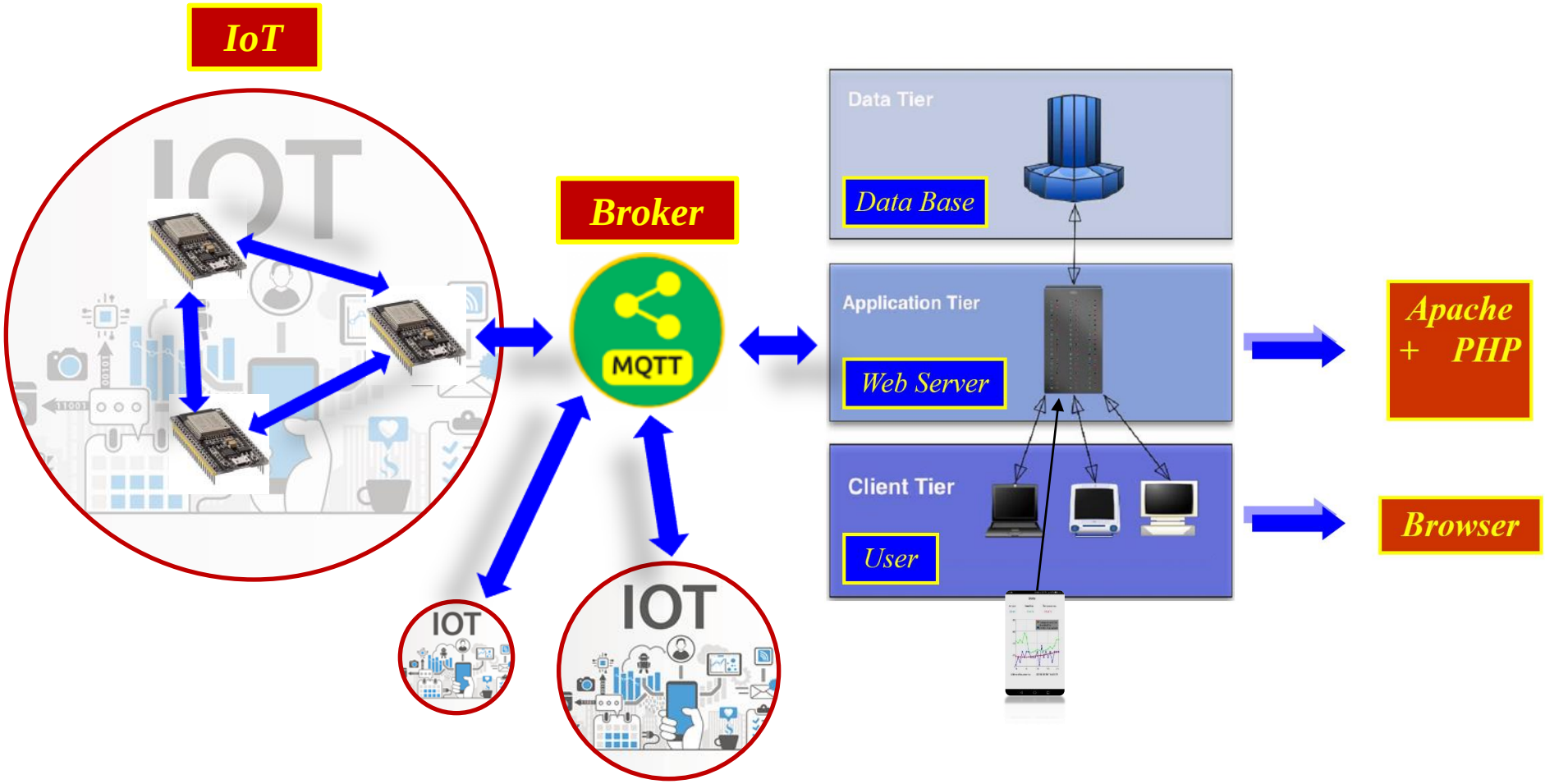
MQTT - Mosquitto

Docente: Carlo Mazzocca
e-mail: carlo.mazzocca@unibo.it

Architettura a Tre Livelli



Communication Middleware



Message Queue Telemetry Transport

Message Queueue Telemtry Transport (MQTT) è un standard per le comunicazione macchina-macchina

I dispositivi IoT usano MQTT per trasmettere e ricevere dati su una rete a risorse limitate in maniera semplice ed efficiente

MQTT supporta il trasferimento di messaggi tra dispositivi e cloud

MQTT e IoT

1) **Leggero ed efficiente**: richiede risorse minime in modo da poter essere supportato da microcontrollori

2) **Scalabilità**: consumo energetico minimo, funzioni built-in per supportare comunicazione con un grande numero di dispositivi IoT

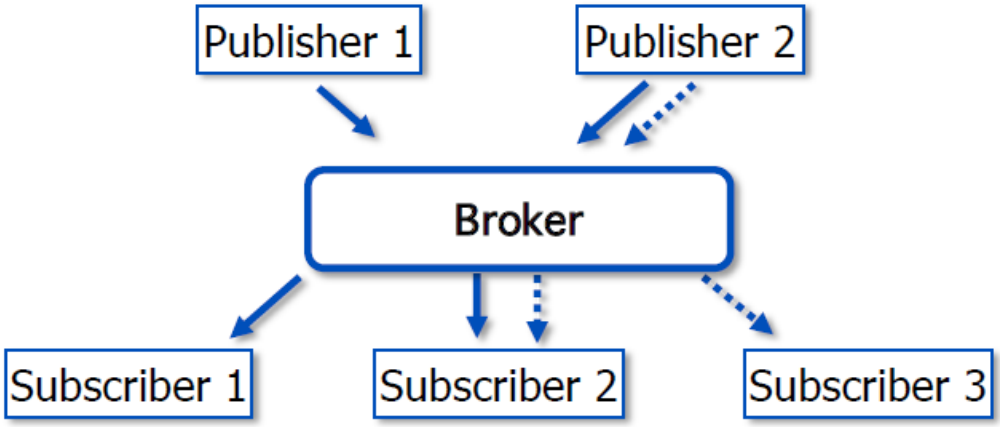
3) **Affidabile**: offre funzioni built-in che reduce il tempo richiesto ai dispositivi per connettersi con il cloud. Definisce tre diversi livelli di qualità del servizio

4) **Sicuro**: i messaggi possono essere cifrati e i dispositivi autenticati attraverso i moderni protocolli

Publish/Subscribe (Pub/Sub)

Pattern che separa un client, che invia messaggi riguardo un argomento specifico, chiamato **Publisher**, da un altro client (o più client), interessato a ricevere il messaggio, chiamato **Subscriber**

Tale pattern prevede la presenza di un **Broker**, in modo tale che ai publisher non sia noto quanti e quali siano i subscriber e viceversa



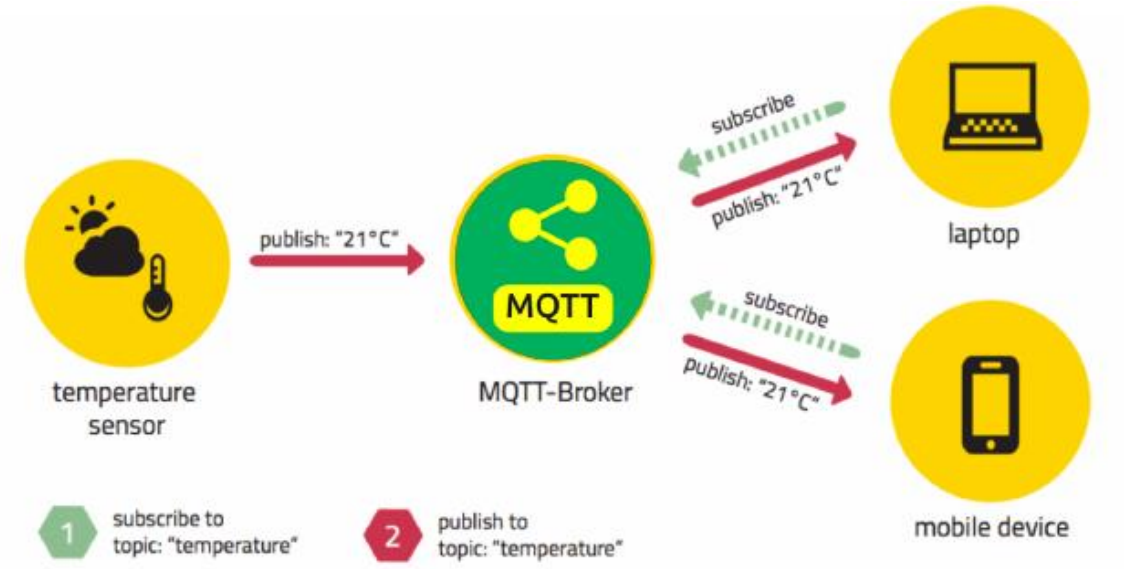
Diversi protocolli:

- **MQTT**, AMQP, XMPP

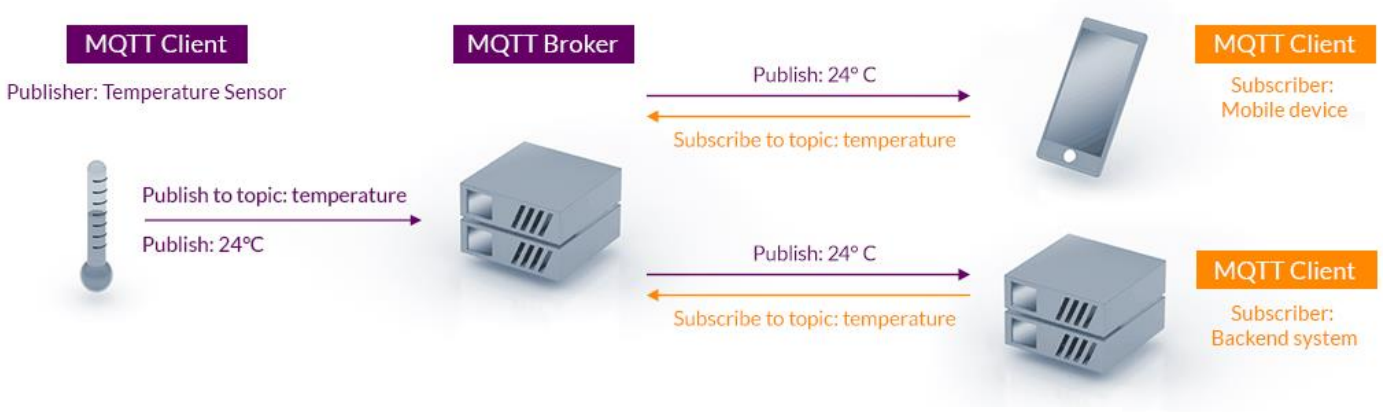
MQTT Comunicazione

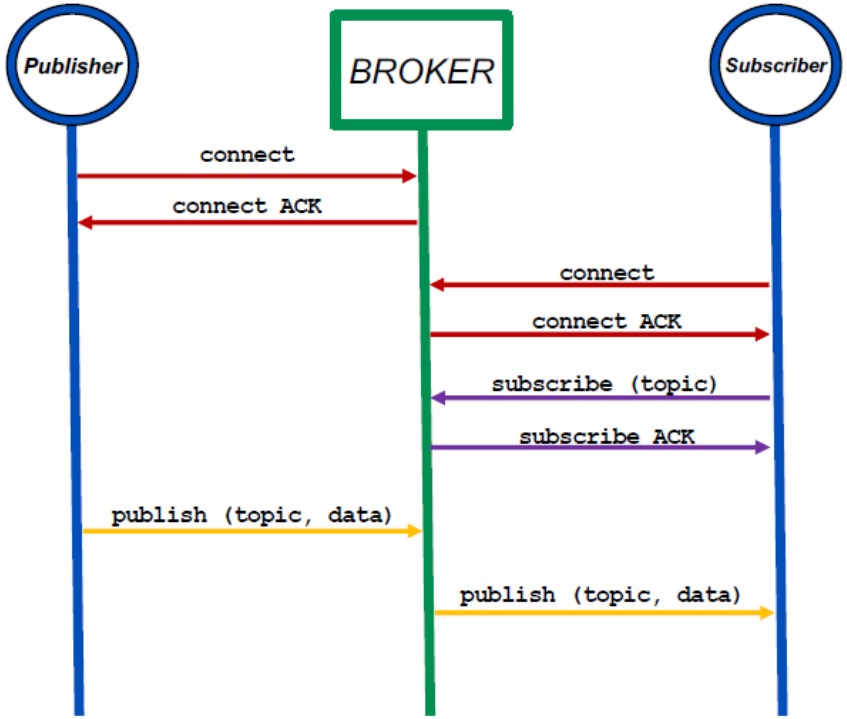
- Utilizza principalmente TCP, esiste una versione per le Wireless Sensor Network dove UDP sostituisce TCP(MQTT for Sensor Networks (MQTT-SN))

- websocket possono essere utilizzate per ricevere dati MQTT direttamente da browser



Interazione Publish/Subscribe





Topic

Un **topic** MQTT è una stringa di caratteri che rappresenta l'indirizzo di destinazione dei messaggi all'interno del sistema

I topic sono organizzati in una **struttura gerarchica** a livelli (/) che consente di filtrare e instradate i messaggi in base ai loro argomenti

```
livello1/livello2/livello3/…
```

Esempi:

```
house/room1/main-light
house/room1/alarm
house/garage/main-light
```

Wildcard

Per sottoscrivere più topic contemporaneamente:

- **#**: Copre tutti i livelli del topic che iniziano con il pattern specificato prima della wildcard. Ad esempio, `house/#` copre: `house/room1/main-light, house/garage/alarm, house/main-door`

- **+**: Copre tutti i topic associati ad una stringa valida arbitratia al posto dellea wildcard. Ad esempio, house/+/main-light: `house/room1/main-light, house/room2/main-light, house/garage/main-light`

MQTT Broker

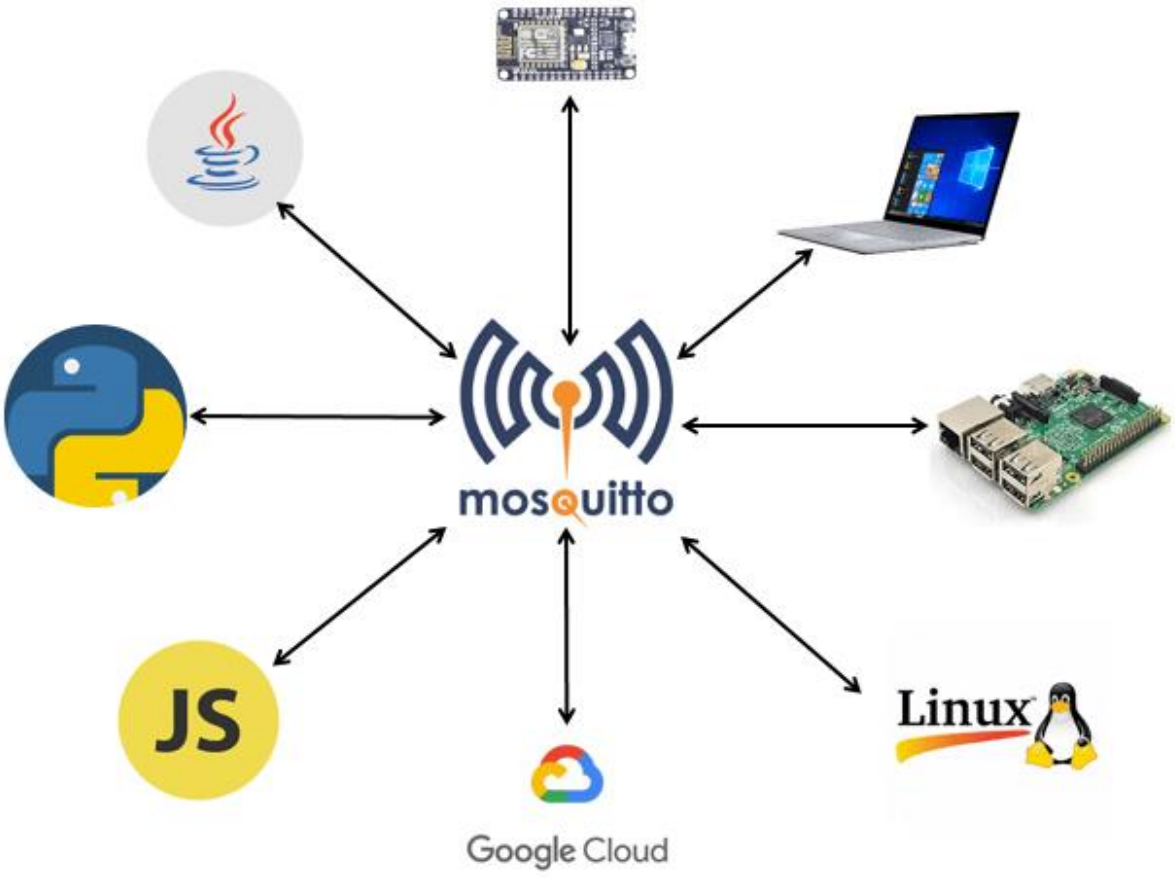
Broker più utilizzati:

- http://mosquitto.org/
- http://www.hivemq.com/ (fino a 25 connessioni)

Alternative:

- https://www.rabbitmq.com/mqtt.html
- http://activemq.apache.org/mqtt.html
- https://github.com/mqtt/mqtt.github.io/wiki/servers

Mosquitto



Sottoscrizione Topic

Sottoscrizione ad un topic:

```
mosquitto_sub -v -h IP_DEL_BROKER -p Porta_BROKER -u USERNAME -P PASSWORD -t "TOPIC"
```

Esempio:

```
mosquitto_sub -v -h 192.168.1.148 -p 550 -t "#"
mosquitto_sub -v -h 192.168.1.148 -p 550 -t "sensore/sens1"
```

Per catturare tutti i messaggi che contengono la stringa '*temp*'

```
mosquitto_sub -v -h 192.168.1.148  -p 550 -t "sensore/#" | grep temp
```

Pubblicazione Topic

Pubblicazione di un topic:

```
mosquitto_pub -h IP_DEL_BROKER -p PORT -u USERNAME -P PASSWORD -t TOPIC -m MESSAGE
```

Esempio:

```
mosquitto_pub  -h 192.168.1.148 -p 550  -t "sensore/sens1"   -m  '30'
```

Accesso Remoto

Il Broker può essere acceduto anche da client non in esecuzione sulla stessa macchina → modificare il file *mosquitto.conf* aggiungendo le seguenti stringhe:

- listner 1883
- allow_anonymous true

Riavviare il Broker:

- **`mosquitto -v -c`** `mosquitto.conf`

Importare in Arduino IDE *«EspMQTTClient»*

NB: Se necessario, disabilitare il firewall (porta 1885)

Esempio

Mosquitto Broker:

- mosquitto -v

Subscriber:

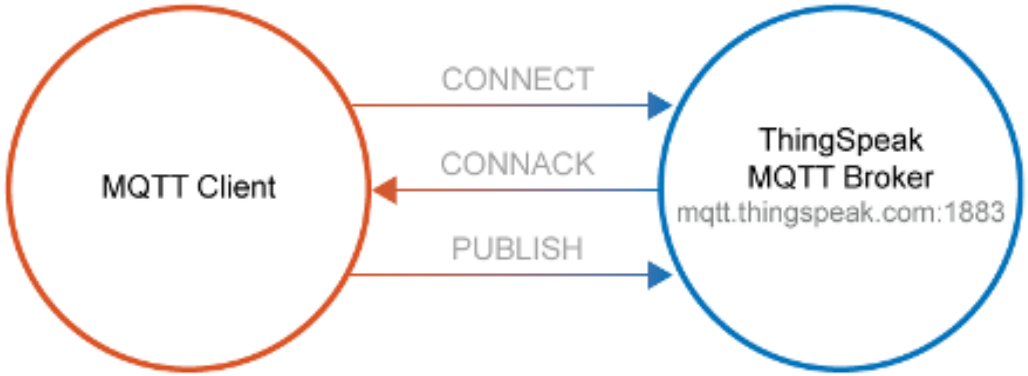
- mosquitto_sub –h 192.168.1.148 -t “sensore/sens1” –v

Publisher:

- mosquitto_pub –h 192.168.1.148 -t “sensore/sens1” -m 'hello'

Connessione Client – Broker MQTT

- Il protocollo MQTT si basa su TCP/IP
- La connessione MQTT non avviene mai direttamente tra due client, ma sempre tra un client e un broker
- Per avviare una connessione, il client invia un messaggio di CONNCECT al broker, il quale risponde con un ulteriore messaggio, CONNACK, e un codice di stato



Connessione Client – Broker MQTT

Non appena la connessione viene stabilita, il broker la mantiene aperta fino a quando il client non invia un comando di disconnessione oppure non appena si interrompe

Se il messaggio di connessione non è corretto o passa troppo tempo tra l'apertura di una socket di rete e l'invio del messaggio di connessione, il broker chiude la connessione

Messaggio CONNECT

- ***clientID***: Identifica ogni client MQTT che si collega ad un broker MQTT, pertanto deve essere univoco;

- ***cleanSession***: Flag che indica al broker se il client desidera stabilire una sessione persistente, e allora il broker archivia tutte le sottoscrizioni per il client in modo tale che se il client dovesse disconnettersi dal broker, alla prossima richiesta di CONNCECT si troverebbe automaticamente già sottoscritto a tutti i topic precedenti. Se il client si sottoscrive in modo non persistente, invece, il broker non memorizza nulla per quel client;

Messaggio CONNECT

- ***Username/Password***: identificare/autorizzare il client;
- ***Will Message***: Quando un client si connette, può fornire ai broker la sua ultima volontà (Last Will Topic) sottoforma di messaggio e topic all'interno del messaggio di connessione. Se il client si disconnette in modo inopportuno, il broker invia il messaggio LWT per conto del client;
- ***Keep Alive***: È un intervallo che rappresenta il periodo di tempo più lungo che client e broker possono aspettare senza inviare un messaggio

Quality of Service (QoS)

- Stabilisce il grado di garanzia col quale il messaggio deve essere trasmesso
- Offre la possibilità di scegliere un livello di servizio che corrisponde all'affidabilità della rete
- Il client che pubblica un messaggio sul broker definisce il livello di QoS del messaggio
- Il broker invia ciascun messaggio ai Subscriber utilizzando il livello di QoS definito da ciascun Subscriber durante il processo di sottoscrizione. Se il Subscriber presenta un QoS inferiore rispetto al Publisher, il broker trasmette il messaggio con una QoS inferiore

Livelli QoS - 0

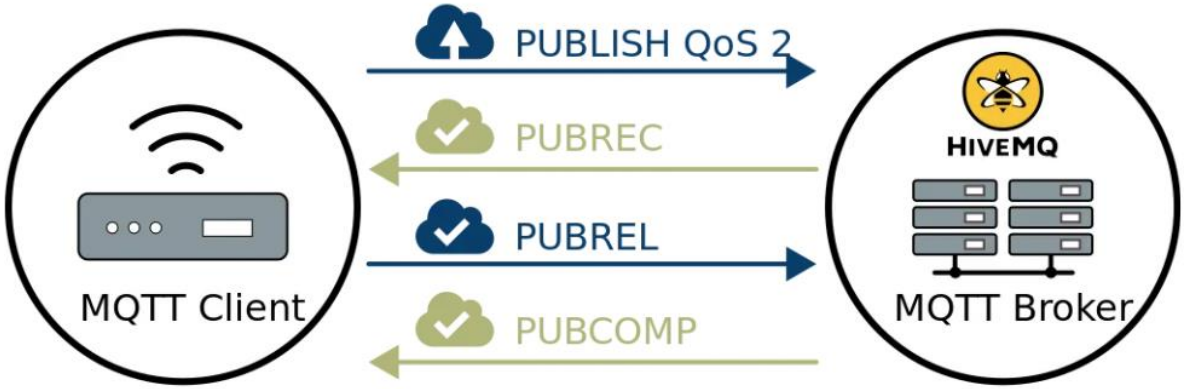
- ***Al massimo una volta* (0)**: Il destinatario non invia alcuna conferma di avvenuta ricezione del messaggio e quest'ultimo non viene in alcun caso archiviato e ritrasmesso

Livelli QoS - 1

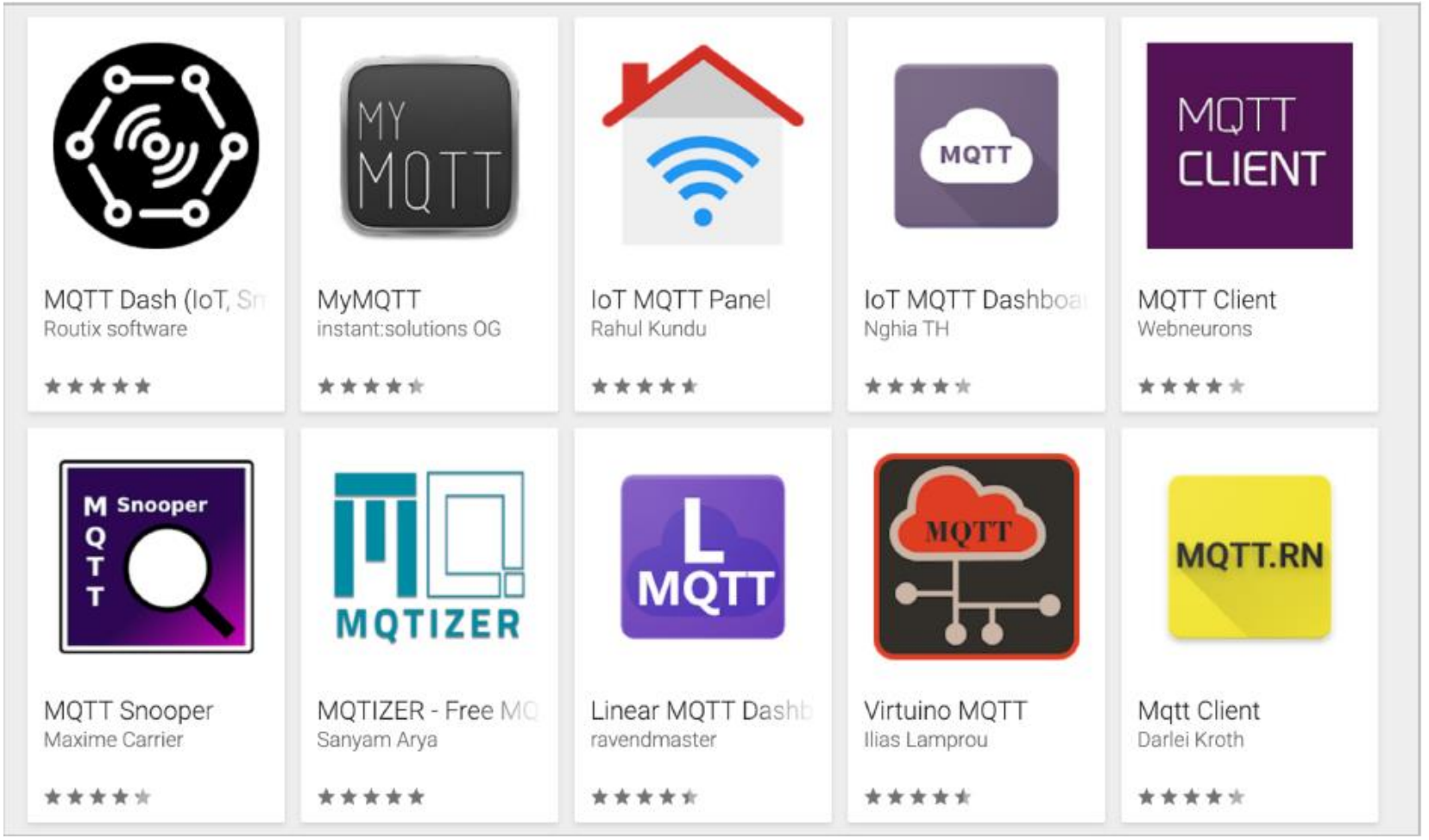
- ***Almeno una volta* (1)**: Il recapito del messaggio al destinatario è garantito per almeno una volta. Il client Publisher mantiene il messaggio archiviato fino a quando non riceve un pacchetto *PUBACK* da parte del destinatario. Il *PUBACK* è legato al messaggio del publisher mediante *packet Identifier*. Se il Publisher non riceve un pacchetto *PUBACK* per il messaggio entro un certo istante di tempo, lo re-invia con il flag duplicato (*DUP*) alto e pari a 1

Livelli QoS - 2

- ***Esattamente una volta* (2)**: È il più alto e sicuro livello di servizio in MQTT, ma anche il più lento. Esso garantisce che ogni messaggio sia ricevuto una sola volta dai destinatari previsti, attraverso almeno due flussi di richiesta/risposta tra il mittente e il destinatario (four-part handshake). In questo caso c'è uno scambio di quattro pacchetti: PUBLISH, PUBREC, PUBREL e PUCOMP. Alla fine del flusso, entrambe le parti sono sicure che il messaggio è stato recapitato



MQTT Client - Android

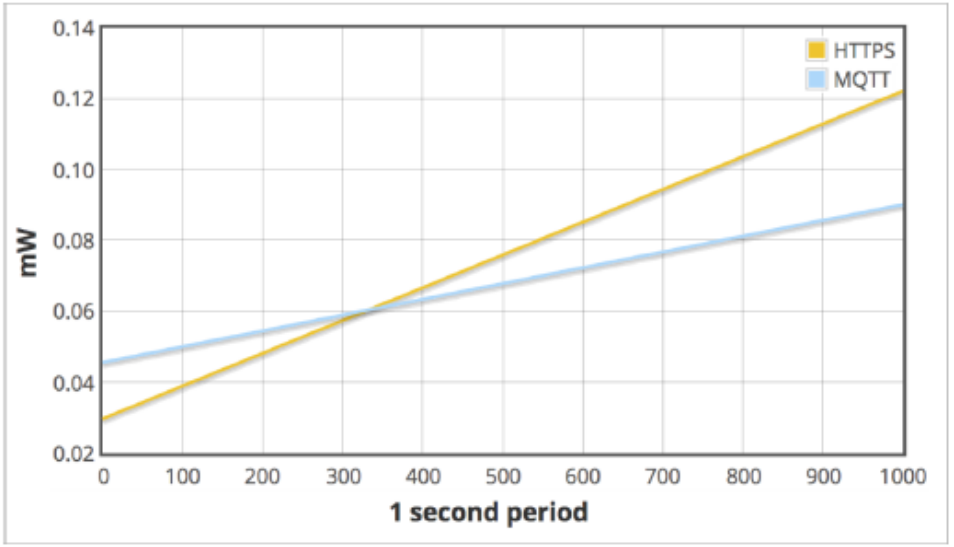


MQTT vs HTTPS

cost of 'maintaining' that connection (in % Battery / Hour):

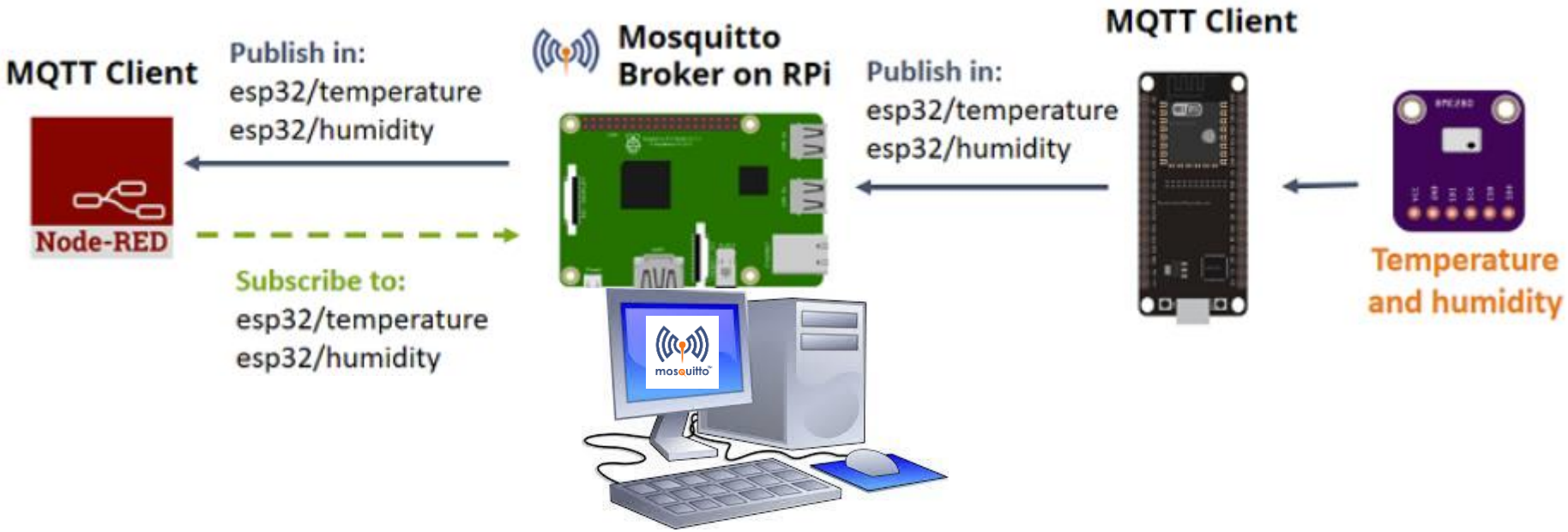
	% Battery / Hour			
	3G		Wifi	
Keep Alive (Seconds)	HTTPS	MQTT	HTTPS	MQTT
60	1.11553	**0.72465**	0.15839	**0.01055**
120	0.48697	**0.32041**	0.08774	**0.00478**
240	0.33277	**0.16027**	0.02897	**0.00230**
480	0.08263	**0.07991**	0.00824	**0.00112**

3G – 240s Keep Alive – % Battery Used Creating and Maintaining a Connection

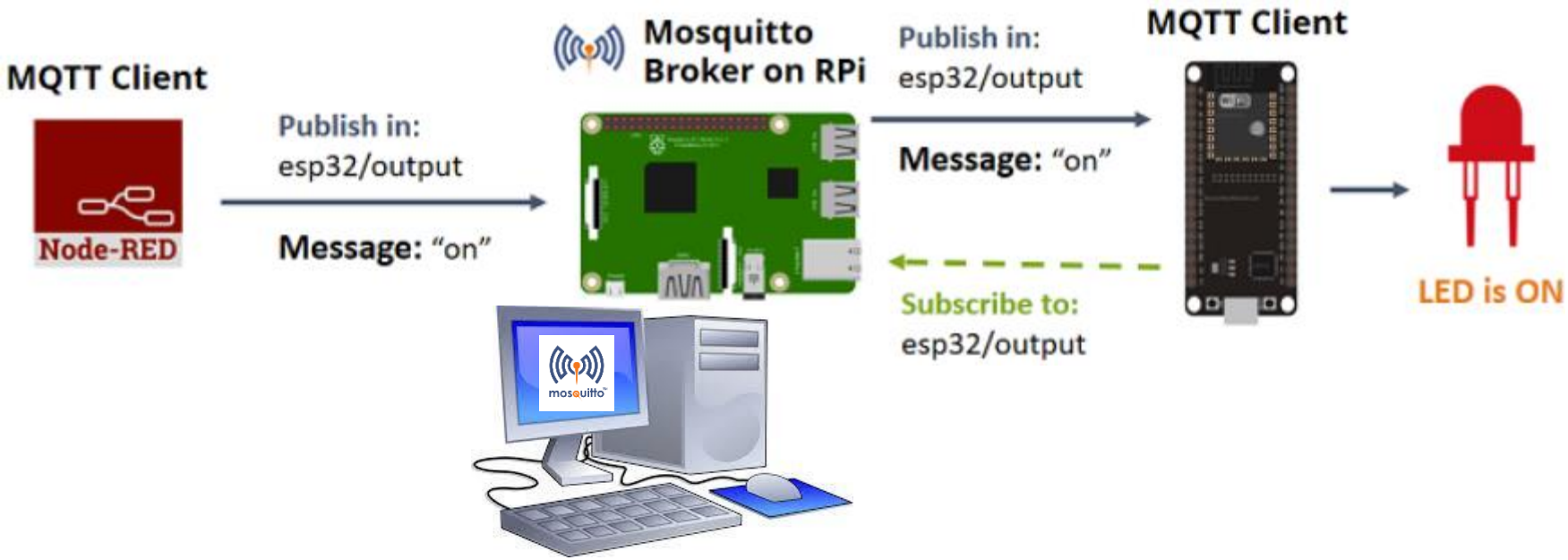


http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https

ESP e MQTT - Publish



ESP e MQTT - Subscribe



Sketch – MQTT Client

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "name";
const char* password = "password";

// MQTT Broker Configuration:
const char *topic = "sensore/sens1";
const char* mqtt_server = "my ip";
const int mqtt_port = 1883; // default 1883

WiFiClient espClient;  // WiFi client
PubSubClient client(espClient);  //MQTT client


void setup() {
  // Start serial port at 9600 bps:
  Serial.begin(9600);
  WiFi.begin(ssid, password);  delay(3000);

  // Start WiFi connection:
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(5); Serial.print(".");
  }
  Serial.println();
  Serial.println("Connected to the WiFi network.");
  Serial.print("ESP IP_Address: ");
  Serial.println(WiFi.localIP());

  // MQTT connection:
  Serial.println();
  client.setServer(mqtt_server,mqtt_port);
  client.setCallback(callback);
  long lastMsg = 0;  char msg[50];  int value = 0;
  if (!client.connected()) {
    reconnect();
  }
}

void loop() {
  client.loop();

  if (Serial.available()) {    // Verifica la presenza di un carattere
    String str = "";
    str = Serial.readString();   // Legge la stringa
    int str_len = str.length() + 1;
    char char_array[str_len];
    str.toCharArray(char_array, str_len); // Converte la stringa in array_char
    boolean rc = client.publish(topic,char_array); // Pubblica la stringa letta
  }
}

void reconnect() {
  // Loop until we're reconnected
  Serial.print("Attempting MQTT connection...");
  while (!client.connected()) {
    Serial.print("...");
    // Attempt to connect
    if (client.connect("esp32")) {
      Serial.println();
      Serial.println("Connected to the mosquitto Broker.");
      // Subscribe
      client.subscribe(topic);
      Serial.print("Subscribed to the topic: ");
      Serial.println(topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Sketch – MQTT Client

```
76  // Function to process received messages
77  void callback(char* topic, byte* message, unsigned int length) {
78    //Serial.print("Message arrived on topic: ");
79    //Serial.println(topic);
80    Serial.print(" - Message: ");
81    String messageTemp;
82
83    for (int i = 0; i < length; i++) {
84      Serial.print((char)message[i]);
85      messageTemp += (char)message[i];
86    }
87    Serial.println();
88  }
```

Esempio – Avvio MQTT

```
C:\Program Files\mosquitto>mosquitto -v -c mosquitto.conf
1715069748: mosquitto version 2.0.18 starting
1715069748: Config loaded from mosquitto.conf.
1715069748: Opening ipv6 listen socket on port 1883.
1715069748: Opening ipv4 listen socket on port 1883.
1715069748: mosquitto version 2.0.18 running
1715069749: New connection from ::1:56131 on port 1883.
```

```
C:\Program Files\mosquitto>mosquitto_pub -t "sensore/sens1" -m "Test"
```

```
C:\Program Files\mosquitto>mosquitto_sub -t "sensore/sens1"
Test
```

```
Connected to the WiFi network.
ESP IP_Address: 192.168.1.207

Attempting MQTT connection......
Connected to the mosquitto Broker.
Subscribed to the topic: sensore/sens1
 - Message: Test
```

Tutorial

- https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/

- https://www.emqx.com/en/blog/how-to-use-mqtt-in-php