ESERCITAZIONE

Esempio di partenza:

Apriamo due terminali nella cartella che contiene il file .perm per avere l'autorizzazione per collegarsi al servizio offerto dall'università. In uno digitiamo

'*mosquitto_sub -t temperatura -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d*' ,

mentre nell'altra

'*mosquitto_pub -t temperatura -m "25" -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d*' .

Il primo è il subscriber, che ritorna il seguente output:

- ➔ Client null sending CONNECT
- ➔ Client null received CONNACK (0)
- ➔ Client null sending SUBSCRIBE (Mid: 1, Topic: temperatura, QoS: 0, Options: 0x00)
- ➔ Client null received SUBACK
- ➔ Subscribed (mid: 1): 0
- ➔ Client null received PUBLISH (d0, q0, r1, m0, 'temperatura', ... (4 bytes))
- ➔ halo

il terminale rimane in esecuzione in attesa di pubblicazioni da parte del publisher sul topic desiderato. In tutto questo, null è l'id del subscriber (in questo caso non ne ha uno, per questo null). Con Connect si connette, con connack si conferma la connessione (ACK, 0). Nella terza riga ci si iscrive al topic 'temperatura', con id iscrizione (Mid:1) e QoS a 0, ossia quality of service di livello 0 (non è richiesta la conferma di ricezione da parte del broker). Con SubACK il broker risponde ACK alla richiesta di sottoscrizione. Segue poi una notifica di pubblicazione da parte del broker, che avverte il subscriber che è stato pubblicato qualcosa nel topic; d0 indica che il messaggio non è un duplicato, q0 che viene inviato con metodo QoS impostato a 0, e r1 indica (IMPORTANTE) che il messaggio NON rappresenta una nuova pubblicazione, ma una precedente al collegamento del subscriber e che è stato programmato per essere pubblicato all'avvio. In particolare, si tratta di un messaggio di peso 4 bytes, opportunamente stampato nella riga seguente ('halo'). Alla terminazione forzata del processo (kill -9 o ctrl^C), avviene la disconnessione con relativo messaggio sul terminale ('Client null sending DISCONNECT').

Per il publisher invece abbiamo quanto segue:

- ➔ Client null sending CONNECT

- ➔ Client null received CONNACK (0)
- ➔ Client null sending PUBLISH (d0, q0, r0, m1, 'temperatura', ... (8 bytes))
- ➔ Client null sending DISCONNECT

Il processo a terminale viene terminato una volta pubblicato il dato richiesto. Nel terminale del subscriber riceveremo il dato “64”, mentre qui, oltre a CONNECT E CONNACK, abbiamo l’effettiva pubblicazione del dato (da notare flag r0 dato che si tratta di una nuova pubblicazione), e la successiva disconnessione del publisher.

N.B: vedere in automatico nel terminale del subscriber righe come ‘Client null sending PINGREQ’ oppure ‘Client null received PINGRESP’ è del tutto normale. Il primo è un messaggio dal subscriber verso il broker per fargli sapere che è ancora attivo e connesso, mentre il secondo è la relativa risposta del broker, che gli fa sapere che anche lui è ancora bello arzillo.

Esercizio 1:
Partendo dall’esercizio mostrato nelle istruzioni, cosa succede se pubblico una temperatura prima di aver lanciato il subscriber? Provare con l’opzione --retain

RISPOSTA:
Se avviamo prima il publisher e poi il subscriber, come detto prima il publisher chiude subito dopo la pubblicazione degli argomenti. Il subscriber quindi, una volta iscritto, non vedrà la nuova pubblicazione perché antecedente la sua stessa iscrizione.

Aggiungendo la flag --retain al publisher e sempre eseguendolo per primo, il messaggio di pubblicazione di “benvenuto” che ho descritto durante l’esempio di prova viene rimpiazzato da questa effettiva pubblicazione.

Esercizio 2:
Partendo dall’esercizio mostrato nelle istruzioni creare un’applicazione pub/sub con 2 sensori di temperatura relativi a 2 stanze diverse. Quante finestre di terminale devo aprire?

RISPOSTA:
Per poter fare questa cosa, ci basta creare due diversi publisher (quindi un terminale a processo), e modificare il topic in cui pubblicano i dati. In uno specifichiamo il topic ‘temperatura/sensore1’, mentre nell’altro ‘temperatura/sensore2’. In questo modo siamo sempre sotto lo stesso macro-topic, ossia la temperatura, ma differenziamo le pubblicazioni

riguardanti due sensori diversi. Sempre mantenendo un senso logico quindi, è possibile modularizzare a piacimento le pubblicazioni e le sottoscrizioni dei vari client.

Quindi, per avere due topic diversi servono chiaramente due publisher, e quindi 2 terminali. Poi sulla base di quanti subscriber vogliamo avere, aggiungiamo altre istanze del terminale. Nel mio caso ho fatto 2 subscriber, uno per publisher, quindi altri 2 terminali (4 in totale).

Esercizio 3:
Partendo dall'esercizio precedente come fare per avere un unico subscriber per entrambe le temperature? Come si fa a distinguere da quale stanza proviene la temperatura?

RISPOSTA:
Risposta banale. Invece di specificare il percorso esatto che porta al topic desiderato come nell'esercizio precedente, andiamo a usare un pattern di percorso che ci permette di indicare la sottoscrizione all'intero macro-topic. Quindi, nel subscriber, invece di scrivere 'temperatura/sensore<N-esimo>' andiamo a scrivere 'temperatura/#'.

In questo specifico caso di esecuzione ci vengono quindi forniti in output tutti i caricamenti da parte di publisher all'interno del macro-topic 'temperatura'. Questo chiaramente comprende anche topic che non ho creato io, come 'casa', 'cucina' o altro ancora. Se avessimo voluto creare una sessione personalizzata al 100%, ossia con topic creati solo da me, avrei dovuto creare un ulteriore sottocartella in cui inserire i sensori, e usare il pattern # al suo interno.

Come spiegato nell'esempio all'inizio, prima di stampare il contenuto del messaggio del publisher, il broker ci manda anche delle informazioni riguardanti la pubblicazione stessa. Oltre alla natura del messaggio (r0 nuova pub, r1 pub precedente), possiamo vedere in chiaro per quale sotto-topic di temperatura è stata fatta la pubblicazione.

Esercizio 4:
Prova a pubblicare un valore di umidità relativa (topic "UR"); il subscriber interessato alle temperature lo riceve? Come si fa a creare un subscriber interessato all'umidità? Costruire un'applicazione pub/sub con 4 finestre per produrre e visualizzare sia valori di temperatura sia valori di umidità.

RISPOSTA:
Chiaramente il sub iscritto alla temperatura non riceve notifiche riguardanti l'umidità, così come vale il contrario. Per invece avere un sub in grado di ricevere entrambe, ci basta usare la wizard +/UR e +/temperatura, che permette l'iscrizione a due sotto-topic appartenenti a due macro topic diversi. Abbiamo quindi due publisher, uno che pubblica un dato in /UR, e l'altro in /temperatura (molto importanti gli '/'). Con il subscriber invece, andiamo a iscriverci a entrambi con (appunto specificando -t +/UR seguito da -t +/temperatura)

mosquitto_sub -t +/temperatura -t +/UR -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

Esercizio 5:
Aprire e studiare con Wireshark il file PCAP contenuto nello ZIP:

- quale protocollo di livello trasporto utilizza MQTT? Quali sono le porte?
- trovare le fasi di publish e subscribe. Quante connessioni TCP apre un subscriber? Quante connessioni TCP apre un publisher?

RISPOSTA:
A livello di trasporto, MQTT utilizza TCP, usufruendo in questo caso specifico delle porte 1883 e 60070.

Per un client MQTT (sia esso un subscriber o un publisher), la comunicazione con il broker avviene generalmente tramite una singola connessione TCP persistente. Questa connessione viene mantenuta attiva per tutta la durata della sessione, permettendo scambi bidirezionali efficienti. In WSH, il subscriber apre una sola connessione TCP con il broker per connettersi, iscriversi ai topic e ricevere tutti i messaggi. Il publisher invece apre una connessione TCP con il broker per ogni pubblicazione, dato che, come abbiamo visto negli esercizi precedenti, dopo la pubblicazione, chiude.

Esercizio 6:
Si vuole costruire con MQTT un servizio di messaggistica universitaria:

- il rettore può leggere tutti i messaggi
- la segreteria può leggere i messaggi dai docenti e dagli studenti
- i docenti possono leggere i messaggi dai docenti e dagli studenti
- gli studenti possono leggere solo i messaggi degli altri studenti

RISPOSTA:

TOPIC DI PUBBLICAZIONE:

- Docenti:

mosquitto_pub -t Università/docenti -m “messaggioDaDocente” -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

- Studenti:

mosquitto_pub -t Università/studenti -m “messaggioDaStudente” -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

Assunto, poiché non richiesto, che segreteria e rettore solo leggano in questa applicazione.

TOPIC DI ISCRIZIONE:

- Rettore:

mosquitto_sub -t Università/# -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

- Segreteria:

mosquitto_sub -t Università/docenti -t Università/studenti -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

Per come è stata specificata l’applicazione, il rettore e la segreteria leggono gli stessi gruppi alla fine. Per come sono state scritte le sottoscrizioni tuttavia, nel caso venisse aggiunto un nuovo gruppo di pubblicazione, il rettore ne sarebbe iscritto automaticamente.

- Docenti:

mosquitto_sub -t Università/docenti -t Università/studenti -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

- Studenti:

mosquitto_sub -t Università/studenti -h 90.147.167.187 -p 8883 -u univr-studenti -P MQTT-esercitazione2024 --cafile ISRG_Root_X1.pem --insecure -d

Volendo provare altro, basta usare i due pub aperti per provare questa parte (2 per pub, 4 per sub) e pubblicare in una cartella diversa da docente/studente per vedere che il rettore riceve comunque.