PABLITTO

Antonino Romeo

Sommario

1. Introduzione

L'obiettivo del progetto pablitto è l'implementazione del protocollo di livello applicativo MQTT utilizzato nel mondo IoT con il suo approccio di publish/subscriber.
La versione del protocollo a cui si fa riferimento nel progetto è la 3.1.1 (protocol level 0x04) ed è liberamente scaricabile al seguente link:

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Le applicazioni sviluppate sono:

- Libreria pablitto-client in C++: la libreria mette a disposizione delle API per comunicare con un broker MQTT. A titolo esemplificativo sono state sviluppate pablitto-pub e pablitto-sub che si occupano rispettivamente della pubblicazione su un determinato topic e della sottoscrizione.
- pablitto-broker in Go.

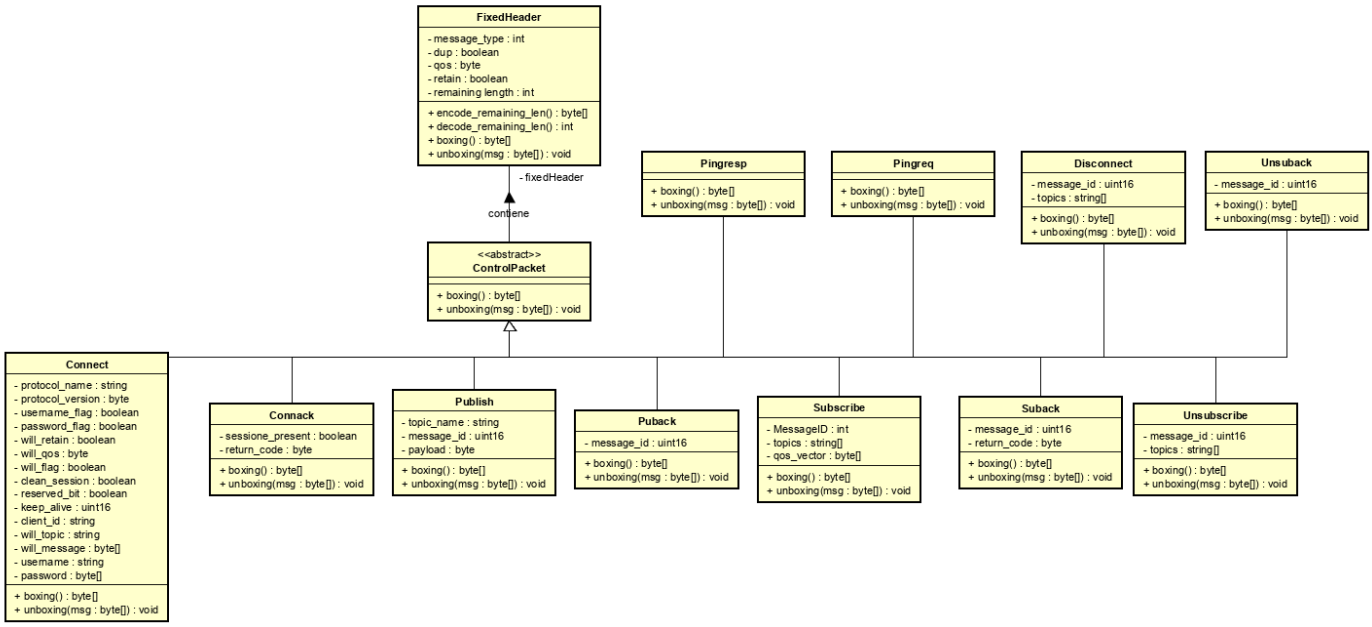
Librerie esterne utilizzate: Nessuna.

Le applicazioni pablitto-pub, pablitto-sub che utilizzano la libreria libpablitto e l'applicazione pablitto-broker rispettando il formato dei control packet di MQTT sono assolutamente indipendenti tra di loro. Ciò è stato testato facendoli comunicare con mosquitto-broker e mosquitto-pub/mosquitto-sub rispettivamente, sarà riportata una descrizione di tale interazione nel seguito.

Si noti che tutto il codice prodotto è stato testato su Debian 10 buster.

2. Control Packet

I messaggi che si scambiano i publisher/subscriber con il broker in MQTT vengono chiamati Control Packet, ognuno di questi è composto da un Fixed Header (minimo 2 byte, massimo 6 byte) e da un variable header. La figura sotto mostra come sono stati modellati nelle rispettive implementazioni. Nei sotto-paragrafi del presente paragrafo sarà riportata una breve descrizione di ciascuno di essi per comprendere il funzionamento del software.



2.1. Fixed Header

Il Fixed Header rappresenta l'intestazione fissa del control packet ed è sempre presente.

- Message-type: indica il tipo di control packet. Il valore va da 0 a 15, dove lo 0 e il 15 sono valori riservati.
- Dup: indica se il control packet inviato è un duplicato, questo campo serve esclusivamente per la qos=2.
- Qos: può assumere i valori 0, 1, 2 che indicano rispettivamente "at most once", "at least once" e "exactly once". Ricordando che MQTT utilizza TCP a livello di trasporto, la consegna dei messaggi è certamente garantita, la qos serve nel caso in cui la connessione di rete cada.
- Remaining length: indica la lunghezza del variable header che è equivalente a $control\_packet - fixed\_header$. Il remaining length può essere rappresentato minimo da 1 byte fino a 4 byte, il metodo encode_remaining_len e decode_remaining_len servono a codificare e decodificare.
  L'algoritmo è semplice, preso un byte, il bit più significativo indica se è 1 che ci sarà un altro byte, se è zero indica che quello è l'ultimo byte della rappresentazione. I 7 bit restanti del byte sono utilizzati per la rappresentazione del numero. I byte sono valutati secondo Big Endian in quanto lo standard impone in questa maniera.

2.2. Connect Packet

Il Connect Packet rappresenta il control packet che serve per la connessione a livello applicativo tra il client e il broker.

- protocol name: questo campo secondo lo standard è “MQTT”, in particolar modo come tutte le stringhe che ci sono nei control packet è rappresentato in utf8, ed è preceduto da due byte denominati string length MSB e string length LSB, che presi insieme ci indicano la lunghezza della stringa che segue. Nel caso di protocol name visto che è fissato dallo standard avremo MSB=00000000, LSB=00000100.
- protocol version o protocol level: indica la versione dello standard, nel caso della 3.1.1 l’identificativo è 0x04.
- clean_session: flag che indica al broker una richiesta di una connessione persistente o meno.
- will_flag: flag che indica la presenza di una “ultima volontà”, nel caso questo sia attivo attiverà la presenza dei campi will_topic, will_message.
- will_topic: indica il topic su cui pubblicare il will_message.
- will_message: contiene il messaggio nel caso di ultime volontà.
- Username_flag, password_flag: indicano se nel Connect Packet saranno presenti i campi username e password. Il meccanismo di autenticazione non viene specificato nello standard, pertanto come trattare questi campi è una scelta del broker.
- Keep_alive: è un campo fondamentale, che indica una durata in secondi, questo dirà al broker che se non riceve messaggi dal client entro il keep_alive stabilito eliminerà lo stato del client. Il keep_alive è un heartbeat che indica al broker che il client è ancora vivo, viene implementato in pablitto-client con un thread.
  il keep_alive è un campo di 16 bit.

2.3. Connack Packet

Il Connack Packet rappresenta il control packet di risposta alla connect del broker. Il broker risponde sempre con una connack a meno che qualcosa non sia andato storto.

- Session_present: dipende dal flag clean_session della connect.
- Return_code: indica la risposta del server alla connect. Di seguito sono mostrati i possibili valori:
  - 0x00 Connection Accepted.
  - 0x01 Connection Refused, unacceptable protocol version.
  - 0x02 Connection Refused, identifier rejected.
  - 0x03 Connection Refused, Server unavailable.
  - 0x04 Connection Refused, bad username or password.
  - 0x05 Connection Refused, not authorized.
  - Valori da 6 a 255 riservati per usi futuri.

2.4. Publish Packet

Il Publish Packet rappresenta il control packet per pubblicare su un topic. Questo control packet viene usato sia dal client verso un broker che da un broker verso un client.

- topic_name: è una stringa che identifica il topic su cui pubblicare. Come viene gestito questo topic verrà spiegato in seguito.
- Message_id: È un uint16_t. Identificativo del messaggio, questo campo serve solo nel caso di QoS differente da zero, in modo da poter verificare che la puback ricevuta sia relativa al pacchetto di publish corrispondente.
- Payload: il payload non è altro che un insieme di byte, cosa rappresentano questi byte trasferiti non è compito dello standard, sta all'applicazione scegliere il significato dei byte inviati.
  Nel caso di pablitto-pub e pablitto-sub, il payload scambiato saranno delle stringhe, in maniera simile a quello che avviene con mosquitto-pub e mosquitto-sub.

2.5. Puback Packet

Il Puback Packet rappresenta il control packet di risposta ad una Publish con QoS uguale a 1 o 2. Contiene il message_id del pacchetto a cui fa riferimento.

2.6. Subscribe Packet

Il Subscribe Packet rappresenta il control packet che un client invia per sottoiscriversi ad un insieme di topic.

- message_id: identificativo del packet, serve in quanto vi sarà la suback corrispondente.
- Topics: sono un insieme di stringhe ciascuna identificativa di un topic cui il client vuole iscriversi.
- qos_vector: rappresenta le qos corrispondenti a ciascuno dei topic specificati.

2.7. Suback Packet

Il Suback Packet rappresenta il control packet che un broker invia come risposta ad una subscribe di un client. Al seguito di una subscribe il broker risponde sempre con una suback a meno che qualcosa non sia andato storto.

- message_id: contiene l'identificativo della subscribe a cui fa riferimento.
- Return_code_vector: contiene la risposta del broker con il QoS che viene garantito a ciascun topic.

2.8. Unsubscribe Packet

L'Unsubscribe Packet rappresenta il control packet che un client invia per rimuovere la sottoiscrizione ad un insieme di topic.

- message_id: identificativo del packet, serve in quanto vi sarà la unsuback corrispondente.
- Topics: sono un insieme di stringhe ciascuna identificativa di un topic cui il client vuole annullare la sua sottoiscrizione.

2.9. Unsuback Packet

L'Unsuback Packet rappresenta il control packet che un broker invia come risposta ad una unsubscribe di un client. Al seguito di una unsubscribe il broker risponde sempre con una unsuback a meno che qualcosa non sia andato storto.

- message_id: contiene l'identificativo della unsubscribe a cui fa riferimento.

2.10. Pingreq Packet

Il Pingreq Packet rappresenta il control packet che un client invia una volta che non ha nulla da inviare e sta scadendo il keep alive stabilito.

2.11. Pingresp Packet

Il Pingresp Packet rappresenta il control packet che un broker invia ad un client una volta che ha ricevuto il Pingreq, questo indica al broker che il client è ancora vivo.

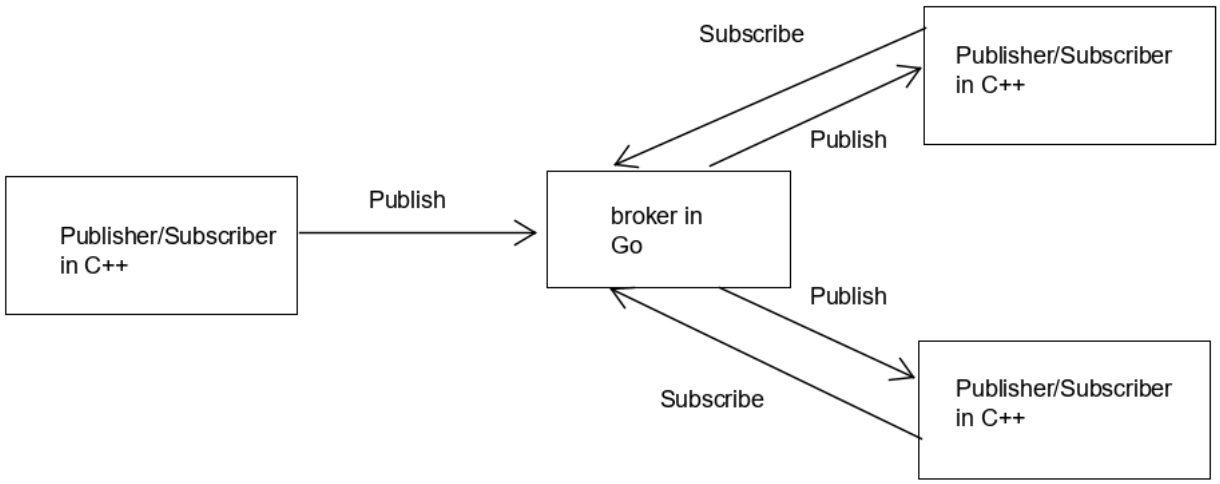
2.12. Disconnect Packet

Il Disconnect Packet rappresenta il control packet che un client invia ad un broker per disconnettersi. Non vi è un ack a questo pacchetto in quanto non è necessario, se il broker non riceve un pingreq o un messaggio utile entro il keep alive disconnetterà lo stesso il client.

Si noti che la QoS=2 non è stata implementata pertanto viene tralasciata la descrizione dei pacchetti pubrec e pubrel.

3. Architettura del sistema

Sotto viene mostrato il flusso tipico dove ci sono dei client che si sono sottoiscritti ad un topic, un client in un momento successivo effettua una publish in un topic che matcha i topic dei client sottoiscritti e quindi il broker si occupa di effettuare lo smistamento.
I client sono totalmente disaccoppiati tra loro, essi ricevono i valori registrati a topic di loro interesse, non hanno alcun rapporto diretto con chi pubblica il valore.

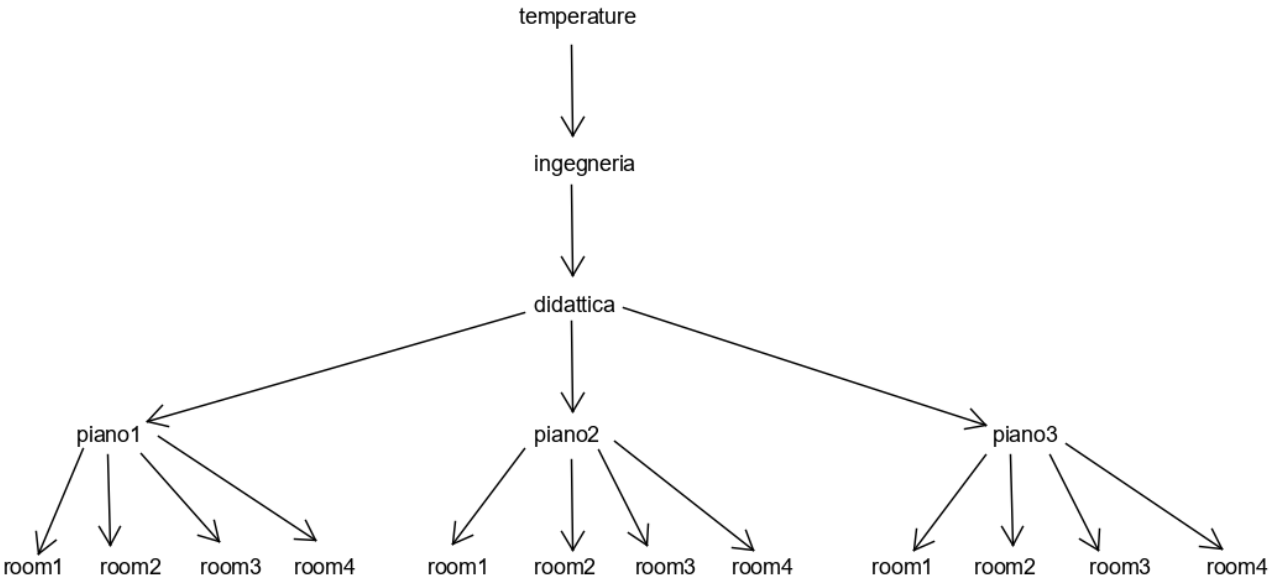


Si ricordi che le due applicazioni sono totalmente indipendenti, dal lato client potrebbe esserci pablitto-client e dal lato broker potrebbe esserci mosquitto, così come dal lato broker potrebbe esserci broker pablitto e dal lato client mosquitto-pub/mosquitto-sub.

Le applicazioni sviluppate parlano con qualsiasi applicazione che implementi MQTT versione 3.1.1.

4. Descrizione di un topic

Il topic rappresenta l'identificativo su cui verrà pubblicato un valore. Esso è rappresentato da una struttura ad albero in modo simile a quanto avviene in un file system. Ad Esempio, si consideri una struttura come quella riportata nella figura sottostante.



Un topic è composto da una serie di stringhe separate da un separatore, nel caso di MQTT il carattere separatore è lo slash ("/"). Sopra è mostrata una struttura gerarchica che rappresenta i sensori di temperatura che troviamo nelle varie stanze di una ipotetica facoltà di ingegneria.

Esempi di topic sono:

- temperature/ingegneria/didattica/piano3/room2
- temperature/ingegneria/didattica/piano1/room4
- e così via.

Se un client è interessato alla temperatura della stanza 2 del piano 3 della didattica della facoltà di ingegneria, effettuerà una subscribe al topic temperature/ingegneria/didattica/piano3/room2.

Quando un publisher pubblicherà un valore su quel topic, i sottoscrittori riceveranno l'informazione.

MQTT utilizza due simboli speciali che sono:

- single level wildcard ("+"). Ad esempio, temperature/ingegneria/didattica/+/room2 matcherà tutte le room2 di tutti i piani dell'edificio.

- Multi level wildcard ("#"). Ad esempio, temperature/ingegneria/didattica/piano3/# matcherà tutto ciò che sta dopo il piano3, nella struttura sopra in questione matcherà tutte le room del piano 3.

5. pablitto-client

Link su github: https://github.com/antromeo/pablitto-client.git

Tra i due applicativi sviluppati, pablitto-client è certamente quello più completa. Il progetto presenta la directory lib dove si trovano tutti i file .cpp contenti le varie implementazioni, e nella directory include dove si trovano tutti i file instestazione (.hh). Nella directory principale inoltre si trovano i file .cpp degli applicativi pablitto_pub e pablitto_sub.

La libreria si compone dei seguenti file:

- fixed_header.hh, control_packet.hh, control_packet.cpp, connect_packet.hh, connect_packet.cpp, connack_packet.hh, connack_packet.cpp, pingreq_packet.hh, pingreq_packet.cpp, disconnect_packet.hh, publish_packet.hh, publish_packet.cpp, puback_packet.hh, puback_packet.cpp, suback_packet.hh, suback_packet.cpp, subscribe_packet.hh, subscribe_packet.cpp, unsubscribe_packet.hh, unsubscribe_packet.cpp, unsuback_packet.hh, unsuback_packet.cpp. Questi si trovano nel namespace packets.
- SocketTCP.hh, SocketTCP.cpp nel namespace network.
- pablittoMQTT.hh, pablittoMQTT.cpp.
- utilities.hh.
- A titolo esemplificativo dell'utilizzo della libreria vi sono le seguenti applicazioni: pablitto_pub.cpp e pablitto_sub.cpp

5.1. Descrizione della gestione interna

La libreria utilizza socket TCP per connettersi ad un broker, utilizza due thread internamente, un thread per controllare se vi sono messaggi in ricezione e metterli in delle code, un thread che controlla che non sia scaduto il keep alive e nel caso manda un pingreq.
Il thread che monitora il keep alive fa l'assunzione di inviare il pingreq dopo keep_alive/2 secondi approssimando la velocità di trasmissione.

Per quanto concerne i thread è stata utilizzata la libreria <thread> disponibile in C++11.

5.2. pablitto-pub

pablitto-pub è un client esemplificativo che si occupa di pubblicare su un determinato topic una stringa.

Si avvia l'applicazione da terminale e si inseriscono i vari parametri, pablitto-pub farà dei controlli in ingresso sulla correttezza di quanto inserito.

Link YouTube che mostra il funzionamento: https://www.youtube.com/watch?v=shWXoqBa2OY&t=7s

5.3. pablitto-sub

pablitto-sub è un client esemplificativo che si occupa di sottoiscriversi su un determinato topic una stringa.

Si avvia l'applicazione da terminale e si inseriscono i vari parametri, pablitto-sub farà dei controlli in ingresso sulla correttezza di quanto inserito.

Link YouTube che mostra il funzionamento: https://www.youtube.com/watch?v=NdLPpxILVYg

5.4. Interazione con mosquitto

La libreria è compatibile con mosquitto e con chi altro rispetta lo standard MQTT v3.1.1, pur non implementando tutti gli aspetti quali ad esempio la QoS=2.

Link YouTube che mostra la compatibilità con mosquitto:

https://www.youtube.com/watch?v=rujloPt8jlU

5.5. Api disponibili

- `PablittoMQTT(string ipv4_broker, int port);`
  Costruttore del PablittoMQTT.
- `PablittoMQTT(int port);`
  Costruttore del PablittoMQTT, di default la porta è 1883.
- `void start(bool debug=false);`
  Avvia PablittoMQTT e quindi i suoi rispettivi thread.
- `void stop();`
  Stoppa PablittoMQTT e quindi i suoi rispettivi thread
- `byte_t connectMQTT(std::string client_id, uint16_t keep_alive);`
  Ritorna il codice di risposta del broker alla connessione MQTT.
- `byte_t connectMQTT(std::string client_id, std::string username,vector<byte_t> password, uint16_t keep_alive);`
- `byte_t connectMQTT(std::string client_id, std::string username,vector<byte_t> password, std::string will_topic, bool will_flag,vector<byte_t> will_message, bool will_retain=false, uint16_t keep_alive=0x60, byte_t will_qos=AT_MOST_ONCE);`

- `uint16_t publish(std::string topic_name, std::vector<byte_t> message, bool retain, byte_t qos);`
  Effettua una publish, è possibile specificare la qos desiderata e il flag retain. Ritorna il

message_id del control_packet. Il parametro qos è opzionale, se assente assume valore AT_MOST_ONCE.

- `uint16_t subscribe(std::string topic_name, byte_t qos);`
  Effettua una subscribe, è possibile specificare il topic_name e la qos desiderata.
- `uint16_t subscribe(std::vector<string> topic_name_vector, std::vector<byte_t> qos_vector);`
  Effettua una subscribe multipla, è possibile specificare un insieme di topic a cui iscriversi con le qos corrispondenti. Ritorna il message_id del control_packet.
- `uint16_t unsubscribe(std::string topic_name);`
  Effettua una unsubscribe al topic specificato. Ritorna il message_id del control_packet.
- `uint16_t unsubscribe(std::vector<string> topic_name_vector);`
  Effettua una unsubscribe multipla ai topic specificati.
- `bool is_valid_topic(const std::string str);`
  Verifica che il topic inserito in una publish non contenga wildcard.
- `void disconnect();`
  Disconnessione.
- `std::pair <std::string, std::vector<byte_t>> get_message();`
  Ritorna un pair, il primo elemento rappresenta il topic, il secondo elemento rappresenta il value ricevuto.

5.6. Come utilizzare la libreria statica libpablitto.a

Il progetto è stato gestito tramite Cmake per la compilazione e il linking dei file.

Una volta installato Cmake (versione minima richiesta 3.13), si apra un terminale e si digiti:

git clone https://github.com/antromeo/pablitto-client.git

Si crei la build e si creino gli eseguibili di pablitto_sub e pablitto_pub con i seguenti comandi:

cd pablitto_client

mkdir cmake-debug-files

cd cmake-debug-files

make ..

Una volta lanciati questi comandi otteniamo la libreria statica libpablitto.a e gli applicativi pablitto_sub e pablitto_pub. Per utilizzare la libreria, la creazione di una semplice app può essere realizzata includendo il file pablittoMQTT.hh e linkando la libreria statica libpablitto.a. Un esempio con gcc assumendo di avere nella directory corrente i file .hh (directory include) e il file libpablitto.a è il seguente:

g++ your_app.cpp -o -L . -lpablitto

6. pablitto-broker

Link su github: https://github.com/antromeo/pablitto-broker.git

pablitto-broker è sicuramente una semplificazione di un broker MQTT, sono state implementate solo le funzioni necessarie ad un corretto smistamento e riconoscimento dei control packet MQTT.

L'applicativo si compone dei seguenti file:

- fixed_headr.go, connack_packet.go, connect_packet.go, control_packet.go, pingreq_packet.go, puback_packet.go, publish_packet.go, puback_packet.go, subscribe_packet.go, unsuback_packet.go, disconnect_packet.go, , suback_packet.go, unsubscribe_packet.go, utilities.go, subscription.go. Questi si trovano nel package control_packet.
- broker.go.

Per avviare il broker, basta digitare nella directory corrispondente go build broker e ./broker numero_porta.

Link tutorial YouTube:

https://www.youtube.com/watch?v=shWXoqBa2OY&t=7s

6.1. Interazione con mosquitto

Anche il broker interagisce senza problemi con mosquitto. Per visionare una dimostrazione si rimanda al link su YouTube sottostante.

Link tutorial YouTube:

https://www.youtube.com/watch?v=rujloPt8jlU