CONOSCERE E USARE Node-RED 5

di FRANCESCO FICILI

Iniziamo a scoprire 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 ci occupiamo dell'implementazione delle funzionalità di connettività usando Node-RED, spaziando dal TCP all'UDP, fino ad arrivare all'HTTP. Quinta Puntata.

Come abbiamo visto sin dall'inizio del corso, Node-RED ha una spiccata propensione per le applicazioni in cui sia richiesto di implementare connettività di vario tipo.
Essendo nato come un tool per lo sviluppo di applicazioni IoT, integra di default una serie di features che permettono di implementare in maniera molto semplice vari tipi di connessioni, usando diversi protocolli e standard di comunicazione.
Questo aspetto è molto importante per le applicazioni IoT e, più in generale per le applicazioni connesse, in quanto è necessario che i nostri dispositivi interagiscano con l'hardware (ad esempio ricavino un dato da un sensore o comandino uno specifico attuatore), ma è altrettanto importante che tali dispositivi dialoghino tra di loro, con un server esterno, o con un dispositivo portatile come uno smartphone o un tablet.

CONNECTIVITY IN NODE-RED

Come anticipato in precedenza, Node-RED dispone di una quantità di nodi dedicati alla connettività. Molti di questi nodi sono pre-installati e raggruppati nella sezione "Network". Tra questi troviamo:

- UDP
- TCP
- HTTP
- Websocket
- MQTT

Inoltre, sempre nella sezione Network, troviamo i nodi Serial, che abbiamo già analizzato nella terza puntata del corso.
La Fig. 1 riporta i nodi della sezione Network.
In questa puntata ci concentreremo maggiormente su TCP, UDP e HTTP, lasciando la trattazione dell'MQTT e dei Websocket alla puntata successiva.

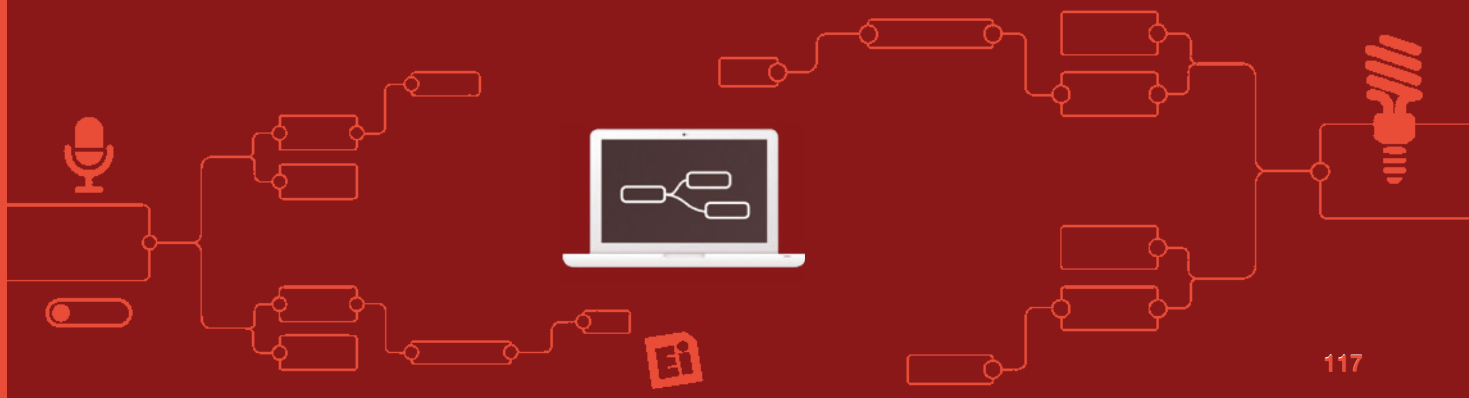

Fig. 1 - Nodi della sezione Network.

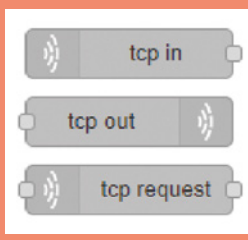

Fig. 2 - Nodi TCP

TCP

Il TCP, acronimo di Transmission Control Protocol, è un protocollo di trasmissione dati per reti informatiche ideato inizialmente nel 1973 da Robert E. Kahn e Vinton G. Cerf, durante un lavoro di ricerca. Questa idea iniziale di protocollo di trasmissione dati è stata migliorata ed espansa nel corso degli anni successivi, fino ad arrivare alla versione odierna che costituisce una delle colonne portanti della cosiddetta "famiglia di protocolli internet".
Il TCP è un protocollo di livello trasporto che permette a due nodi terminali di una rete di trasmissione dati di realizzare una connessione attraverso la quale può avvenire uno scambio bidirezionale di informazione. Il TCP è in grado di riconoscere, nell'ambito di questo interscambio di informazione, perdite di dati, ed è pertanto definito un protocollo affidabile. Essendo posizionato al livello 4 della pila OSI (trasporto) il TCP deve necessariamente appoggiarsi su un protocollo di rete, e nella stragrande maggioranza dei casi questo protocollo è il protocollo IP, tanto che spesso di parla della combinazione dei due come di standard TCP/IP.
A sua volta il protocollo TCP è spesso utilizzato dagli strati superiori della pila OSI come protocollo di trasporto per la realizzazione di protocolli di livello applicazione (ad esempio come nel caso dell'HTTP). Sebbene quest'ultimo sia il caso d'uso tipico, il protocollo TCP può essere anche utilizzato direttamente per realizzare trasmissione di dati affidabile tra due dispositivi, sebbene la grande proliferazione di protocolli di livello applicativo, estremamente standardizzati, specializzati e di semplice utilizzo e implementazione, renda questo caso sempre più raro.
Node-RED mette a disposizione una serie di nodi per la gestione del protocollo TCP, rappresentati in Fig. 2.
Sono presenti i seguenti nodi:
tcp in: questo nodo permette di ricevere dati tramite collegamento TCP. Il nodo consente sia di connettersi su una porta di un host remoto oppure di accettare connessioni in entrata su una specifica porta. I dati ricevuti sono inoltrati in output sul *msg.payload*.
tcp out: questo nodo permette di inviare dati tramite collegamento TCP. Il nodo consente sia di connettersi su una porta di un host remoto, di accettare connessioni in entrata su una specifica porta o di rispondere ad una richiesta TCP su una connessione esistente. Solo i dati presenti su *msg.payload* sono inviati in uscita.
tcp request: questo nodo implementa una semplice richiesta TCP, quindi esegue in sequenza le seguenti operazioni:

- Effettua la connessione
- Invia la richiesta
- Legge la risposta

Risulta molto utile per implementare sessioni di comunicazione, riducendo il numero di nodi da utilizzare. L'output può essere restituito in vari modi: aspettando la ricezione di un carattere speciale, dopo un timeout predefinito, dopo la ricezione di un determinato numero di caratteri, immediatamente, oppure si può scegliere di mantenere la connessione aperta non ritornando mai nulla in uscita.
Si noti che, di default, il nodo restituisce un array sul *msg.payload*, quindi per visualizzare l'output su una stringa è necessario una ulteriore elaborazione (ad esempio utilizzare il metodo *toString()* in un function node).
Vediamo un semplice esempio di utilizzo dei nodi TCP per la comunicazione tra dispositivi. Quello che ci proponiamo di realizzare è una semplice comunicazione bidirezionale su rete locale (ma sarebbe possibile eseguirla anche su rete internet) tra due nodi.

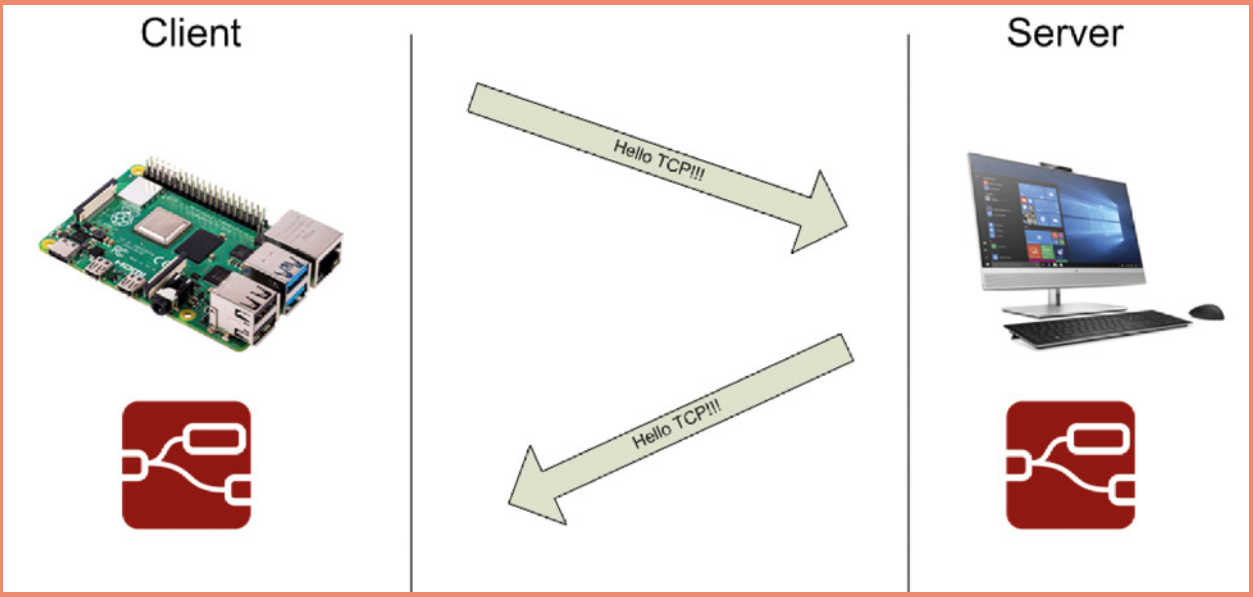

Fig. 3 - Setup utilizzato per l'esempio TCP.

Il nodo A effettuerà la connessione al nodo B (che funge quindi da server) su una specifica porta, ed invierà una stringa. Il nodo B, in ascolto sulla porta concordata, riceverà la stringa e risponderà con un eco (ossia ritrasmettendo la stessa stringa al nodo che ha effettuato la 'richiesta').
Si capisce subito che per implementare questo semplice esempio ci occorrono due dispositivi. Il primo dispositivo è la Raspberry Pi che abbiamo già utilizzato durante il resto del corso, mentre il secondo può essere un'altra Raspberry Pi (o dispositivo similare, in grado di far girare una istanza di Node-RED), oppure un PC. Come sappiamo infatti Node-RED può girare su vari tipi di macchine, inclusi PC windows. Questo è esattamente il setup che abbiamo utilizzato noi, usando una Raspberry Pi ed un PC windows, setup che risulta molto utile per effettuare test in cui è richiesta la comunicazione tra dispositivi, e che è anche rappresentato in Fig. 3. Per i dettagli su come installare e far girare Node-RED su windows potete fare riferimento alla guida ufficiale presente sul sito di Node-RED: *https://nodered.org/docs/getting-started/windows*.

Partiamo dall'implementazione del nodo client, che posizioniamo sulla Raspberry Pi e che ha il compito di inviare la richiesta (inviare la stringa "Hello TCP!!!") e restare in attesa della risposta (eco da parte del server).
Per effettuare la richiesta ci serviremo del nodo tcp request visto in precedenza.
Sia lato client che lato server useremo un carattere di terminazione (\n, newline) per indicare che la trasmissione dati è completa, quindi, dato che il nodo inject non permette nativamente di inserire caratteri speciali sulle stringhe, useremo un nodo function e il metodo *concat()* per aggiungere una nuova linea alla stringa.
Inoltre useremo anche un nodo function con il metodo *toString()* per convertire l'array ritornato in output dal nodo tcp request.
Per implementare questo flow ci occorrono quindi:

- Un nodo Inject
- Due nodi Function
- Un nodo Tcp Request
- Un nodo debug

Fig. 4 - Struttura del flow lato client.

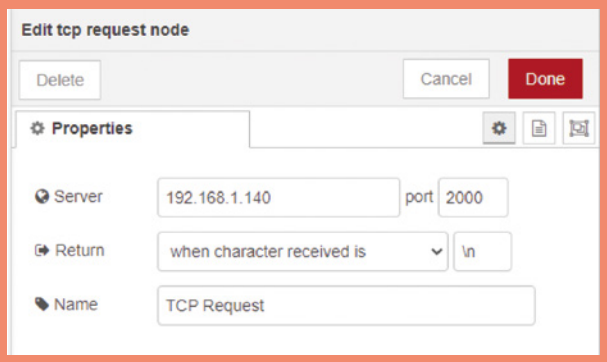

Fig. 5 - Configurazione del nodo TCP request.

La struttura del flow è rappresentata in Fig. 4.
Vediamo in dettaglio la configurazione dei nodi.
Il nodo Inject inietta nel flow semplicemente la stringa "Hello TCP!!!", mentre il nodo debug non ha configurazione.
I nodi concat e toString implementano sostanzialmente dei metodi javascript che operano su *msg.payload*, per eseguire delle manipolazioni sulla stringa contenuta nell'oggetto JS.
In particolare:
Nodo Function Concat: serve a concatenare il carattere speciale Newline (\n), tramite il seguente snippet di codice javascript:

```
msg.payload = msg.payload.concat('\n');
```

Nodo Function toString: serve a convertire l'output del nodo TCP requeest, che di default è un array, in una stringa, tramite il seguente snippet di codice javascript:

```
msg.payload = msg.payload.toString();
```

Infine vediamo come è configurato il nodo TCP request, avvalendoci della Fig. 5.
Il nodo TCP request richiede all'utente di fornire l'indirizzo dell'host (in questo caso l'indirizzo di rete locale 192.168.1.140, che rappresenta l'indirizzo IP del PC windows sul quale gira il nostro "echo server") e la porta (2000). Si noti che è stata usata la porta 2000 in quanto, su molti sistemi, l'accesso alle porte da 0 a 1024 è limitato.
Infine il nodo richiede di configurare come debba essere ritornata la risposta: come indicato in precedenza, scegliamo di aspettare il carattere speciale newline.
Passiamo ora al flow del server, implementato sul PC windows e rappresentato in Fig. 6.

Come si può vedere si tratta di un flow molto semplice, che fa uso di:

- Un nodo TCP in
- Un nodo TCO out
- Un nodo Function

Come è facile evincere, i nodi TCP in e TCP out servono, rispettivamente, per ricevere la richiesta ed inviare la risposta, mentre il nodo Function Concat ha la stessa funzione vista in precedenza, ossia concatenare il carattere newline.
Vediamo in dettaglio la configurazione dei noti TCP (in quanto il nodo function è identico all'implementazione vista in precedenza), partendo dal nodo TCP in, la cui finestra delle proprietà è rappresentata in Fig. 7.
Il nodo TCP sta sostanzialmente in ascolto sulla porta 2000 (listen on port 2000) per connessioni in ingresso e fornisce in output uno stream in formato stringa (Output: stream of String), delimitato dal carattere speciale \n (delimited by \n).
Ancora più semplice le configurazione del nodo TCP output, che deve sostanzialmente rispondere su una connessione TCP aperta, e quindi l'unica configiurazione da fare è settare la proprietà Type come "Reply to TCP", come illustrato in Fig. 8.
I nostri flow lato client e lato server sono ora pronti e possiamo semplicemente testare questa implementazione iniettando alcune stringe nel flow ed esaminando ciò che viene ricevuto sulla finestra di debug (Fig. 9), ossia l'eco della stringa inviata.

UDP

Dopo aver esaminato in dettaglio come implementare connessioni TCP in Node-RED passiamo ad esaminare l'UDP.

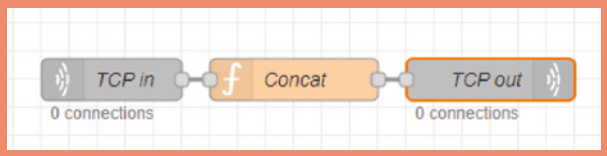

Fig. 6 - Flow del TCP server.

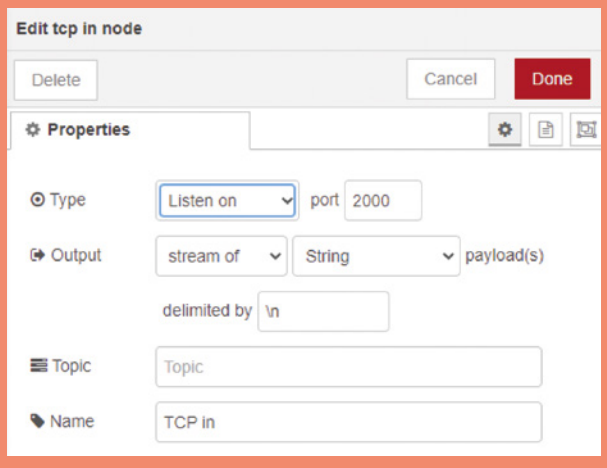

Fig. 7 - Configurazione del nodo TCP in.

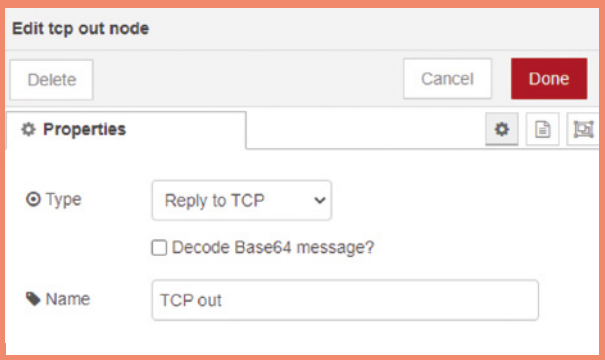

Fig. 8 - Configurazione del nodo TCP out.

L'UDP (User Datagram Protocol), è un altro protocollo di livello trasporto, a pacchetto, tipicamente usato in combinazione con un livello di rete IP. La caratteristica base dell'UDP, che lo differenzia dal suo parente prossimo, ossia il TCP, è quella di essere connectionless. In sostanza, mentre una trasmissione TCP ha luogo solo dopo una fase obbligatoria di autenticazione e la creazione di una connessione tra i due nodi che intendono scambiarsi dati, l'UDP rinuncia a questa caratteristica in funzione di una minore latenza nello scambio dati e quindi una maggiore velocità di trasmissione. Il fatto di essere connectionless semplifica notevolmente il protocollo, che di fatto risulta essere una alternativa snella e con latenze ridotte al minimo del TCP. Il risvolto della medaglia è che l'UDP è estremamente meno affidabile del TCP, infatti non c'è garanzia di consegna dei pacchetti, non viene gestita automaticamente la ritrasmissione dei pacchetti persi e non è garantita la sequenzializzazione. Tutte caratteristiche che invece sono presenti nel TCP, che è sicuramente più lento, ma anche estremamente più affidabile

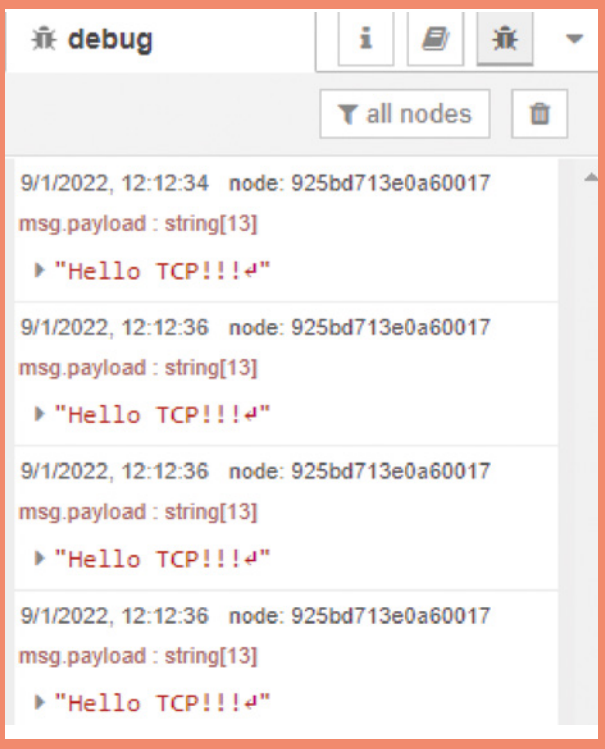

Fig. 9 - Finestra di debug (lato client) che mostra l'eco ricevuto dal server.

Di conseguenza applicazioni tipiche in cui si fa largo uso del protocollo UDP sono quelle in cui la perdita di dati è tollerabile in luogo di un maggior rate di trasmissione.
Esempi tipici sono lo streaming audio/video, le richieste DNS e le connessioni VPN.
La gestione dell'UDP in Node-RED è realizzata con due nodi, riportati in Fig. 10.
udp in: questo nodo permette di ricevere dati tramite collegamento UDP. Il nodo invia in output su *msg.payload* il buffer di trasmissione ricevuto sulla porta specificata. E' supportato sia IPv4 che IPv6, come anche le trasmissioni multicast. L'output può essere formattato sia come buffer di dati che come stringa.
udp out: questo nodo permette di inviare dati tramite collegamento UDP. Il nodo permette di indirizzare uno specifico host su una data porta. Anche in questo caso è supportato sia IPv4 che IPv6.
Anche per l'UDP possiamo realizzare un semplicissimo esempio, con un setup identico a quello utilizzato per il TCP. In questo caso è leggermente improprio parlare di client e server, dal momento che non viene instaurata una connessione, ma abbiamo, di fatto, due nodi della rete (diciamo A e B) che comunicano bidirezionalmente.
Diciamo anche che il nodo A è il nodo che inizia la comunicazione, mentre il nodo B è quello che implementa la funzionalità di eco dei dati.
L'implementazione in questo caso è molto semplificata. Partiamo dal nodo A, che nel nostro specifico caso è sulla macchina Windows (ma le parti potrebbero essere tranquillamente invertite). Per implementare la comunicazione UDP secondo le nostre specifiche iniziali, dobbiamo realizzare un flow che includa:

- Un nodo inject
- Un nodo debug
- Un nodo UDP out
- Un nodo UDP in

Il flow che implementa il nodo A è riportato in Fig. 11.

Fig. 10 - Nodi UDP di Node-RED.

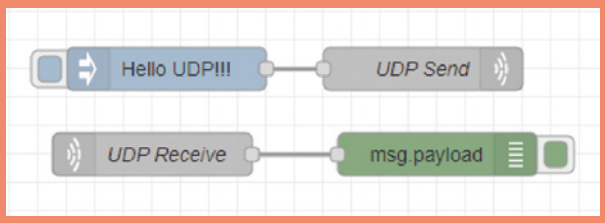

Fig. 11 - Flow che implementa il nodo A.

Vediamo il dettaglio della configurazione dei due nodi UDP, partendo dal nodo UDP send, la cui finestra delle proprietà è riportata in Fig. 12.
Come si può facilmente notare inseriamo la porta sulla quale vogliamo inviare i dati (3000 in questo caso) e l'indirizzo IP del nodo destinazione (ricordatevi sempre di adattarlo a quello del dispositivo destinazione nella vostra rete locale). Il resto delle configurazioni possono essere mantenute ai valori di default.
Passiamo adesso alla configurazione del nodo UDP Receive, illustrata in Fig. 13.
In questo caso ci basta specificare la porta sulla quale ricevere i dati e il tipo di output, che vogliamo sia una stringa.
A questo punto ci possiamo spostare sul nodo B, il cui flow, semplicissimo, è rappresentato in Fig. 14.
Le configurazioni dei due nodi UDP Receive e UDP Send, sono sostanzialmente identiche a quelle viste per il nodo A. L'unica differenza è il campo Address del nodo Send, che, ovviamente, questa volta deve contenere l'indirizzo IP del nodo A.
L'implementazione dei nostri flow è ora completa e possiamo passare al test. Se tutto è stato fatto correttamente, ogni volta che inviamo una stringa dal nodo A ne riceveremo l'eco da parte del nodo B, su collegamento UDP.
La Fig. 15 mostra l'output ottenuto nella finestra di debug di Node-RED.

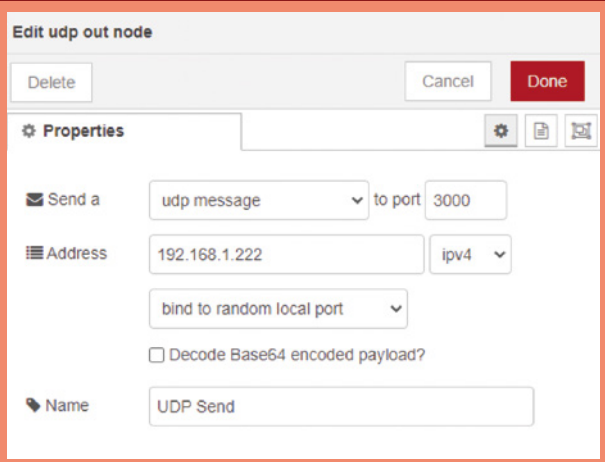

Fig. 12 - Finestra delle proprietà del nodo UDP send.

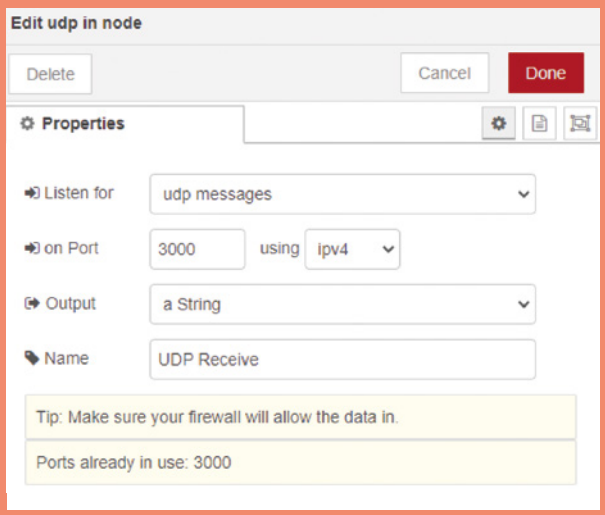

Fig. 13 - Finestra delle proprietà del nodo UDP receive.

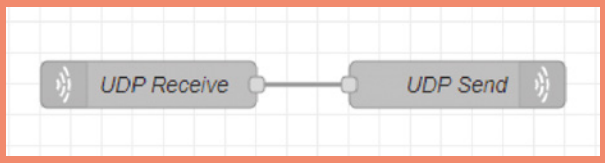

Fig. 14 - Flow del nodo B.

HTTP

L'acronimo HTTP sta per "HyperText Transfer Protocol", e sta ad indicare un protocollo sviluppato negli anni '80 da Tim Berners-Lee come una delle tre componenti costituenti il nucleo base del World Wide Web, che rappresenta uno dei principali servizi offerti da internet, e di cui tutti noi facciamo ampio uso ormai da più di due decenni.
Gli altri due componenti del nucleo base sono il linguaggio HTML e gli URL. Gli URL sono le stringhe che rappresentano i domini dei siti web, e l'HTML è il linguaggio che viene utilizzato per creare i contenuti delle pagine dei siti.
L'HTTP invece, è il protocollo che governa il trasferimento dei dati tra un client (ossia un web browser) e il server che ospita il sito che vogliamo visualizzare. Il protocollo HTTP è quindi un protocollo client-server e si colloca al livello applicazione della pila OSI.
L'HTTP è quindi generalmente utilizzato per permettere la comunicazione client-server nel World Wide Web, ossia per far comunicare i browser con i server.
Il suo funzionamento è molto semplice:

- il client esegue una richiesta di una certa risorsa (ad esempio una pagina di un sito web).
- Una volta ricevuta la richiesta il server cerca il contenuto desiderato (ad esempio la homepage del sito, index.html) ed invia prima un header con un codice di stato che serve ad informare il client dell'esito della richiesta.

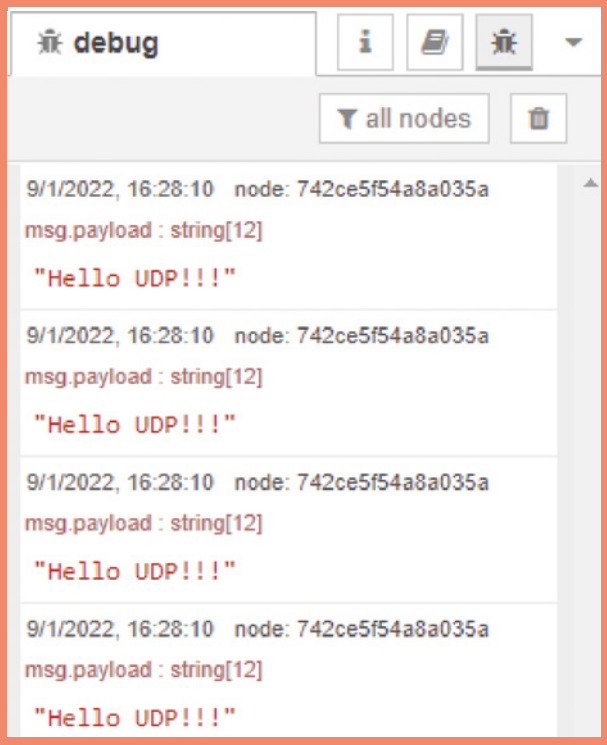

Fig. 15 - Output della finestra di debug del nodo A.

- Se il contenuto viene trovato, il server lo invia al client nel body del messaggio
- Il client riceve il dato e, se si tratta di un browser, lo visualizza come pagina web

C'è da notare che non sempre l'HTTP è utilizzato per la comunicazione tra un browser e un server che ospita un sito web; infatti si può utilizzare l'HTTP per realizzare comunicazione machine-to-machine, o per l'implementazione di servizi web come ad esempio API REST, o anche per accedere ad un database web. Node-RED fornisce 3 nodi per la gestione del protocollo http (Fig. 16):
http in: permette di creare un http endpoint per la gestione delle richieste standard previste dal protocollo http (GET, POST, PUT, DELETE, PATCH), quindi in sostanza permette di implementare un webservice. Il nodo richiede anche il path locale della richiesta (campo URL).
http response: permette di inviare risposte alle richieste ricevute da un nodo http in.
http request: questo nodo implementa il lato client del protocollo http, quindi permette di inviare richieste ad una specifica URL e ritorna indietro le risposte ricevute dal server.

ESEMPIO PRATICO: CONTROLLO DI UN RELÈ TRAMITE HTTP

Concludiamo questa puntata dedicata alla connettività con il classico esempio pratico. Ciò che ci proponiamo stavolta è di implementare un semplice webserver che ci permetta di azionare un relè da remoto, usando una richiesta standard http. Per implementare questo webserver ci occorrono i seguenti nodi:

- 2 Nodi http in
- 1 Nodo http response
- 2 Nodi change
- 1 Nodo rpi-gpio out

Ciò che vogliamo fare è intercettare due differenti richieste http GET, riferite ai path interni fasulli /relay_on e /relay_off del nostro server.
A queste due richieste forniremo comunque risposta positiva e sfrutteremo i due flussi per triggerare 2 change node che andranno a comandare il nodo rpi-gpio out con i valori corretti per attivare e disattivare il relè. Il flow risultante è visibile in Fig. 17:
Analizziamo la configurazione dei Nodi http, aiutandoci con la finestra delle proprietà del nodo Relay ON (Fig. 18).
Come si può facilmente notare, abbiamo utilizzato un metodo http GET e la risorsa che vogliamo indirizzare è il path interno /relay_on (l'URL del server sarà l'indirizzo IP della raspberry).
Questo nodo verrà triggerato quando verrà effettuata una richiesta http su quel path specifico. Il nodo Relay OFF è del tutto simile, cambierà solamente il path, che sarà /relay_off.

Fig. 16 - Nodi Node-RED per la gestione del protocollo HTTP.

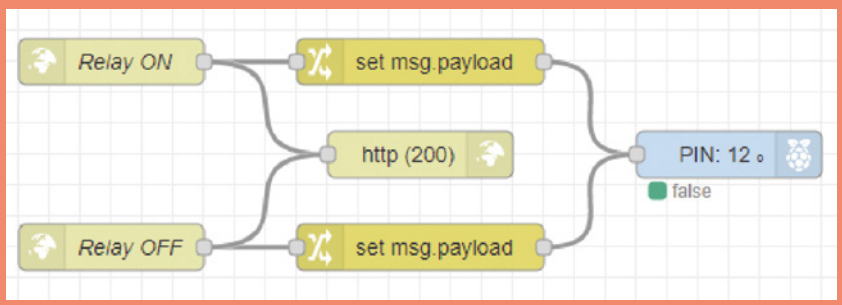

Fig. 17 - Flow che implementa il webserver per il controllo di un relè tramite http.

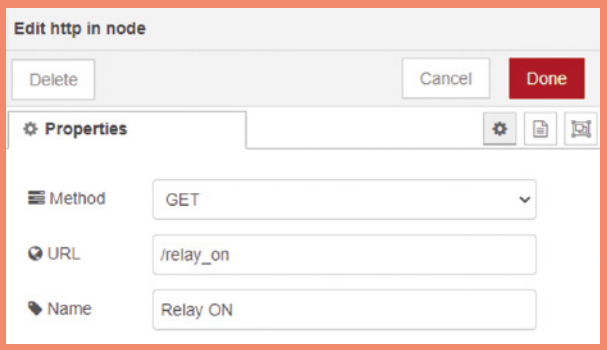

Fig. 18 - Proprietà del nodo Relay ON (http in).

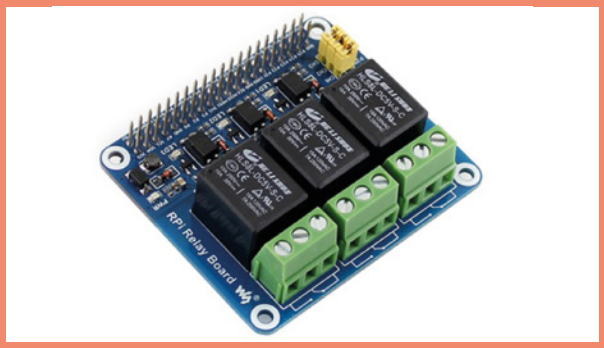

Fig. 20 - HAT RPIRELEBOARD.

Alla richiesta è opportuno fornire una risposta, che nel nostro caso sarà sempre positiva, in quanto non abbiamo la reale possibilità che si verifichi un errore, dovendo controllare semplicemente un pin (controlli più sofisticati, in applicazioni reali sono comunque possibili).
Usiamo quindi un nodo http response in cui configuriamo lo status code come 200 (risposta positiva del server standardizzata in http), come si può vedere dalla Fig. 19, che rappresenta la finestra delle proprietà del nodo.
L'output dei due nodi http in, che viene triggerato ogni volta che viene fatta una richiesta GET dei path /relay_on e /relay_off, non è adatto a pilotare direttamente il nodo rpi-gpio out, quindi utilizziamo due nodi change per modificare opportunamente il contenuto del msg.payload.
Il nodo change che manipola il flusso del ramo relay_on setta a true il valore di msg.payload, mentre l'altro lo setta a false. Entrambi gli output dei due nodi vanno in input al nodo rpi-gpio out, che comanda il canale corrispondente al relè che vogliamo attivare. Per testare il nostro flow possiamo usare l'HAT *RPIRELEBOARD*, un HAT che monta 3 relè a 250 VAC/5 A e 30 VDC / 5 A, commercializzato da Futura Elettronica, una cui immagine è visibile in Fig. 20.

Per effettuare le richieste possiamo usare un qualsiasi browser, collegandoci ai seguenti indirizzi:
<IP Raspberry>:1880\relay_on ➜ per attivare il relè
<IP Raspberry>:1880\relay_off ➜ per disattivare il relè
In alternativa è possibile utilizzare delle app android che permettono di effettuare richieste http tramite shortcuts su una interfaccia grafica.
Una app di questo tipo molto funzionale e completamente gratuita è http shortcuts, che permette di creare shortcuts con nome, descrizione ed icone associate.
La creazione delle shortcuts è molto intuitiva ed un esempio di pannello che si riesce a realizzare è riportato in Fig. 21.

CONCLUSIONI

Siamo arrivati alla conclusione di questa puntata in cui abbiamo visto come gestire la connettività in Node-RED usando i protocolli TCP, UDP e HTTP. Queste caratteristiche ci consentono diverse possibilità implementative, garantendoci di poter gestire la comunicazione tra i dispositivi con diverse modalità. Nella prossima puntata introdurremo altri due standard molto utilizzati in ambito IoT: l'MQTT e i Websocket. □

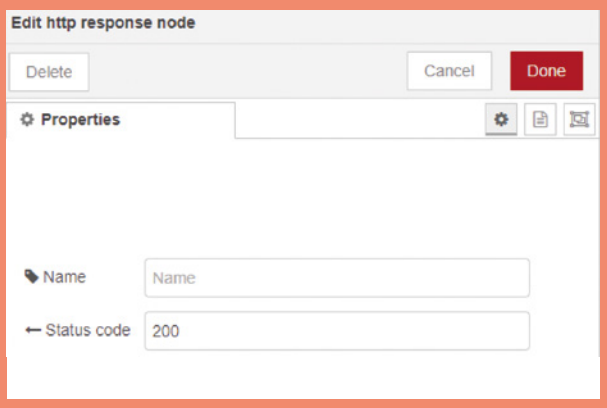

Fig. 19 - Finestra delle proprietà del nodo http response.

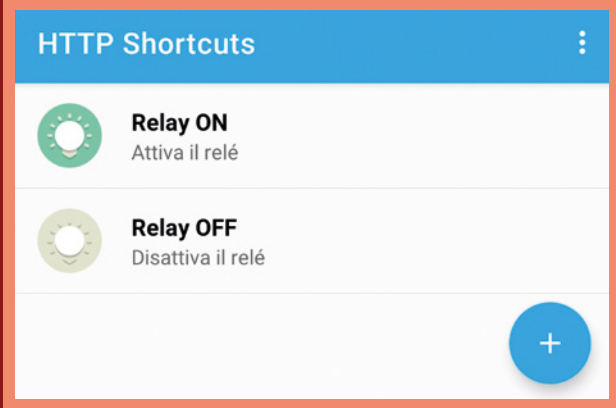

Fig. 21 - Pannello per la gestione del relè realizzato tramite HTTP shortcuts.