CONOSCERE E USARE

Node-RED

di FRANCESCO FICILI

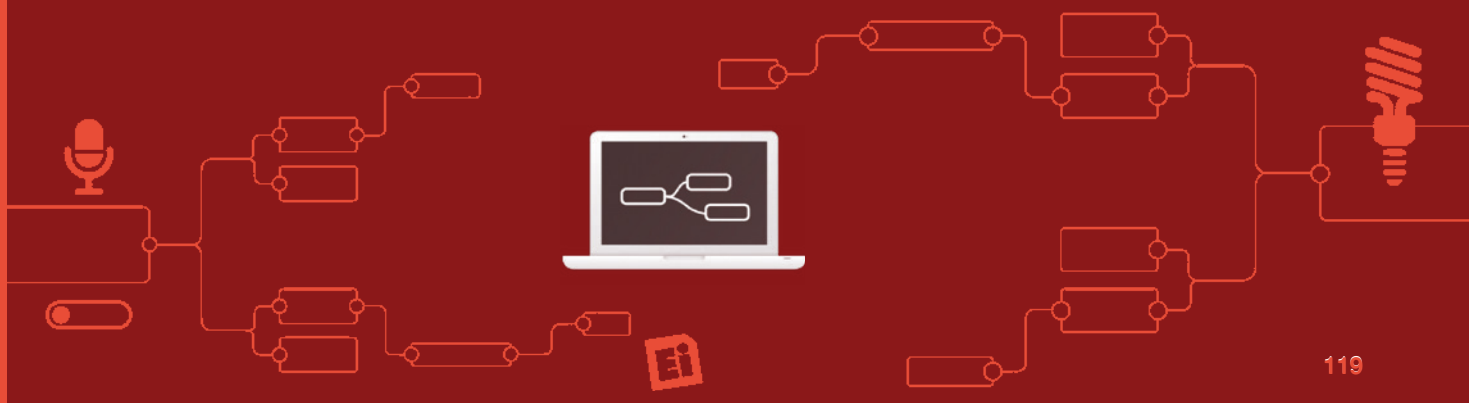
6

Impariamo ad usare Node-RED, un tool di flow-based programming orientato all'IoT ed alla connettività, originariamente sviluppato dall'IBM Emerging Technology Services team e adesso parte della JS Foundation. In questa puntata continuiamo a parlare di connettività introducendo i websockets e il protocollo MQTT. Sesta Puntata.

Come abbiamo potuto vedere nella scorsa puntata, Node-RED è un tool con una forte predisposizione allo sviluppo di applicazioni connesse. Infatti, tra i nodi presenti di default esiste un'ampia sezione dedicata alla connettività, identificata dal gruppo di nodi Network.
Nella scorsa puntata ci siamo addentrati nello studio dei nodi per la gestione di comunicazioni via UDP, TCP e http. Tuttavia, in tempi più recenti, l'avvento delle applicazioni IoT ha visto sempre di più l'affermazione di tecnologie di comunicazione specializzate, capaci di soddisfare i requisiti specifici e di superare le limitazioni imposte da quella classe di dispositivi che sono al centro del mondo dell'internet degli oggetti, ossia i dispositivi embedded. Tali tecnologie frequentemente continuano ad appoggiarsi su TCP/IP come livello di trasporto, ma offrono una implementazione a livello applicativo (Livello 7 OSI) più consona a quelle che sono le esigenze dei dispositivi embedded di quanto non faccia un protocollo come http, che è più pensato per il world wide web e la comunicazione tra browser e server di quanto non lo sia per la comunicazione machine-to-machine.
Ci sono diverse tecnologie di comunicazione che stanno cercando di imporsi nel panorama IoT in questi anni; in particolare, in questa puntata, noi ci concentreremo su due meccanismi: i websockets ed il protocollo MQTT.

WEBSOCKET

Una delle caratteristiche del protocollo http che abbiamo analizzato nella scorsa puntata è quella di ridurre il numero di connessioni aperte a quelle strettamente necessarie. Infatti in una tipica transazione dati http il client invia la richiesta, che viene servita dal server con una appropriata risposta, dopodiché la connessione tra client e server viene



generalmente chiusa e per essere nuovamente aperta il client deve necessariamente inviare una nuova richiesta. Questo comportamento è ideale per il Wordl Wide Web, in cui le pagine dei siti spesso contengono dei collegamenti a pagine ospitate da altri server, diminuendo così il numero di connessioni attive a quelle effettivamente necessarie. Tuttavia questa tipologia di funzionamento cosiddetto connection-less introduce delle latenze nello scambio dati e può risultare meno adatto ad applicazioni in cui la responsività sia importante. Per ovviare a questo limite sono state sviluppate tecnologia alternative, come appunto i Websockets.
Il protocollo websocket è una tecnologia web che permette di implementare canali di comunicazione full-duplex usando una singola connessione, che viene mantenuta aperta, tipicamente basata sul protocollo TCP. Questa tecnologia facilita l'interazione tra client e server, facilitando la realizzazione di applicazioni in cui la componente real-time riveste una certa importanza.
Come anche il protocollo http, il protocollo websocket si posiziona al livello 7 della pila protocollare OSI, quindi è anch'esso un protocollo di livello applicazione.

WEBSOCKET IN NODE-RED

Node-RED offre nativamente due nodi per la gestione della comunicazione tramite websocket, rappresentati in Fig. 1.

- *websocket in*: questo nodo permette di ricevere messaggi tramite protocollo websocket. Una volta che il nodo ha ricevuto il messaggio lo invia su *msg.payload*.
- *wesocket out*: questo nodo permette di inviare messaggi tramite protocollo websocket. Il nodo invia il nodo contenuto in *msg.payload* di qualsiasi messaggio in ricezione.

UN ESEMPIO DI UTILIZZO DEI WEBSOCKET

Vediamo adesso un semplice esempio di utilizzo del protocollo websocket con Node-RED. Ciò che ci proponiamo di realizzare è una semplice comunicazione machine-to-machine usando il protocollo websocket. Come fatto in precedenza per gli esempi con TCP ed UDP, useremo una Raspberry Pi e il nostro PC per ospitare le due istanze di Node-RED che dovranno comunicare via websocket. Oltre alla Raspberry Pi e al PC ci occorrerà anche un server esterno che faccia da mediatore nello scambio dati. A questo server la Raspberry Pi si collegherà per inviare i dati, mentre il PC si metterà in

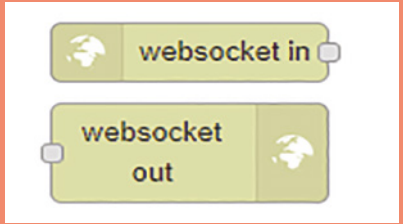


Fig. 1 - Nodi Websocket in Node-RED.

Fig. 2 - PieSocket test server.

ascolto per riceverli. Come server di test abbiamo deciso di utilizzare un servizio offerto da PieSocket, che è possibile trovare a questo indirizzo: *https://www.piesocket.com/websocket-tester#*.
In Fig. 2 è possibile vedere la schermata in cui è rappresentata la console di controllo del server di test. Nella barra della URL, dopo la parte fissa *www://demo.piesocket.com/v3/* è possibile inserire un path interno che verrà utilizzato per lo scambio dati (nel nostro esempio */ElettronicaIn*), e successivamente è presente l'hash della API key utilizzata per la connessione. Copiate l'intera stringa e salvatela da qualche parte, ci servirà in seguito.
Partiamo dalla creazione del flow sulla Raspberry Pi, ossia quello che trasmette il dato. Preleviamo dalla palette il solito nodo *Inject* ed un nodo *Websocket out* e posizioniamoli sul workspace. Possiamo configurare il nodo inject per inviare una qualsiasi stringa (nel nostro esempio la stringa "Ciao da Raspberry!!!"). Dopo aver fatto questa semplice operazione apriamo la finestra delle proprietà del nodo websocket out, che possiamo vedere rappresentata in Fig. 3.
A questo selezioniamo dal dropdown menu Type la voce "Connect To" e clicchiamo sull'cona a forma di matita per configurare il websocket client. La configurazione del websocket client è riportata in Fig. 4.
Qui è sufficiente copiare nel campo URL la stringa precedentemente salvata (che contiene l'indirizzo del server, il path interno e l'API key). Salviamo e chiudiamo. Per evitare che il nodo venga rappresentato con un'icona eccessivamente lunga nel workspace possiamo anche dare un nome al nodo websocket out (nell'esempio lo abbiamo rinominato Websocket Test). A questo punto non resta altro da fare che connettere l'uscita del nodo inject all'ingresso del nodo websocket out, e il flow del nostro nodo trasmettitore è completo.
Passiamo adesso al nodo ricevitore, implementato sul PC (ovviamente sarebbe anche possibile sostituire questo nodo con una seconda Raspberry Pi). I nodi che ci occorrono in questo flow sono un nodo websocket in ed un nodo debug. La configurazione del nodo websocket in (Fig. 5) è del tutto simile a quella dal nodo websocket out, con la differenza che

questa volta il campo type dovrà essere settato al valore "Listen on", in quanto il nodo è in ascolto.
Anche in questo caso andrà creato un configuration node (per il listener questa volta), la cui finestra delle proprità è riportata in Fig. 6.
A questo punto non ci resta che collegare l'output del nodo websocket in al nodo debug e anche il flow del nodo ricevitore è completo. Ora possiamo testare la nostra implementazione, cliccando sul pulsante integrato del nodo inject per inviare il contenuto del suo campo payload dal nodo trasmettitore al nodo ricevitore via websocket. Se abbiamo fatto tutto correttamente, dal lato del ricevitore vedremo comparire sulla finestra di debug le stringhe rappresentate in Fig. 7.

MQTT

Il protocollo MQTT (Message Queue Telemetry Transport) è un protocollo di messaggistica, originariamente creato da Andy Stanford-Clark e Arlen Nipper nel 1999. MQTT è un protocollo estremamente leggero e che utilizza un pattern di tipo pubblicazione-sottoscrizione (publish-subscribe o PubSub), ossia non esiste una connessione diretta tra in nodi che si scambiano un messaggio, ma piuttosto esistono una serie di nodi client che pubblicano su o richiedono dati da una terza parte, denominata normalmente broker.
Come http e websocket, MQTT è un protocollo di livello applicazione e sfrutta un livello trasporto basato su TCP/IP. Per via delle sue caratteristiche, il protocollo MQTT è molto utilizzato in ambito IoT, in quanto è molto leggero e veloce, e quindi adatto ad essere implementato anche in sistemi embedded con risorse limitate, ancorché su smartphone e server. Ne è stata sviluppata anche una variante, denominata MQTT-SN, destinata a sistemi embedded che per il networking non fanno uso di un livello trasporto basato su TCP/IP, ma su protocolli alternativi come 802.15.4 o ZigBee. Recentemente alcune importanti aziende hanno affermato di fare uso di MQTT nelle loro implementazioni, ed uno degli esempi più rilevanti è costituito da Facebook Messenger, che ha fatto uso di MQTT per aumentare le performance in termini di velocità e occupazione di banda.

IL PATTERN PUBLISH-SUBSCRIBE

Come abbiamo già accennato in precedenza, il protocollo MQTT è basato su un modello protocollare di tipo pubblicazione-sottoscrizione. In sostanza vengono definite due entità della rete di telecomunicazione: un certo numero di client e un message broker. I client sono tipicamente i dispositivi terminali della rete, ossia i reali produttori e consumatori dei

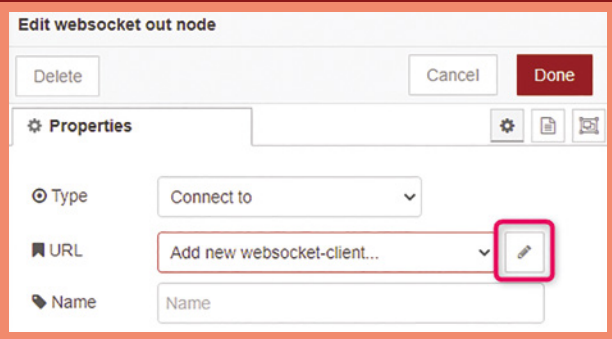


Fig. 3 - Finestra proprietà del nodo websocket out.

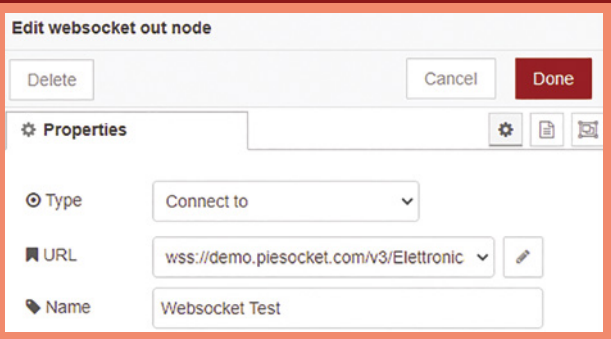


Fig. 5 - Configurazione del nodo websocket in.

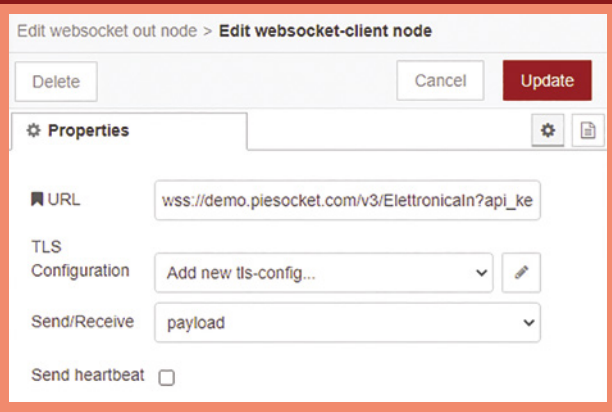


Fig. 4 - Configurazione del websocket client node.

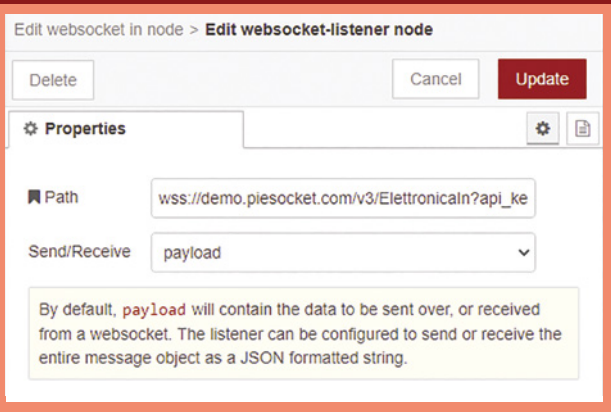


Fig. 6 - Finestra proprietà del configuration node websocket-listener.

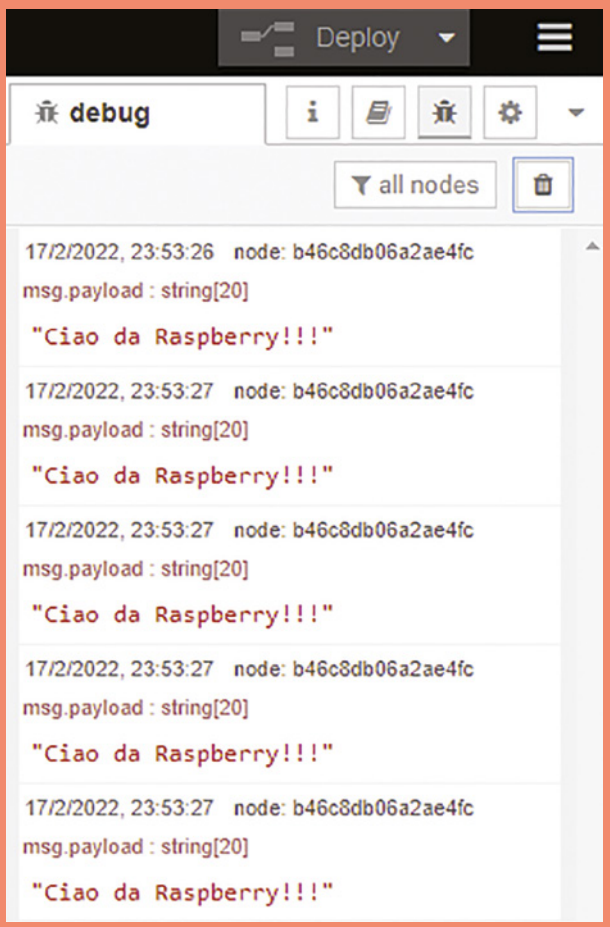


Fig. 7 - Finestra di debug del nodo ricevitore.

Fig. 8 - Logo del protocollo websocket.

messaggi (tipicamente nodi contenenti sensori o attuatori), mentre il broker non è altro che un server dedicato che riceve tutti i messaggi dai vari client che intendono comunicare e li inoltra ai vari client che invece sono interessati a riceverli. Quindi, usando il protocollo MQTT, se un ipotetico nodo A volesse comunicare un messaggio su uno specifico argomento (comunemente detto topic) ad un ipotetico nodo B, non sarebbe necessario stabilire una connessione diretta e sincrona tra i due nodi (come avviene invece in http tra client e server), ma il nodo A pubblica semplicemente il messaggio con il topic specifico (ad esempio la lettura di un determinato sensore) sul message broker (che ha il compito specifico di ricevere tali messaggi) e sarà poi il broker, in maniera asincrona rispetto al client che trasmette, ad inviare lo specifico messaggio al client interessato (in questo caso il nodo B). Si intuisce anche che il nodo interessato (ossia sottoscrivente) ad uno specifico topic non debba essere necessariamente un client singolo, ma il broker potrebbe inoltrare il messaggio anche ad un gruppo di nodi sottoscriventi lo stesso topic (quindi è nativamente supportato anche il broadcasting). La Fig. 9 schematizza il modello di comunicazione publisher-subscriber.

PROTOCOLLO MQTT

Il protocollo MQTT è stato sin da subito pensato per la comunicazione machine-to-machine, quindi ha una serie di caratteristiche che lo rendono intrinsecamente adatto ad essere utilizzato in ambito IoT. Infatti si tratta di un protocollo estremamente leggero e poco verboso rispetto ad altri protocolli di rete come ad esempio l'http, e quindi è molto indicato per essere utilizzato in dispositivi con poche risorse in termini di memoria, capacità di calcolo e banda. Ciononostante il protocollo è in grado di garantire anche un livello di servizio affidabile, quando quest'ultimo sia una caratteristica significativa dell'applicazione e si sia disposti a sacrificare banda trasmissiva per ottenerla. Infatti in MQTT sono previsti tre livelli di qualità del servizio o QoS:

- QoS 0 o At most once: questa è la più bassa qualità del

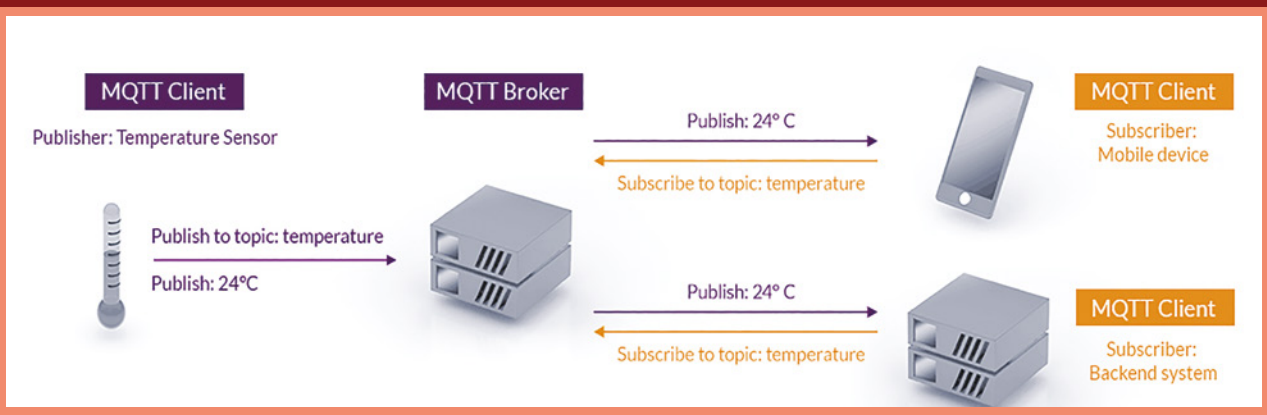


Fig. 9 - Schematizzazione del modello pubblicazione-sottoscrizione utilizzato dal protocollo MQTT.

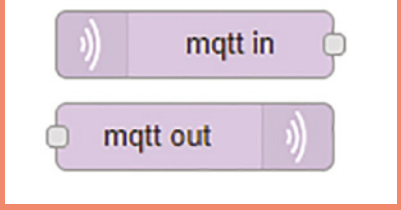


Fig. 10 - Nodi MQTT in Node-RED.

servizio prevista dal protocollo. Il messaggio viene trasmesso una singola volta, senza garanzia di consegna. Il destinatario del messaggio non conferma la ricezione ed il massaggio non è memorizzato e ritrasmesso dal mittente. Questo livello del servizio è spesso denominato "fire and forget" e sostanzialmente offre le stesse garanzie di consegna del protocollo TCP sottostante.

- **QoS 1** o **At least once**: questo livello del servizio garantisce la consegna del messaggio almeno una volta al nodo destinatario. Infatti il mittente continua ad inviare il messaggio finchè non riceve una conferma di ricezione da parte del destinatario. Tuttavia, con questa metodologia, esiste la possibilità che il messaggio venga ricevuto dal destinatario più di una volta.
- **QoS 2** o **Exactly once**: questo è il più alto livello di servizio raggiungibile in MQTT: il messaggio viene consegnato esattamente una volta al destinatario.
  Tramite uno scambio protocollare più complesso tra nodo trasmettitore e nodo ricevitore, viene confermata la ricezione e c'è la certezza che lo specifico messaggio sia stato ricevuto una sola volta.

Il protocollo dispone inoltre di una serie di messaggi "speciali"; questi messaggi possono essere utilizzati per attuare delle politiche di gestione della rete al verificarsi di determinate condizioni. Essi sono:

- **Birth**: questo messaggio viene pubblicato quando viene stabilita la connessione.
- **Close**: questo messaggio viene pubblicato prima che la connessione venga chiusa normalmente (ad esempio perché un nodo viene ri-deployato o spento).
- **Will**: questo messaggio viene pubblicato su un nodo specifico nel caso in cui un nodo perda inaspettatamente la connessione.

Infine il protocollo MQTT supporta anche un certo livello di security infatti è anche possibile trasmettere username e password tramite i suoi pacchetti ed è possibile cifrare le connessioni tramite l'uso di SSL, sebbene questa caratteristica appesantisca il protocollo.

MQTT IN NODE-RED

Node-RED dispone nativamente di due nodi per la gestione della comunicazione tramite protocollo MQTT, illustrati in Fig.10.

- **mqtt in**: questo nodo permette di connettersi ad un broker MQTT e di effettuare il subscribe su un topic specifico.
- **mqtt out**: questo nodo permette di connettersi ad un broker MQTT e di pubblicare su un topic specifico.

Inoltre è presente un configuration node in cui l'utente può impostare la configurazione del broker.
In particolare è possibile configurare una serie di opzioni per le seguenti categorie:

- **Connection**: permette di configurare l'indirizzo IP del broker, la porta alla quale connettersi, se usare o meno la security ed anche la specifica versione del protocollo da utilizzare.
- **Security**: se si intende utilizzare una connessione sicura, in questa sezione è possibile configurare username e password da utilizzare.
- **Messages**: in questa sezione è possibile configurare i messaggi di birth, close e will (in termini di topic e payload da utilizzare, QoS e politica di retain).

In Fig.11 è riportata a schermata di configurazione del nodo configuration mqtt-broker (tab connection).

MQTT BROKER IN NODE-RED

Come si è potuto intuire dalle sezioni precedenti, per poter far funzionare una applicazione MQTT con Node-RED ci occorre comunque un Broker. Esistono diverse implementazioni di Broker MQTT disponibili ad oggi sul mercato, incluse soluzioni free estremamente performanti. Un esempio abbastanza noto è costituito da Mosquitto, un broker MQTT sviluppato on collaborazione con la eclipse foundation.
Può essere scaricato al seguente indirizzo web:

https://mosquitto.org

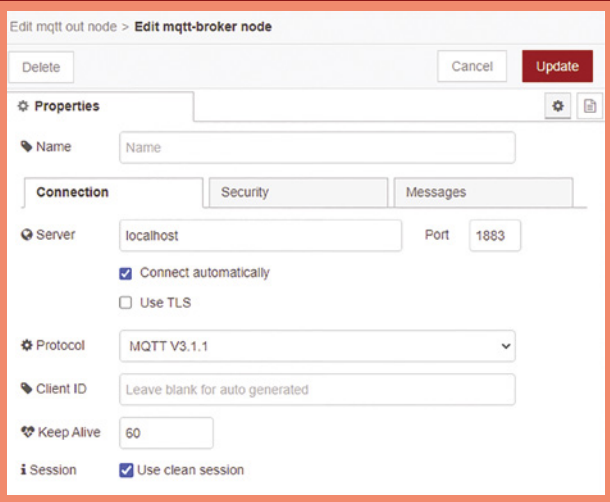


Fig. 11 - Schermata di configurazione del nodo mqtt-broker.

Fig. 12 - Nodo Aedes.

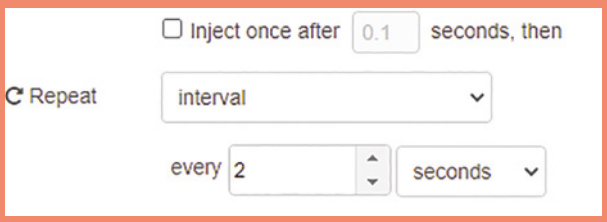


Fig. 13 - Configurazione della periodicità del nodo Inject.

Esiste in diverse versioni per differenti sistemi operativi, come Linux, Mac e Windows. Mosquitto è un broker MQTT estremamente performante ed è sicuramente la soluzione più indicata per implementazioni con molti nodi e con un traffico dati importante. Esistono tuttavia anche implementazioni più "light", sottoforma di nodi Node-RED che possono essere facilmente installate direttamente dalla palette di Node-RED, e possono essere utilizzate nelle situazioni più semplici o anche a scopo di test. Una valida implementazione di questo tipo è costituita da Aedes, un semplice broker MQTT installabile all'interno di Node-RED sottoforma di nodo (Fig. 12). Maggiori informazioni possono essere reperite al seguente indirizzo web:

https://flows.nodered.org/node/node-red-contrib-aedes

Per installarlo basta cercare la parola chiave "Aedes" tramite la palette di Node-RED ed installare il nodo. L'utilizzo è semplicissimo, sarà sufficiente trasportare il nodo "Aedes Broker" dalla palette al workspace ed eseguire un deploy, al termine del quale il broker sarà già funzionante.

UN ESEMPIO DI UTILIZZO DI MQTT

Passiamo adesso all'esempio pratico di questa puntata. Ciò che ci proponiamo di realizzare è un semplice sistema di comunicazione basato su MQTT in cui un nodo publisher (che implementeremo con una istanza di Node-RED su PC) invia periodicamente lo stato di un LED and un nodo subscriber che implementeremo con una Raspberry Pi che lo attua utilizzando un GPIO. Questo stato vogliamo che si alterni tra True e False alla frequenza di 1Hz; in sostanza vogliamo far lampeggiare il LED sulla nostra Raspberry Pi via MQTT. Inoltre decidiamo che il broker sarà implementato tramite il nodo Aedes sul nodo subscriber, ossia sulla Raspberry Pi (occorre quindi conoscere l'indirizzo IP di questo dispositivo sulla rete locale). Partiamo dall'implementazione del nodo publisher, che nel nostro esempio sarà implementato tramite una istanza di Node-RED basata su PC. Se non ricordate come installare Node-RED su PC (sia esso Linux, Mac o Windows), potete consultare la sezione "running locally" della getting started guide di Node-RED al seguente indirizzo:

https://nodered.org/docs/getting-started/local

Avere una installazione locale, se non avete a disposizione molte Raspberry Pi, può essere un ottimo sistema per eseguire i vostri test di connettività, l'importante è che il PC e la Raspberry Pi stiano sulla stessa rete locale. Il nodo subscriber ha il compito di inviare periodicamente il nuovo stato del LED, alla frequenza di 1Hz. Se ricordate, proprio all'inizio del corso Node-RED abbiamo visto come far lampeggiare un LED alla frequenza desiderata, e ciò che ci serve è un nodo *Inject* ed un nodo *Trigger*. Inseriamo sul nostro workspace anche un nodo *mqtt out*, che ci servirà in seguito. Configuriamo il nodo Inject per inviare il suo payload ad un intervallo di 2 secondi, come illustrato in Fig.13, mentre il nodo trigger va configurato per inviare prima il valore booleano true, aspettare per 1 secondo e poi inviare il valore false, come illustrato in Fig.14. Ora non ci resta che configurare il nodo mqtt out. Clicchiamo due volte sul nodo stesso per aprire la sua finestra delle proprietà ed impostiamo:

- **Server**: clicchiamo sull'icona a forma di matita per aprire la configurazione del broker. Nel campo Server dobbiamo inserire l'indirizzo IP della Raspberry Pi, che conterrà il broker MQTT. Lasciamo il resto della configurazioni ai valori di default e clicchiamo su Update.
- **Topic**: Led

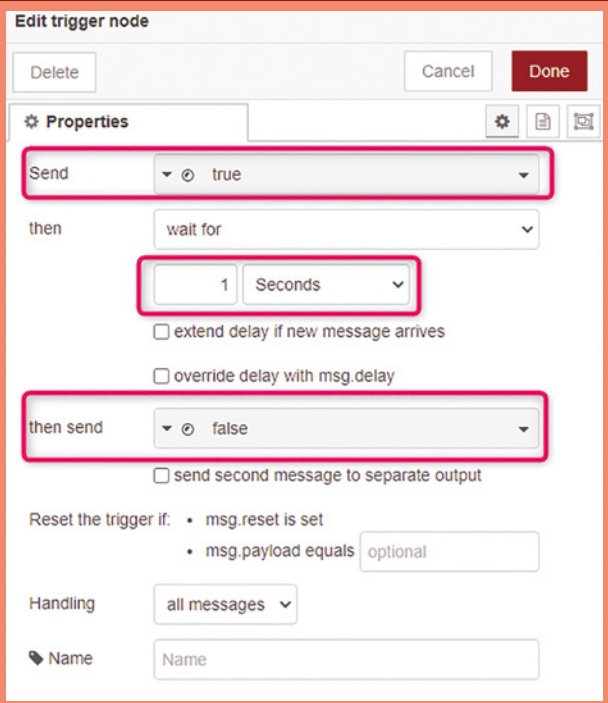


Fig. 14 - Configurazione del nodo trigger.

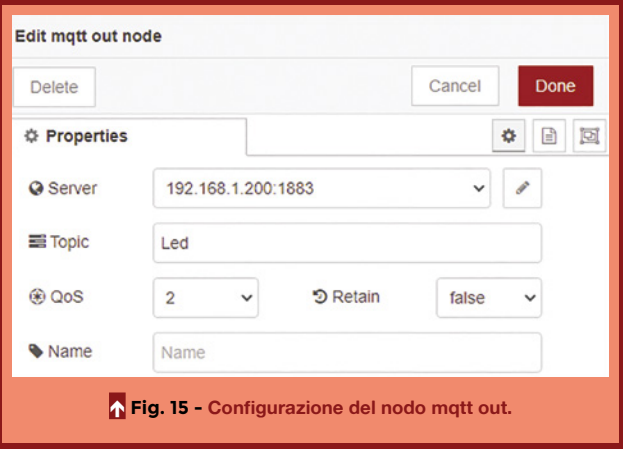


Fig. 15 - Configurazione del nodo mqtt out.

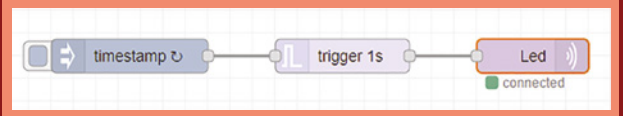


Fig. 16 - Flow che implementa il nodo publisher.

- QoS: 2
- Retain: false

Se tutto è stato fatto correttamente, dovremmo vedere una finestra delle proprietà come quella di Fig.15 (naturalmente l'indirizzo del server sarà uguale all'indirizzo della Raspberry Pi nella rete locale, nel nostro esempio è 192.168.1.200).
A questo punto connettiamo tra di loro i nodi come illustrato in Fig.16.
L'implementazione del nodo subscriber è ancora più semplice. Ci occorre un nodo Aedes, un nodo mqtt in ed un nodo rpi gpio out. Collochiamo questi tre nodi sul workspace ed iniziamo a configurare il nodo mqtt in. Anche in questo caso va configurato il configuration node mqtt broker, ma stavolta possiamo comodamente inserire "localhost" sul campo server, in quanto il broker MQTT è presente sullo stesso dispositivo (più precisamente sulla stessa istanza di Node-RED). Il resto delle configurazioni sono identiche a quelle viste per il broker del nodo publisher, quindi in questo caso dovreste ritrovarvi con una finestra delle proprietà identica a quella di Fig.17.
A questo punto passiamo alla configurazione del nodo rpi gpio out; in questo caso vogliamo utilizzare il pin 12 (GPIO 18) per il controllo del LED, quindi impostiamo questo GPIO come digital output ed inizializziamolo a 0 come stato logico, come abbiamo visto nelle prime puntate, in cui abbiamo imparato a gestire il GPIO del Raspberry Pi tramite Node-RED.
Il nodo Aedes non ha necessità di particolari configurazioni, ci basterà posizionarlo da qualche parte nel workspace. A questo punto colleghiamo la porta di uscita del nodo mqtt in alla porta di ingresso del nodo rpi gpio out ed eseguiamo il deploy sia del nodo publisher che del nodo subscriber. Una volta che i due nodi saranno in esecuzione vedremo lampeggiare il LED collegato sulla Raspberry Pi alla frequenza impostata.
Come hardware per l'esempio pratico di questa puntata possiamo usare lo shield Raspberry Pi I/O (cod. *FT1060*), distribuito da Futura Elettronica, una cui immagine è visibile in Fig. 18.
Questo semplice esempio mostra quanto sia agevole utilizzare il protocollo MQTT per la comunicazione machine-to-machine e come quest'ultimo sia una scelta perfetta per le applicazioni IoT basate su Node-RED e Raspberry Pi.

CONCLUSIONI

Siamo arrivati alla conclusione di questa puntata in cui abbiamo visto come gestire connettività via Websockets ed MQTT in Node-RED. Questi due protocolli costituiscono due potenti strumenti per la gestione della comunicazione, permettendoci di scambiare molto agevolmente informazioni tra più dispositivi o tra dispositivi e servers.

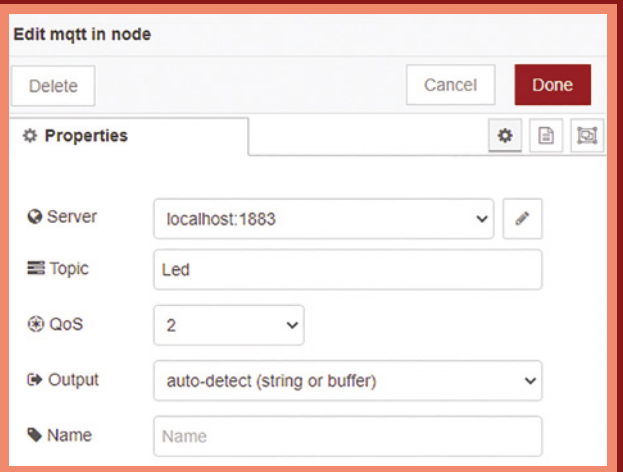


Fig. 17 - Finestra delle proprietà del nodo mqtt in.

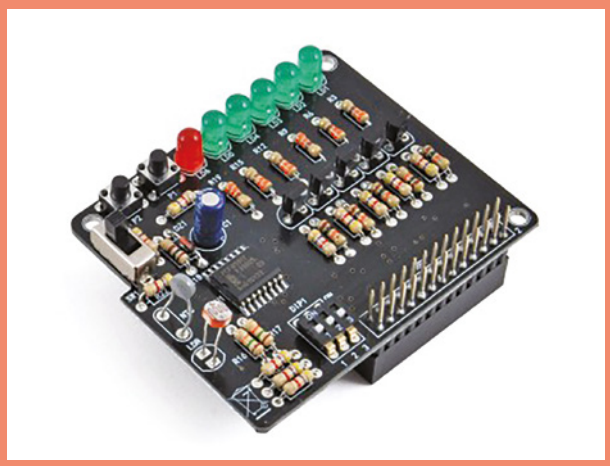


Fig. 18 - Shield Raspberry Pi I/O di Futura Elettronica.