

Corso Node-RED

Corso di <https://futuranet.it/download/corso-node-red/>
riformattato per la stampa e la versione epub.

Node-RED è un tool di programmazione grafica sviluppato dall'*Emerging Technology Services* team di IBM. Il progetto nasce all'inizio del 2013, da Nick O'Leary e Dave Conway-Jones, come un proof-of-concepts per la visualizzazione ed il mapping tra topics MQTT, per diventare velocemente un tool più generale che consente lo sviluppo di applicazioni di vario tipo. Il progetto è open-source praticamente dalla sua concezione, e quest'orientamento open è culminato nel divenire uno dei 32 progetti della JS fundation, nell'ottobre 2016. Node-RED consiste sostanzialmente in una runtime di Node.js, che implementa un server al quale si può far puntare un qualsiasi browser per accedere all'editor grafico. All'interno dell'editor visualizzato dal browser è possibile creare i programmi sotto forma di insieme di nodi (flussi o flows) interconnessi tra di loro. I nodi vengono inseriti nei flussi prelevandoli da un'apposita palette presente nell'editor. La palette dei nodi può facilmente essere espansa tramite altri nodi creati dall'utente stesso o dalla community. Una volta che il flusso è completo è possibile eseguirne il deploy e quindi mandarlo in esecuzione, tramite un apposito pulsante.

- Essendo basato su Node.js, Node-RED può virtualmente essere eseguito su qualsiasi sistema in grado di supportare Node.js, compresi:
 - PC Linux o Windows;
 - sistemi embedded come Raspberry Pi o BeagleBoard;
 - sistemi Android;
 - server.

Nel resto di questo corso utilizzeremo Node-RED su Raspberry PI, ma i concetti esposti restano validi per l'utilizzo di Node-RED anche su altre piattaforme; ciò che può cambiare sono principalmente alcune caratteristiche relative all'installazione, per le quali rimandiamo i lettori all'apposita sezione sul sito web ufficiale: <https://nodered.org/docs/getting-started/>.

Indice

Corso Node-RED	1
1. Prima Puntata	3
COS'È NODE-RED	3
FLOW-BASED PROGRAMMING	4
INSTALLARE ED AVVIARE NODE-RED	4
NODE-RED EDITOR	6
CONCLUSIONI	8
2. Seconda puntata	18
FLUSSI, NODI E MESSAGGI.....	18
CORE NODES	20
CONCLUSIONI	24
3. Terza puntata	32
PERIFERICHE RASPBERRY PI	32
GESTIONE GPIO RASPBERRY PI	33
DIGITAL I/O	33
GPIO out	34
GPIO in	34
ESEMPIO PRATICO: LETTURA DI UN SENSORE DS18B20 TRAMITE INTERFACCIA SERIALE	37
CONCLUSIONI	39
4. Quarta Puntata.....	45
CONFIGURARE NODE-RED	45
OPZIONI DI SECURITY.....	46
NODE-RED DASHBOARD	47
DASHBOARD NODES.....	48
ESEMPIO	49
SECURITY DELLA UI	50
ESEMPIO PRATICO: STAZIONE METEO CON BME280	50
CONCLUSIONI	52
5. Quinta Puntata.....	59
CONNECTIVITY IN NODE-RED.....	59
TCP	59
UDP	62
HTTP	63
ESEMPIO PRATICO: CONTROLLO DI UN RELÈ TRAMITE HTTP	64
CONCLUSIONI	65
6. Sesta Puntata.....	73
WEBSOCKET	73
WEBSOCKET IN NODE-RED	73
UN ESEMPIO DI UTILIZZO DEI WEBSOCKET	74
MQTT	74
IL PATTERN PUBLISH-SUBSCRIBE	75
PROTOCOLLO MQTT	75
MQTT IN NODE-RED	76
MQTT BROKER IN NODE-RED	76
UN ESEMPIO DI UTILIZZO DI MQTT	77
CONCLUSIONI	78
7. Settima Puntata.....	84
INVIARE EMAIL TRAMITE NODE-RED	84
INTEGRAZIONE CON WHATSAPP	86
INTEGRAZIONE CON TELEGRAM	86
ESEMPIO PRATICO: CONTROLLARE UN LED TRAMITE TELEGRAM	88
I bot di Telegram	88
CONCLUSIONI	89
8. Ultima Puntata.....	97
INTEGRAZIONE DI ALEXA IN NODE-RED	97
NODE-RED COME ALEXA DEVICE	97
INTERFAZIAMENTO CON ALEXA API	99
CONCLUSIONI	101

1. Prima Puntata.

Iniziamo a scoprire 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. Spiegheremo cos'è Node-RED ed inizieremo a prendere confidenza con l'ambiente e con l'editor grafico, mostrando poi alcuni esempi di utilizzo su Raspberry Pi.

Negli ultimi anni si è assistito ad una sempre maggiore diffusione di applicazioni IoT, tanto che ormai la connettività è una caratteristica di base della maggior parte degli oggetti che utilizziamo ogni giorno. Basti pensare ad un comune elettrodomestico moderno, come una lavatrice o un frigorifero, per renderci conto che ormai il fatto che l'oggetto sia connesso è una caratteristica quasi imprescindibile in ogni genere di applicazioni, per non parlare di oggetti più specializzati, come ad esempio gli assistenti vocali di Amazon e Google, che fanno della connettività e della loro vocazione IoT la loro caratteristica principale. Anche nel mondo dei maker e degli hobbisti si percepisce sempre più la necessità di poter connettere i propri progetti con la Rete, per eseguire le più svariate funzionalità; pensiamo ad esempio ad un impianto di irrigazione che desideriamo controllare dal nostro smartphone o ad una stazione meteo che ci fornisca i dati acquistati dai suoi sensori direttamente su un PC o via mail, o, ancora meglio, tramite Telegram.

Utilizzando piattaforme ed ambienti di sviluppo tradizionali è sicuramente possibile ottenere risultati di questo tipo, ma è necessario scrivere parecchie righe di codice, utilizzando librerie di una certa complessità che spesso pregiudicano la manutenibilità e la scalabilità delle nostre applicazioni.

Una soluzione alternativa al problema, che semplifica moltissimo l'approccio e consente di avere delle applicazioni sempre facilmente leggibili e scalabili, è l'utilizzo di un ambiente grafico specializzato nella realizzazione di applicazioni IoT ed orientate alla connettività. Dato che questa necessità è fortemente sentita, un gruppo di programmati della IBM ha sviluppato un ambiente di programmazione che segue questo approccio, dando vita a quello che oggi è conosciuto come Node-RED.

COS'È NODE-RED

Node-RED è un tool di programmazione grafica sviluppato dall'Emerging Technology Services team di IBM.

Il progetto nasce all'inizio del 2013, da Nick O'Leary e Dave Conway-Jones, come un proof-of-concepts per la visualizzazione ed il mapping tra *topic* MQTT, per diventare velocemente un tool più generale che consente lo sviluppo di applicazioni di vario tipo. Il progetto è open-source praticamente dalla sua concezione, e quest'orientamento open è culminato nel divenire uno dei 32 progetti della JS foundation, nell'ottobre 2016.

Node-RED consiste sostanzialmente in una runtime di *Node.js*, che implementa un server al quale si può far puntare un qualsiasi browser per accedere all'editor grafico. All'interno dell'editor visualizzato dal browser è possibile creare i programmi sotto forma di insieme di nodi (flussi o flows) interconnessi tra di loro.

I nodi vengono inseriti nei flussi prelevandoli da un'apposita palette presente nell'editor. La palette dei nodi può facilmente essere espansa tramite altri nodi creati dall'utente stesso o dalla community.

Una volta che il flusso è completo è possibile eseguirne il deploy e quindi mandarlo in esecuzione, tramite un apposito pulsante.

Essendo basato su *Node.js*, Node-RED può virtualmente essere eseguito su qualsiasi sistema in grado di supportare *Node.js*, compresi:

- PC Linux o Windows;
- sistemi embedded come Raspberry Pi o BeagleBoard;
- sistemi Android;

- server.

Nel resto di questo corso utilizzeremo Node-RED su Raspberry Pi, ma i concetti esposti restano validi per l'utilizzo di Node-RED anche su altre piattaforme (sono riassunte nella **Figura 1-2**); ciò che può cambiare sono principalmente alcune caratteristiche relative all'installazione, per le quali rimandiamo i lettori all'apposita sezione sul sito web ufficiale: <https://nodered.org/docs/getting-started/>.

FLOW-BASED PROGRAMMING

Il concetto principale su cui si basa Node-RED è quello della cosiddetta “Flow-Based Programming” o FBP. Tale paradigma di programmazione, enunciato per la prima volta da J. Paul Morrison nel 1970, descrive le applicazioni come una network di processi “black box”, ossia di cui non si conoscono i dettagli implementativi, ma di cui sono noti i dati in ingresso, il tipo di elaborazione eseguita su di essi e i dati in uscita. I processi black box sono anche noti come nodi (questo è proprio il termine utilizzato da Node-RED) e tali nodi si scambiano dati attraverso messaggi instradati tramite connessioni predefinite (network). Ne consegue che la FBP è naturalmente “component oriented”, ossia è un paradigma che enfatizza la modularità nello sviluppo software. La FBP vede un'applicazione non come un processo sequenziale, che inizia ad un certo istante di tempo e fa una cosa alla volta finché non completa la sua esecuzione, ma piuttosto come un insieme di processi asincroni che comunicano tra di loro per mezzo di uno stream di dati strutturati, chiamati “*information packet*” (IP). Il focus è quindi sui dati applicativi e sulle trasformazioni applicate su di essi al fine di produrre l'output desiderato. La network è definita esternamente ai processi e viene interpretata a sua volta come una parte di software, normalmente chiamata “scheduler”.

In FBP i processi comunicano per mezzo di connessioni e tali connessioni sono collegate ai processi per mezzo di porte (in ingresso o in uscita); in un certo lasso temporale, un IP può essere manipolato solo da un singolo processo, oppure può essere in transito tra due processi.

Data la sua natura, la definizione di una network in FBP è normalmente un diagramma, ma può anche essere convertita in una lista di connessioni e nodi, se viene descritta con un linguaggio di basso livello o una particolare notazione (vedremo che questo avviene anche in Node-RED).

In **Figura 1-3** è riportato un semplice diagramma FBP.

A, B e C sono processi, mentre O1, O2 e IN sono porte. M ed N invece sono connessioni che connettono le porte dei processi tra di loro. Le connessioni hanno una capacità fissa in termini di numero di IPs che possono essere immagazzinati in un certo istante di tempo.

Node-RED implementa tutti i concetti base della FBP, quindi un programma o, per usare la terminologia Node-RED, un flow, è descritto come una serie di nodi interconnessi tra di loro. Ogni nodo implementa una specifica funzione, riceve un set di dati da una porta in ingresso, li manipola e li passa alla porta/e in uscita. La network è responsabile del flusso di dati attraverso i vari nodi. In **Figura 1-4** è riportato un esempio di flow in Node-RED.

Prima di addentrarci sui dettagli della programmazione di un flow vediamo però come installare ed avviare l'ambiente.

INSTALLARE ED AVVIARE NODE-RED

Come abbiamo già accennato, Node-RED può girare su una moltitudine di piattaforme: di fatto, su ogni piattaforma capace di far girare Node.js. Dal punto di vista dell'utilizzo non c'è una grossa differenza, ma ovviamente la procedura di installazione può differire da caso a caso.

Oltre a questo, in alcuni casi occorre preparare opportunamente la piattaforma. In questa sezione forniremo una serie di istruzioni complete per l'installazione su Raspberry Pi, ma si tenga conto che non si tratta di istruzioni valide in generale quindi consultate la relativa sezione del sito web di Node-RED per maggiori informazioni.

Per installare ed eseguire Node-RED su Raspberry PI occorre innanzitutto munirsi di una Raspberry. Non ci sono particolari limitazioni rispetto al modello da utilizzare, ma si consiglia di non usare un modello troppo antiquato.

Per le nostre prove abbiamo utilizzato una Raspberry Pi 4 Model B con 4 GB di RAM.

Per prima cosa occorre preparare una scheda SD con un'immagine di Raspberry OS al suo interno, usando *Raspberry Pi Imager*. Se si utilizza un'immagine “*full with recommended SW*”, Node-RED risulterà già preinstallato, ma sconsigliamo di utilizzare questa opzione sia per i tempi di creazione dell'immagine che per il fatto che la versione preinstallata di Node-RED potrebbe essere non aggiornata all'ultima versione.

In alternativa è possibile creare un'immagine di tipo Lite ed installare Node-RED successivamente.

Creiamo quindi un'immagine di tipo Lite e predisponiamo la Raspberry al collegamento con la nostra rete domestica in modalità headless, abilitando al contempo la connessione SSH. Per l'accesso headless rimandiamo alla guida reperibile su <https://www.raspberrypi.org/documentation/configuration/wireless/headless.md> presente sul sito web della Raspberry PI foundation. Per abilitare la connessione SSH inserire un file privo di estensione e con nome “SSH” all'interno della partizione di boot della SDcard.

Una volta eseguite queste operazioni sarà possibile collegarsi alla Raspberry Pi tramite un client SSH, come ad esempio PuTTY (Figura 1-5).

Per installare Node-RED e le sue dipendenze principali (ossia Node.js e npm) è possibile utilizzare il seguente script, che installa i vari componenti ed i nodi specifici per Raspberry Pi. Lo script può anche essere utilizzato per aggiornare un'installazione di Node-RED esistente.

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

Più in dettaglio lo script esegue le seguenti operazioni:

- rimuove le versioni di Node-RED e Node.js preesistenti, se presenti;
- installa la versione di Node.js LTS corrente usando NodeSource; se Node.js è già presente viene eseguito un controllo di compatibilità per assicurarsi che Node sia almeno alla versione 12;
- installa l'ultima versione di Node-RED usando npm;
- se desiderato dall'utente installa una serie di nodi specifici per Raspberry Pi;
- setta Node-RED per funzionare come service e fornisce una serie di comandi specifici per utilizzare i servizi.

Il risultato dell'esecuzione è quello proposto nella schermata di Figura 1-6. Si noti inoltre che Node-RED è stato anche inserito nei repository di Raspberry Pi Os, e compare nella lista dei software raccomandati dalla foundation, e può quindi essere installato anche tramite

```
apt-get (apt-get install nodered).
```

Tuttavia, con questa procedura non verrà installato npm (mentre è comunque incluso Node.js).

La documentazione di Node-RED comunque suggerisce l'utilizzo dello script per installare ed aggiornare Node-RED su Raspberry PI. Come spiegato, lo script esegue anche i settaggi necessari per permettere la gestione di Node-RED come servizio; ciò significa che è possibile far girare Node-RED in background ed avviarlo automaticamente al boot.

Sono disponibili i seguenti comandi per la gestione del servizio:

- *node-red-start*, che avvia Node-RED e visualizza i log sulla console;
- *node-red-stop*, che arresta Node-RED;
- *node-red-restart*, che arresta e riavvia Node-RED;
- *node-red-log*, che visualizza i log del servizio.

È anche possibile fare in modo che il servizio venga avviato automaticamente al boot.

Per farlo è sufficiente eseguire il seguente comando sulla console:

```
sudo systemctl enable nodered.service
```

Per disabilitare l'autostart al boot basta invece eseguire questo comando:

```
sudo systemctl disable nodered.service
```

NODE-RED EDITOR

Ora che Node-RED è correttamente installato sulla nostra Raspberry Pi, possiamo aprire l'editor; questo può essere aperto con un comune browser, puntando all'indirizzo nel quale il server sta girando. Quindi se vogliamo accedere "localmente", ossia dalla stessa Raspberry nella quale abbiamo installato Node-RED è sufficiente puntare alla porta 1880 del localhost: <http://localhost:1880>. Tuttavia nella maggior parte dei casi risulta più conveniente accedere a Node-RED da un altro PC, puntando all'indirizzo IP della Raspberry nella nostra network locale sulla porta 1880, ossia

`http://<hostname>:1880`, dove `<hostname>` rappresenta appunto l'indirizzo IP della Raspberry, che può essere recuperato dal nostro access point casalingo. Una volta aperto il browser all'indirizzo corretto e sulla porta 1880, dovremmo trovare una schermata simile a quella in Figura 1-7. L'editor è composto essenzialmente dai seguenti quattro componenti:

- **Header:** contiene il pulsante di deploy, il menu principale, e, se la user authentication è abilitata, lo user menu.
- **Workspace:** è l'area di lavoro sulla quale i flows possono essere creati, inserendo nodi e connessioni tra di essi.
- **Palette:** contiene i nodi che possono essere utilizzati. Un nodo viene selezionato nella Palette e trascinato sul workspace per essere posizionato.
- **Sidebar:** contiene una serie di tools aggiuntivi che possono essere utilizzati dall'utente.

Analizziamo questi elementi più nel dettaglio per comprenderne meglio alcune caratteristiche:

Header

Come abbiamo visto, l'header contiene essenzialmente il menu principale, accessibile cliccando sull'icona con tre barre orizzontali ed il pulsante di deploy. Torneremo sul menu principale in seguito nelle puntate successive per illustrarne l'utilizzo. Per quanto riguarda invece il pulsante di deploy è importante sottolineare che la funzione di questo pulsante serve per rendere effettive le modifiche che abbiamo eseguito sul flow; sostanzialmente tutte le volte che modifichiamo un flow le informazioni vengono accumulate sul client ma non trasferite sul server che gira sulla Raspberry. Per far sì che il flow lato server venga aggiornato deve appunto essere premuto il pulsante di deploy, che si colora di rosso se il flow ha delle modifiche di cui non è ancora stato fatto il deploy. Questo pulsante permette 3 differenti modalità di deploy (come evidenziato in Figura 1-8):

- **Full:** viene eseguito il deploy di tutti flows sul workspace
- **Modified Flows:** viene eseguito il deploy solo dei flow modificati
- **Modified Nodes:** viene eseguito il deploy solo dei nodi modificati

Workspace

Il workspace è la principale area di lavoro di Node-RED, dove i flows sono sviluppati prendendo nodi dalla Palette e collegandoli insieme. Il workspace ha una serie di tab in alto, corrispondenti ai vari flows che sono stati creati ed aperti. Il *footer* del workspace ha alcuni tool di ausilio alla navigazione dei flows (Figura 1-9). In particolare, sono presenti pulsanti per eseguire zoom e de-zoom nel flow e per ripristinare il valore di zoom di default. Inoltre, è disponibile un pulsante per attivare la navigazione del flow, se questo dovesse essere talmente grande da non poter essere visualizzato per intero nella finestra.

È inoltre possibile seguire alcune customizzazioni del workspace tramite l'opzione "settings" del menu principale, come visibile in Figura 1-10.

Palette

La Palette contiene la lista di tutti i nodi installati e disponibili all'uso. I nodi sono organizzati in categorie (ad esempio alcune delle categorie di base sono *common*, *output* e *functions*). Cliccando l'header di una categoria è possibile comprimere o espanderne la visualizzazione (inoltre due pulsanti presenti al fondo della palette, con il simbolo di una doppia freccia verso l'alto o verso il basso possono essere utilizzati per comprimere ed espandere tutte le categorie). Nella parte alta della palette è presente anche un input testuale che permette di filtrare la lista dei nodi per nome.

Sidebar

Infine, la sidebar è una barra laterale che contiene una serie di tools aggiuntivi a disposizione dell'utente. I tool di base sono i seguenti:

- **Information**: permette di visualizzare informazioni sui nodi.
- **Help**: permette di visualizzare l'Help dei nodi.
- **Debug**: permette di visualizzare i messaggi passati al nodo Debug.
- **Configuration Nodes**: permette di gestire i *Configuration Nodes* (un particolare tipo di nodo non visualizzato sul workspace).
- **Context Data**: permette di analizzare il contenuto dei context, un elemento che non è stato ancora presentato, e che verrà analizzato nella prossima puntata.

Inoltre, alcuni nodi contribuiscono alla sidebar aggiungendo pannelli specifici (è il caso della Node-RED Dashboard, che vedremo in una delle puntate successive).

USARE NODE-RED SU RASPBERRY PI

Ora che abbiamo introdotto i concetti base di Node-RED e abbiamo presentato l'editor, passiamo al primo semplice esempio pratico, in modo da evidenziare la semplicità che Node-RED consente di raggiungere nello sviluppo delle applicazioni. Ciò che vogliamo realizzare è il classico lampeggio di un LED, pilotandolo da un GPIO della Raspberry alla frequenza di 1Hz. Per realizzare questo semplice esempio ci bastano 3 nodi:

- un nodo **Inject**;
- un nodo **Trigger**;
- un nodo **Rpi – Gpio Out**.

Iniziamo posizionando il nodo Inject, prelevandolo dalla Palette all'interno della categoria common. Il nodo *Inject* (Figura 1-11) è utilizzato per iniettare dati sul flow, di diverso tipo e con varie modalità. Noi lo utilizzeremo per generare un *inject* periodico che fornisca la periodicità di esecuzione al nostro flow.

Una volta posizionato il nodo va configurato, per dargli il comportamento desiderato. In questo caso noi vogliamo che il nodo Inject fornisca la periodicità. Per configurarlo dobbiamo cliccare due volte sul nodo con il tasto sinistro del mouse. Una volta eseguita questa operazione si aprirà la finestra di Figura 1-12, dalla quale è possibile configurare il nodo Inject. In questo particolare caso a noi basta configurare l'opzione Repeat, presente in basso, impostandola su *Interval* e indicando 2 secondi come intervallo.

Non ci interessa modificare il Payload del nodo, che in questo caso è un timestamp, perché non lo utilizzeremo per pilotare direttamente il LED, ma solo per fornire la periodicità di attivazione.

Una volta fatto clicchiamo su *Done* per chiudere. Ora possiamo inserire il nodo trigger, prelevandolo dalla categoria *functions* della Palette (Figura 1-13).

La funzione del nodo trigger è quella di inviare un messaggio con un determinato payload quando attivato e, dopo un lasso di tempo configurato dall'utente, un secondo messaggio, sempre con un determinato payload. Questo è il nodo che noi utilizzeremo per generare l'informazione di controllo per il LED (prima on e dopo un secondo off). Apriamo

la finestra di configurazione del nodo e modifichiamo il delay tra il primo e il secondo messaggio impostandolo ad 1 secondo (Figura 1-14). In questo modo il nodo invierà, quando attivato, ossia alla ricezione di un messaggio in ingresso (il messaggio inviato dal nodo *inject* ogni 2 secondi), prima un messaggio contenente “1” e poi un secondo messaggio contenente “0”. In questo modo abbiamo generato l’invio di un messaggio contenente “1” e dopo un secondo di un messaggio contenente “0” ogni 2 secondi, ossia abbiamo una serie di messaggi che rappresentano il “lampeggio” del nostro LED. Non ci rimane altro da fare che inserire un nodo che piloti direttamente il pin di GPIO cui vogliamo collegare il nostro LED.

Il nodo che fa al caso nostro è il nodo *rpi – gpio out*, presente nella categoria Raspberry Pi (Figura 1-15).

Posizioniamo il nodo sul workspace ed apriamone la configurazione. Qui dobbiamo semplicemente selezionare il GPIO che vogliamo utilizzare (il GPIO 4 nell’esempio, ma può essere utilizzato qualsiasi GPIO adatto allo scopo) e il tipo di gestione (campo *Type*: Digital output) e, optionalmente, se vogliamo inizializzare il pin ad un certo livello logico, come illustrato in Figura 1-16.

A questo punto non ci rimane altro da fare che creare i collegamenti. La creazione dei collegamenti è molto semplice, è sufficiente posizionarsi con il puntatore del mouse su una porta di un nodo cliccare e mantenere premuto il tasto sinistro, trascinare il puntatore sulla porta che si desidera collegare e rilasciare. Colleghiamo quindi l’output del nodo *inject* all’input del nodo *trigger* e l’output del nodo *trigger* all’input del nodo *rpi – gpio out*, come illustrato in Figura 1-17.

A questo punto possiamo premere il tasto *deploy* per mandare in esecuzione il nostro flow. Si noti che il nodo *rpi-gpio out* fornisce una indicazione dello stato fisico del pin (icona verde nell’angolo in basso a sinistra del nodo, con raffigurato il valore logico assunto dalla linea di I/O che stiamo utilizzando). In Figura 1-18 è rappresentato lo schema delle connessioni da utilizzare, usando una breadboard, un LED e una resistenza da $1\text{k}\Omega$, più dei normali cavetti di collegamento da breadboard. Per i nostri esperimenti abbiamo utilizzato una Raspberry Pi 4 Model B, ma sostanzialmente qualsiasi modello di raspberry è utilizzabile (per maggiori dettagli sul nodo RPI gpio è possibile consultare la relativa pagina sul sito di Node-RED: <https://flows.nodered.org/node/node-red-node-pi-gpio>).

CONCLUSIONI

Siamo arrivati alla conclusione di questa prima puntata di questo corso Node-RED; in questa puntata abbiamo spiegato cos’è Node-RED ed il principio della programmazione flow-based. Inoltre, abbiamo iniziato a prendere confidenza con l’editor e visto i primi semplici esempi su Raspberry PI.

Nella prossima puntata inizieremo ad esaminare i più importanti nodi di base e vedremo alcune particolarità dell’ambiente, come ad esempio i context.

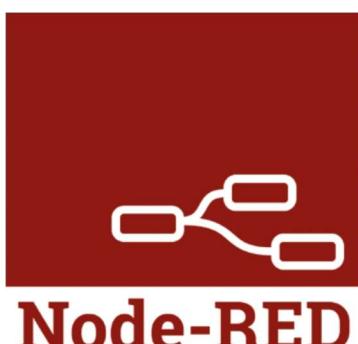


Figura 1-1 - Logo di Node-RED

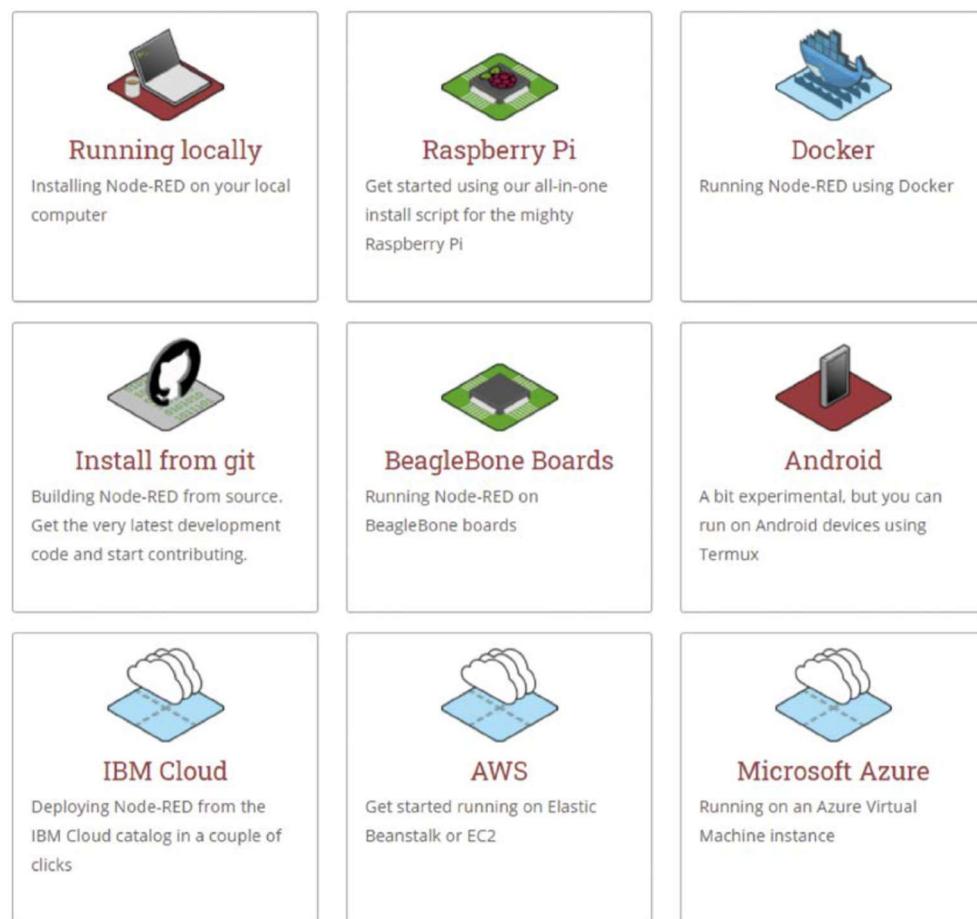


Figura 1-2 - Le varie piattaforme che supportano l'esecuzione di Node-RED.

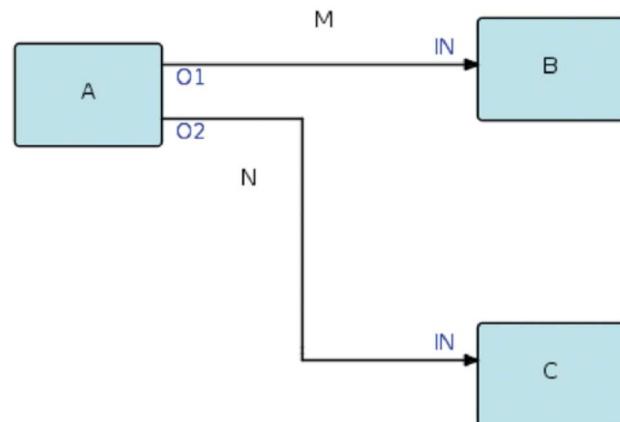


Figura 1-3 - Un semplice diagramma di Flow-Based Programming.

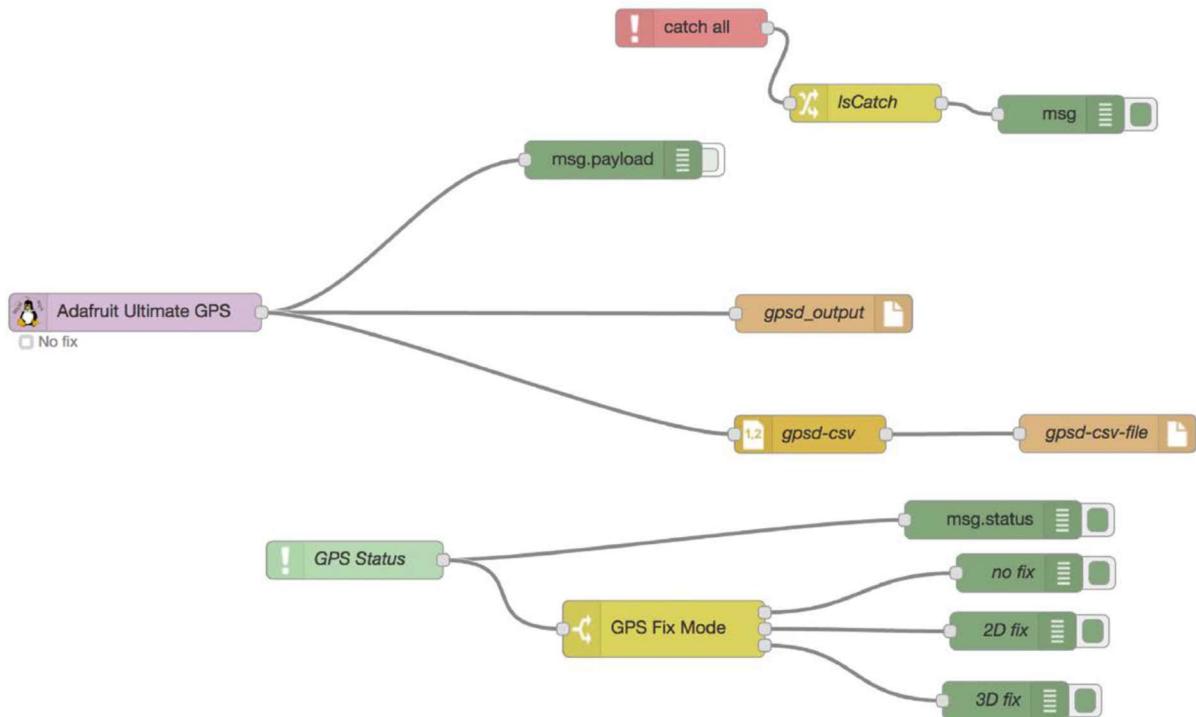


Figura 1-4 - Esempio di Flow in Node-RED.

```

pi@raspberrypi: ~
[ 1.17] login as: pi
[ 1.17] pi@192.168.1.13's password:
Linux raspberrypi 5.10.17-v71+ #1414 SMP Fri Apr 30 13:20:47 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $ 
  
```

The screenshot shows a terminal window titled 'pi@raspberrypi: ~'. It displays the initial boot sequence of a Raspberry Pi OS. The user logs in as 'pi' and enters the password. The system information includes the kernel version '5.10.17-v71+ #1414 SMP Fri Apr 30 13:20:47 BST 2021 armv7l'. It also provides a note about the absence of warranty and a warning about SSH security.

Figura 1-5 - Schermata della console di Raspberry PI Os visualizzata in SSH tramite PuTTY.

```

pi@raspberrypi: ~
Running Node-RED install for user pi at /home/pi on raspbian

This can take 20-30 minutes on the slower Pi versions - please wait.

Stop Node-RED
Remove old version of Node-RED
Remove old version of Node.js
Install Node.js 14 LTS
Clean npm cache
Install Node-RED core
Move global nodes to local
Npm rebuild existing nodes
Install extra Pi nodes
Add shortcut commands
Update systemd script

Any errors will be logged to /var/log/nodered-install.log
All done.
You can now start Node-RED with the command node-red-start
or using the icon under Menu / Programming / Node-RED
Then point your browser to localhost:1880 or http://(your_pi_ip-address):1880

Started : Sun 27 Jun 12:01:29 BST 2021
Finished: Sun 27 Jun 12:05:24 BST 2021
pi@raspberrypi: ~
  
```

Figura 1-6 - Schermata della console dopo l'esecuzione dello script

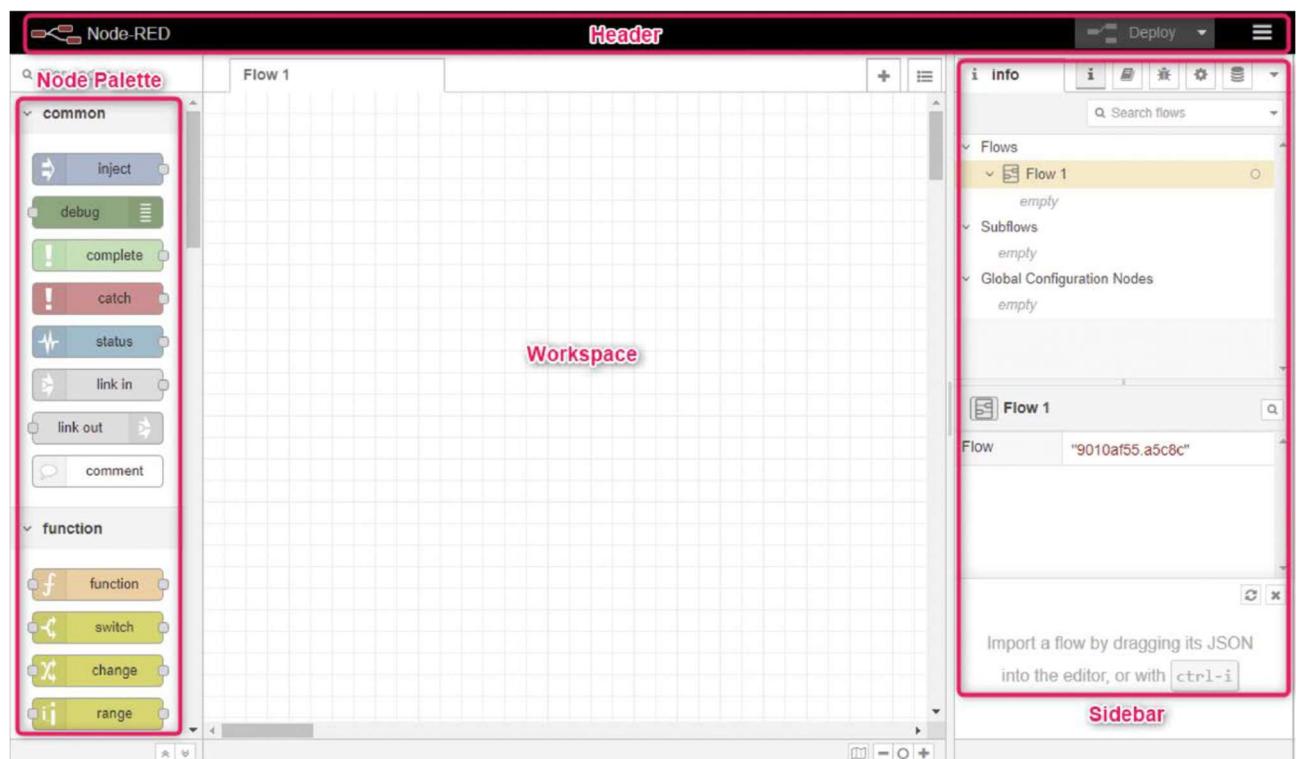


Figura 1-7 - Editor di Node-RED.

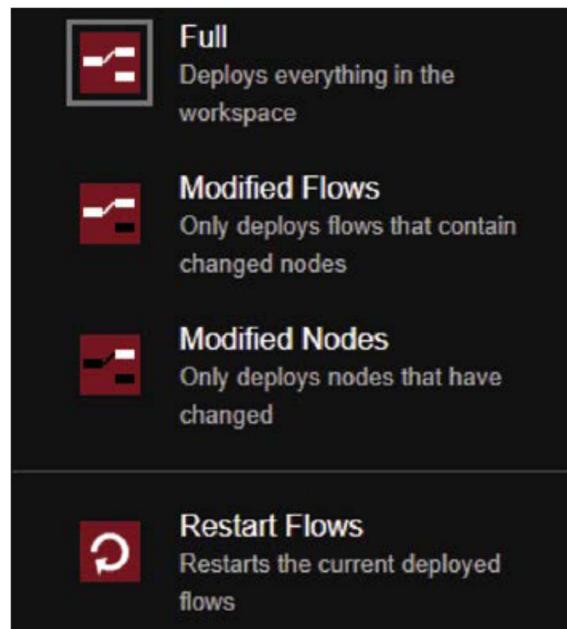


Figura 1-8 - Differenti opzioni di deploy



Figura 1-9 - Footer del workspace con la Navigation view attiva.

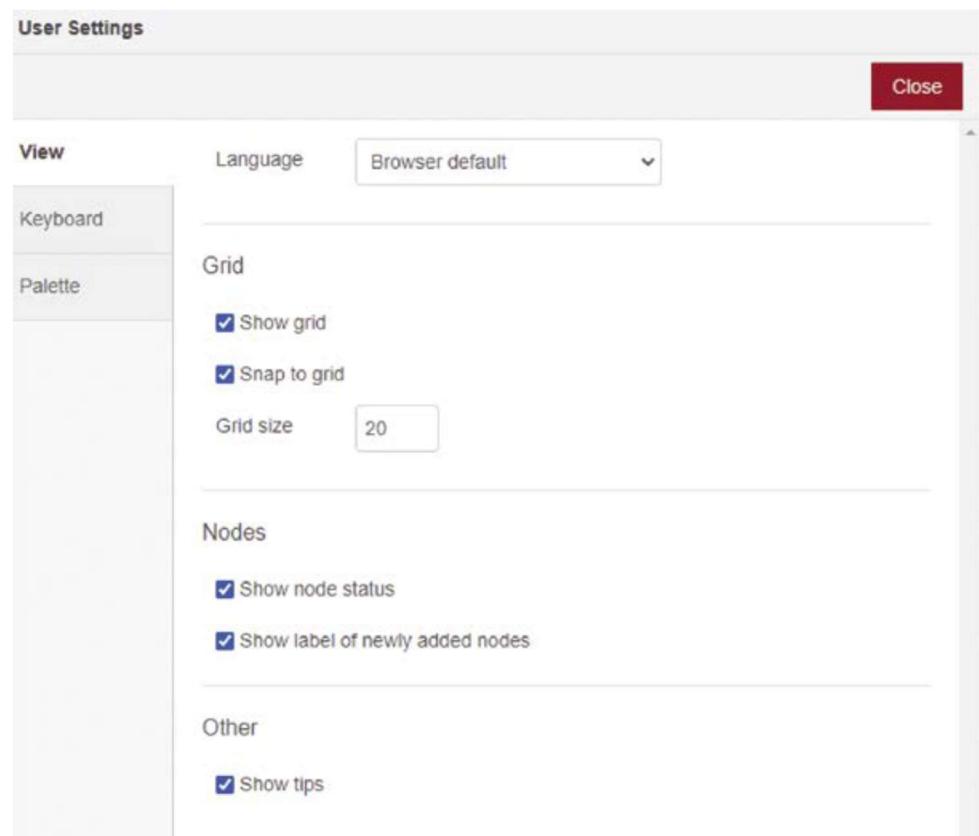


Figura 1-10 - User Settings.



Figura 1-11 - Nodo Inject.

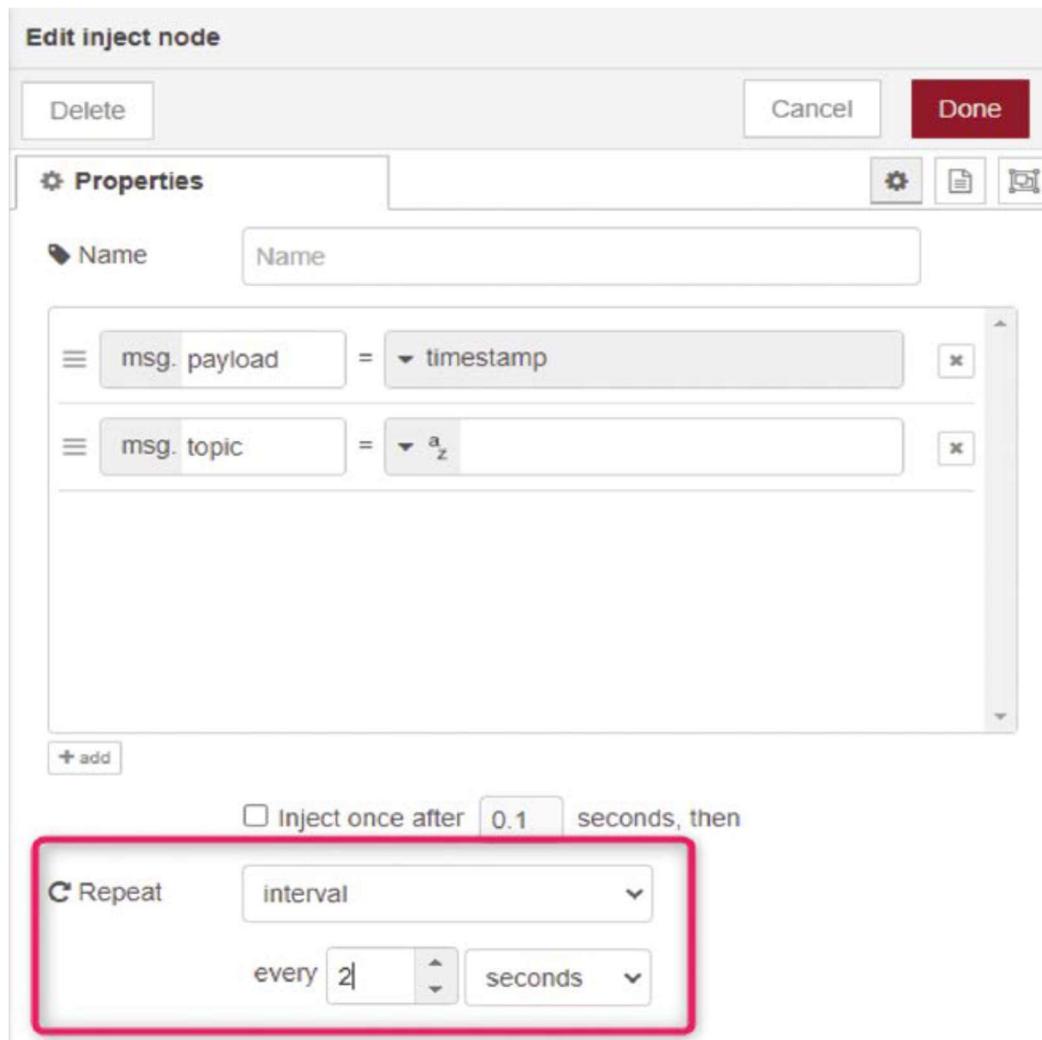


Figura 1-12 - Configurazione del Nodo Inject.



Figura 1-13 - Nodo Trigger.

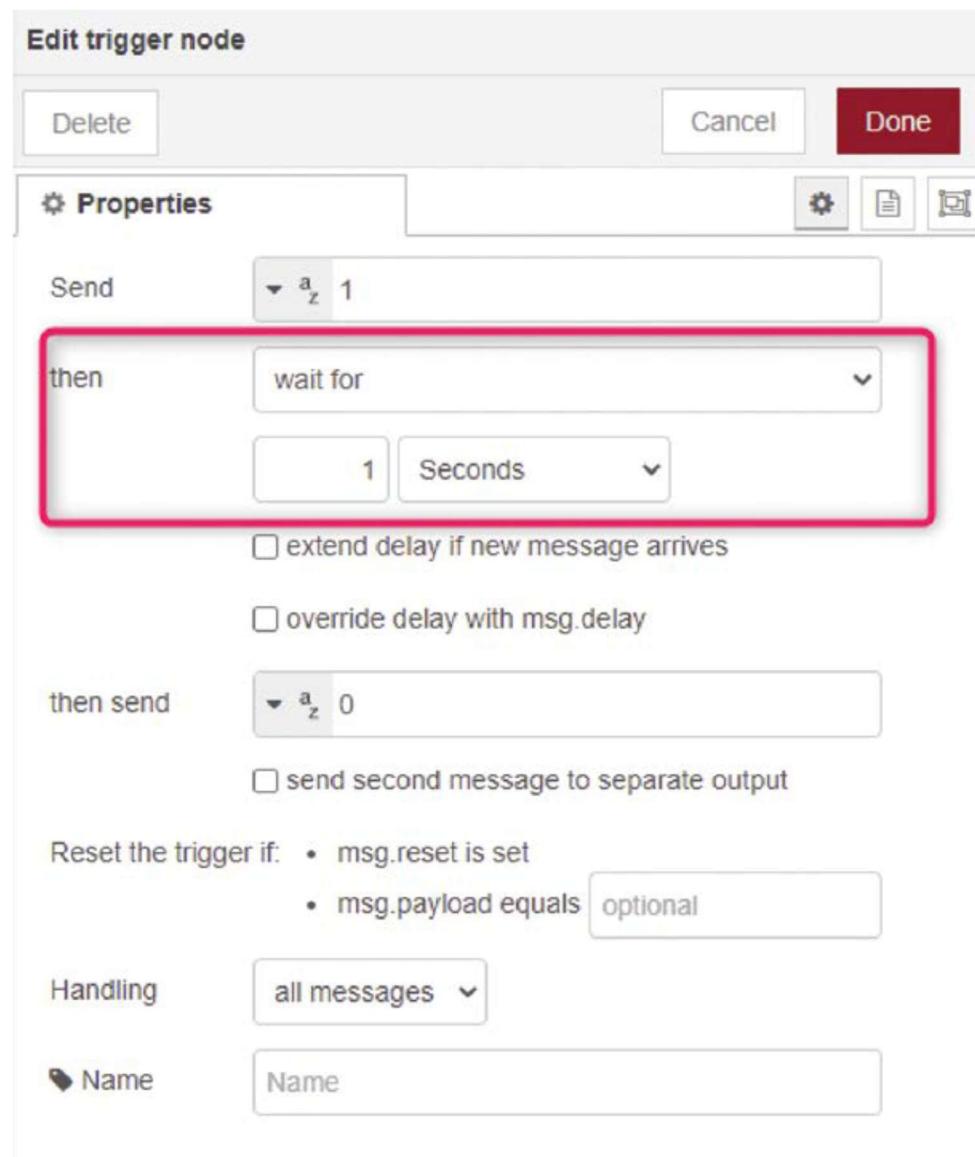


Figura 1-14 - Configurazione del nodo Trigger

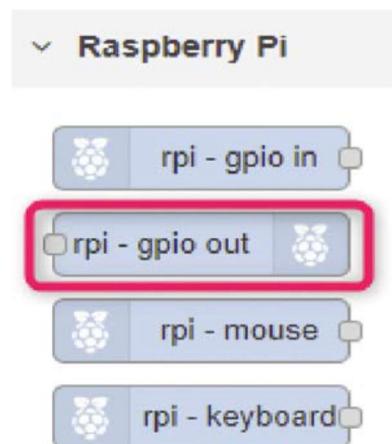


Figura 1-15 - Nodo rpi - gpio out.

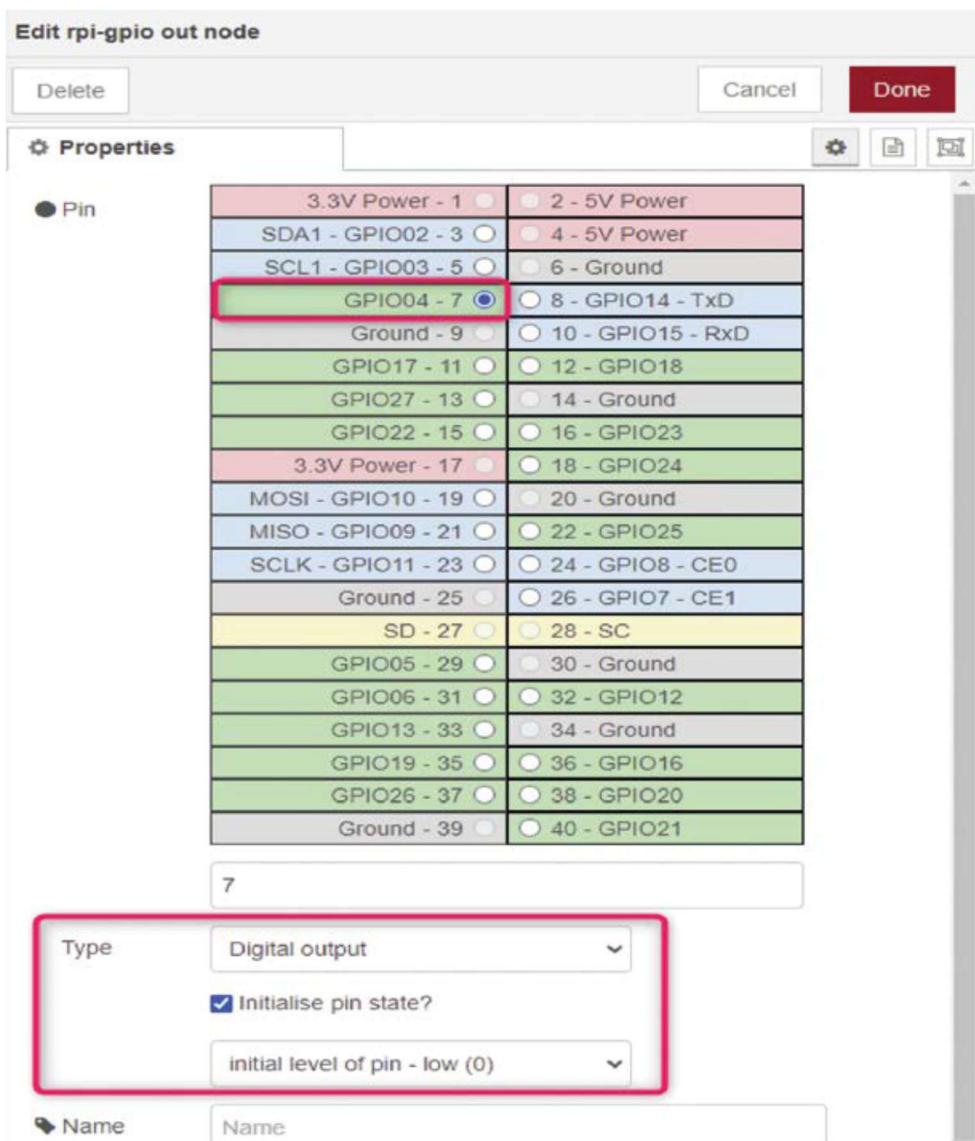


Figura 1-16 - Configurazione del nodo rpi - gpio out

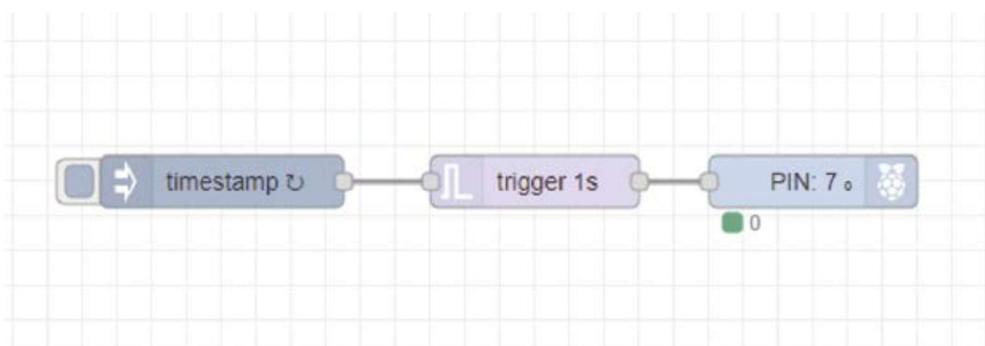


Figura 1-17 - Flow per il lampeggio di un LED a 1Hz.

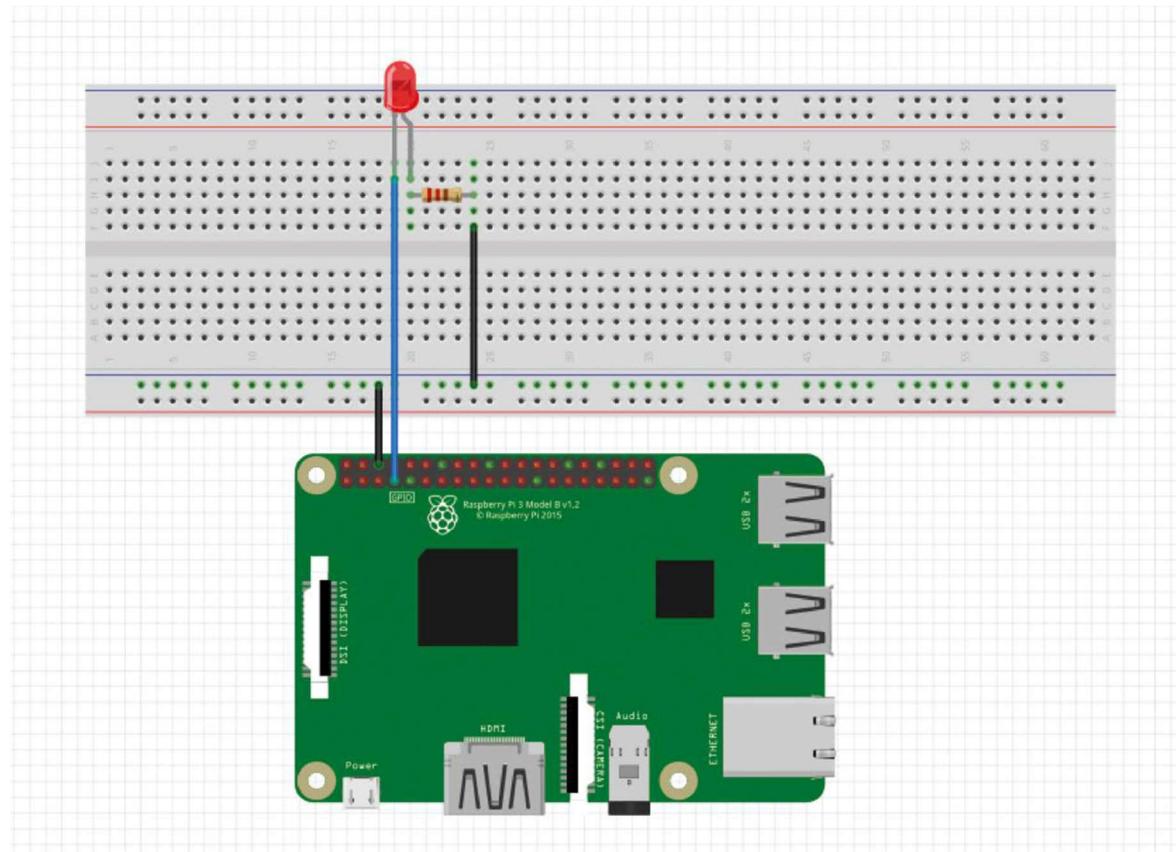


Figura 1-18 - Schema dei collegamenti per il progetto di esempio.

2. Seconda puntata.

Abbiamo iniziato questo corso introducendo, nella prima puntata il tool di flow-based programming Node-RED, descrivendone i concetti di base, quindi passando alla pratica installando l'ambiente ed analizzando le caratteristiche dell'editor grafico. Come di consueto, ci siamo quindi spostati sul lato pratico, realizzando anche un semplice esempio di utilizzo (il classico esempio di Hello World usato nei sistemi embedded, ossia il lampeggio di un LED) su Raspberry Pi. Ritenendo di avervi dato le basi per proseguire proficuamente la trattazione, in questa seconda puntata approfondiremo la nostra conoscenza dell'ambiente concentrandoci sulle principali categorie di nodi a disposizione spieghiamo come aggiungere nuovi nodi installandoli sulla palette ed introduciamo il concetto dei context, che ci permettono di immagazzinare informazioni con diversi gradi di visibilità all'interno dei nostri flow.

FLUSSI, NODI E MESSAGGI

Come abbiamo brevemente accennato nella puntata precedente, gli elementi fondamentali che costituiscono un programma Node-RED sono:

- Flussi;
- Nodi;
- Messaggi.

Espandiamo questi concetti di base, in modo da comprendere meglio queste fondamentali strutture ed il loro funzionamento.

Flussi

Un flusso (o flow) è il principale meccanismo per contenere una struttura connessa di nodi. Al livello dell'editor di Node-RED è rappresentato dal workspace, all'interno del quale i nodi possono essere trascinati (prelevandoli dalla palette) e connessi tra di loro per costituire il codice grafico. Una singola istanza di Node-RED può contenere diversi flows, rappresentati come una serie di tab nella parte alta del workspace stesso. Node-RED mostra sul workspace, di volta in volta, i nodi e le connessioni del flow attivo.

Per aggiungere un nuovo flow al workspace bisogna premere il tasto '+' situato nell'angolo in alto a destra del workspace (alternativamente può essere utilizzata l'opzione Flows → Add del menu principale), come illustrato in Figura 2-1.

Il termine flusso può anche essere utilizzato per indicare informalmente un insieme atomico di nodi connessi, di conseguenza un flusso (tab) può essere costituito da un insieme di flussi (intesi come molteplici set i nodi interconnessi).

In Node-RED un flusso può essere salvato come un file JSON (Javascript Object Notation). Questo permette di memorizzare e scambiare molto facilmente i flows come file JSON. I flussi possono essere esportati ed importati tramite le due opzioni "Import" ed "Export" presenti sul menu principale (Figura 2-2). Eseguendo un doppio click sul tab di un flusso si accede alla sua finestra delle proprietà, dalla quale è possibile eseguire una serie di operazioni aggiuntive, come rinominare il flusso, abilitarlo o disabilitarlo o anche cancellarlo (Figura 2-3). Sempre dalla finestra delle proprietà è possibile inserire una descrizione del flusso, in modo da creare una utile documentazione utente del programma che stiamo sviluppando. La descrizione usa la sintassi di MarkDown e compare nella sezione info della sidebar.

Subflows

I flow possono anche essere collassati in un singolo nodo ed in questo caso si parla di **subflows**. I subflows sono usati per ridurre la complessità di un flow o per impacchettare un flow (inteso come gruppo di nodi) all'interno di un singolo nodo da riutilizzare più volte in altri flows. Una volta creati, i subflows appaiono nella palette come un nodo disponibile e possono essere aggiunti al workspace come i nodi standard.

Nodi

Un flow Node-RED è popolato da Nodi (l'elemento principale di Node-RED) e relative connessioni. I nodi costituiscono le parti eseguibili di un flow e si scambiano messaggi tramite wires (fili) che vengono collegati alle porte di ingresso e di uscita di un nodo. I nodi sono triggerati sia ricevendo un messaggio su una porta d'ingresso o tramite un evento interno (come ad esempio lo scadere di un timer, il cambio di stato di una line digitale, o una richiesta proveniente da un canale di comunicazione). Quando viene triggerato, il nodo elabora il messaggio ricevuto o l'evento interno, e successivamente, può inviare un messaggio ai nodi successivi del flow. In qualche caso i nodi hanno degli elementi integrati nella loro icona, con i quali l'utente può interagire direttamente dal flow. Questi elementi possono essere pulsanti, elementi di output, eccetera.

Ci sono tre modi per aggiungere un nodo ad un flow:

- Prelevandolo dal Node Palette
- Tramite la quick-add dialog box
- Tramite la funzione Import

La prima opzione è quella più comune: si seleziona il nodo dalla Node Palette premendo e mantenendo premuto il pulsante sinistro del mouse e lo si trascina sul workspace, rilasciando il pulsante per posizionare il nodo.

Per utilizzare la quick-add dialog box invece è sufficiente tenere premuto il tasto Ctrl e cliccare con il tasto sinistro all'interno del workspace. Questo farà comparire una finestra di selezione come quella riportata in Figura 2-4 (la quick-add dialog box, appunto...) dalla quale è possibile selezionare il nodo da aggiungere. L'opzione di import infine permette di importare un nodo (o più comunemente un intero flow) precedentemente salvato sottoforma di file JSON (tramite la funzione duale export). Esattamente come i flows, anche i nodi dispongono di una finestra delle proprietà, alla quale si può accedere eseguendo un doppio click sul nodo specifico (o alternativamente premendo il tasto invio quando il nodo ha il focus). La finestra delle proprietà di un nodo è divisa in tre sezioni principali, che descriviamo qui di seguito.



Properties: da questa sezione è possibile editare le proprietà specifiche del nodo in questione. Questa sezione varia da nodo a nodo, poiché nodi differenti hanno differenti opzioni di configurazione e proprietà.



Description: da questa sezione è possibile editare la documentazione del nodo, formattata con sintassi MarkDown. La descrizione inserita in questa sezione comparirà nella sezione info della sidebar, come documentazione del nodo. Esattamente come per i flows, questa funzionalità permette di creare una documentazione utente molto ben fatta dei nostri programmi.



Appearance: da questa sezione è possibile customizzare l'aspetto grafico del nodo. Ad esempio è possibile definire se visualizzare o meno la label del nodo, è possibile selezionarne l'icona (nel caso non si voglia usare quella di default) ed inoltre è possibile definire delle label specifiche per ogni porta.

La Figura 2-5 rappresenta le tre sezioni Properties, Description ed Appearance della finestra delle proprietà per un nodo Inject.

Configuration Nodes

Non sempre i nodi sono visibili sul workspace, infatti esiste un tipo particolare di nodo, chiamato configuration node, che non ha una parte grafica (icona e porte di connessione), ma costituisce un insieme di configurazioni che possono essere riutilizzate da più nodi in un flow. I nodi che necessitano un configuration node per funzionare ne permettono la creazione direttamente dalla loro finestra delle proprietà. Ad esempio i nodi serial in e serial out, che necessitano di un configuration node serial port (che contiene le opzioni di configurazione della porta seriale, che possono essere utilizzate da diverse istanze dei nodi di comunicazione seriale), ne permettono la creazione (o, se il nodo è già esistente,

l'associazione) direttamente dalla loro finestra delle proprietà, come visibile in Figura 2-6 e Figura 2-7. Non avendo parte grafica, la finestra delle proprietà dei configuration nodes è costituita dalla sola parte Properties. Una lista completa dei configuration nodes presenti può essere ricavata dalla sezione configuration nodes della sidebar.

Messaggi

Durante il funzionamento di un flow, i vari nodi che lo compongono si scambiano messaggi, e questa è una caratteristica peculiare di Node-RED e della FBP. I messaggi che i nodi si cambiano sono semplici oggetti JavaScript che possono avere qualunque set di proprietà.

Tipicamente una proprietà denominata “payload” è sempre presente, ed è la proprietà con cui più frequentemente i nodi lavorano, ma possono essercene molte altre.

Node-RED inoltre aggiunge ad ogni messaggio una proprietà “_msgid”, che sostanzialmente identifica in maniera univoca il messaggio all'interno di un flow (Figura 2-8).

Il valore di una proprietà di un messaggio può essere un qualsiasi tipo JavaScript, ad esempio:

- Booleano – true/false
- Numerico – 1, 123.4
- Stringa – “Hello World”
- Array – [1,2,3,4,5,6,7,8,9]
- Object – {"a" : 1, "b" : 2}
- NULL

CORE NODES

Come abbiamo visto in precedenza, l'elemento principale di un flow Node-RED sono i nodi. Node-RED contiene una ricca dotazione di nodi predefiniti e moltissimi altri nodi possono essere aggiunti alla paletta.

Per ragioni di spazio è impossibile descrivere nel dettaglio tutti i nodi a disposizione nell'ecosistema di Node-RED, nonché anche poco utile in pratica, dato che si tratta di un ambiente in continua evoluzione.

Esiste tuttavia un set di nodi standard, a cui ci si riferisce spesso con il termine “Core Nodes” che è importante conoscere (Figura 2-9). Questi nodi sono:

- Inject;
- Debug;
- Function;
- Change;
- Switch;
- Template.

Qui di seguito li descriviamo uno per uno.

Inject

Il nodo Inject permette di inviare messaggi all'interno di un flow, sia manualmente (interagendo con il pulsante integrato nell'icona del nodo), che automaticamente. Il messaggio inviato può avere qualsiasi proprietà si desideri e le proprietà possono essere settate in vari modi. Ad esempio una proprietà di un messaggio del nodo inject può essere:

- uno dei tipi base supportati da JavaScript (ad esempio un boolean, un valore numerico, una stringa, un array...);
- un timestamp;
- una stringa JSON;
- una espressione JSONata.

In questo modo possono essere iniettati nel nostro flow messaggi contenenti qualsiasi dato.

Di conseguenza il nodo inject è molto utilizzato per il debug e per testare il funzionamento di altri nodi, pur trovando anche altri interessanti utilizzi (ad esempio per l'inject periodico di valori all'interno del flow, come abbiamo visto nell'esempio pratico della scorsa puntata).

Debug

Il nodo Debug permette di visualizzare i messaggi che riceve dalla sua porta in ingresso nella visuale di debug della sidebar.

La finestra di debug della sidebar permette di visualizzare in maniera strutturata le informazioni contenute nel messaggio, rendendo quindi il nodo di debug molto utile per il debugging dei flow (co006De era anche facilmente intuibile dal suo nome).

Function

Il nodo Function (Figura 2-10) permette di scrivere del codice JavaScript che agisce direttamente sul messaggio passato al nodo stesso.

Il messaggio è passato in un oggetto chiamato msg, e possono essere creati quanti output si vuole.

Dalle versioni più recenti di Node-RED è possibile importare moduli JavaScript da utilizzare all'interno del blocco ed è inoltre possibile invocare codice JavaScript all'avvio ed allo stop del nodo (ad esempio allo scopo di inizializzare delle variabili di stato e via di seguito).

Change

Il nodo Change è utilizzato per modificare una proprietà di un messaggio senza dover ricorrere al nodo function.

Ogni nodo può essere configurato per eseguire una serie di operazioni in ordine.

Le operazioni disponibili sono:

- **Set**: settare una proprietà ad un certo valore. Il valore può essere una varietà di tipi differenti, o può essere preso ad un'altra proprietà.
- **Change**: cercare e rimpiazzare parte di una proprietà di un messaggio.
- **Move**: muovere o rinominare una proprietà.
- **Delete**: cancellare una proprietà

Switch

Il nodo Switch permette di indirizzare messaggi verso differenti porte in uscita, in base alla valutazione di regole interne al nodo. A tale riguardo va precisato che sono previsti i quattro tipi di regola descritti qui di seguito.

- **Valore**: Il valore di una specifica proprietà può essere controllato con espressioni comparative o booleane (ad esempio maggiore di, minore di, uguale, vero falso, etc..).
- **Sequenza**: la sequenza di arrivo dei messaggi può essere controllata, estraendo solo alcuni elementi della sequenza stessa, dalla testa, dalla coda o da un numero arbitrario di elementi intermedi. Vengono triggerate le porte di uscita per cui si verifica un match rispetto al controllo di sequenza.
- **JSONata**: il messaggio viene controllato per mezzo di una espressione JSONata (viene fornito un apposito editor per impostare e testare la query) e viene triggerato lo specifico output solo se l'espressione ritorna true.
- **Otherwise**: questo è il classico caso di default delle espressioni switch dei linguaggi classici, viene triggerato se nessuna delle precedenti regole viene soddisfatta.

Template

Il nodo Template, infine, è un nodo che è in grado di generare un output testuale utilizzando le proprietà dei messaggi per riempire un template usando la sintassi del Mustache template language (maggiori informazioni sul Mustache template language possono essere reperite a questo indirizzo web:

<https://mustache.github.io/mustache.5.html>).

Questo nodo è molto utile per generare testi formattati in modi differenti partendo da tags contenuti su delle message properties.

Ad esempio, un nodo template con all'interno questa dicitura:

Il messaggio di testo contenuto nel payload è: {{payload}}

A fronte della ricezione di un messaggio contenente nella property payload la stringa "Ciao Mondo!!!", genererebbe il seguente output testuale:

Il messaggio di testo contenuto nel payload è: Ciao Mondo!!!

che andrà pertanto a sostituire al tag {{payload}} il contenuto di msg.payload.

AGGIUNGERE NODI ALLA PALETTE

Oltre a questi nodi di base, Node-RED contiene diversi altri nodi pre-installati di default, aventi interessanti funzioni aggiuntive, ma che non possono essere descritti uno per uno per ovvie ragioni di spazio.

Ma non è tutto, infatti la natura fortemente open dell'ambiente e la sua vasta community hanno portato alla produzione di una quantità di nodi aggiuntivi, che possono essere installati. Sarebbe assolutamente impossibile descriverli tutti, trattandosi di diverse migliaia di nodi, mentre può essere utile descrivere brevemente la procedura per aggiungere dei nodi alla palette che contiene quelli installati per impostazione predefinita.

L'aggiunta di un nuovo nodo può essere fatta selezionando l'opzione "Manage Palette" presente nelle opzioni di Node- RED (Figura 2-11).

Una volta cliccato sull'opzione si aprirà una finestra di dialogo, ed in questa finestra bisogna selezionare il tab "Install" e poi cercare il nodo desiderato usando l'apposito motore di ricerca integrato.

Supponiamo, ad esempio, di voler installare un nodo per la gestione della camera pi (<https://www.raspberrypi.org/products/camera-module-v2>).

Per farlo selezioniamo l'opzione "Manage Palette", e nella finestra di dialogo che ci viene presentata selezioniamo il tab install e scriviamo "camerapi" nell'apposita textbox per la ricerca, come illustrato in Figura 2-12.

Selezionando il pulsante "Install" di uno dei risultati della ricerca, Node-RED procederà all'installazione del nodo. Nel caso dell'esempio appena visto, ossia l'installazione del nodo camerapi, al termine dell'operazione troveremo il nodo di Figura 2-13 tra i nodi della palette.

COSA SONO I CONTEXT

Passiamo adesso all'ultimo argomento di questa puntata, ossia i Context: Node-RED prevede un sistema per lo stoccaggio di informazioni in memoria che possono essere utilizzate da diversi nodi oppure da differenti istanze di esecuzione di un nodo stesso: si tratta dei context.

Ebbene, esistono tre diversi tipi di context in Node-RED, cui sono associate tre differenti visibilità (o scope):

- **Node:** visibile solo da uno specifico nodo;
- **Flow:** visibile all'interno di uno specifico flow;
- **Global:** visibile globalmente da parte di tutti i nodi di Node-RED.

Il tipo specifico di scope da utilizzare dipende da quello che si desidera fare con il context: se, a titolo di esempio, il valore di un determinato dato deve essere visibile solo all'interno di un function node, allora è sufficiente il Node Context. Se invece il dato deve

essere visibile a più nodi, allora sarà necessario utilizzare un Flow Context oppure un Global Context. I context vengono salvati in RAM di default, quindi si comportano in maniera simile a delle variabili globali, ma tuttavia non sono persistenti ad un riavvio di Node-RED. Esiste comunque la possibilità di cambiare lo store type dei context per far sì che il loro valore venga salvato sul file system anziché in RAM. I context possono essere manipolati in diversi modi. Il modo più semplice è quello di utilizzare un change node, ad esempio la regola in Figura 2-14 salva il contenuto presente in msg.payload sul context globale myContextData.

Esistono inoltre delle API che permettono di manipolare (leggere e scrivere) i context da un function node. Abbiamo 3 tipologie di accesso per ognuno dei 3 tipi di context, ed ognuna di queste può accedere a metodi di set e get:

- **Context:** node context;
- **Flow:** flow context;
- **Global:** global context.

Ad esempio, per leggere il valore myContextData con node scope da un function node possiamo scrivere la seguente riga di codice all'interno di un nodo function:

```
var myData = context.get("myContextData");
```

Invece per scrivere il valore della nostra variabile locale, una volta che è stato aggiornato, sul context, possiamo utilizzare la seguente istruzione:

```
context.set("myContextData", myData);
```

I due esempi appena proposti utilizzano il node scope (context), ma sono validi anche per il flow e global context, basterebbe cambiare la keyword context appunto con flow o global.

ESEMPIO PRATICO

Passiamo adesso all'esempio pratico di questa puntata.

Ciò che vogliamo fare è nuovamente far lampeggiare un LED, ma stavolta applicando alcuni dei nuovi concetti esposti in questa puntata.

Utilizzeremo ancora una volta un nodo inject per creare la temporizzazione ed il medesimo nodo rpi-gpio-out per controllare la linea di uscita, ma stavolta, al posto del nodo trigger, faremo uso di un function node, all'interno del quale scriveremo un po' di codice javascript per generare la logica necessaria al lampeggio ed utilizzeremo un context per immagazzinare il valore booleano corrispondente allo stato del LED tra un'attivazione del nodo function e la successiva.

Iniziamo inserendo sul workspace i nodi necessari al nostro progetto, ossia:

- un nodo **inject**;
- un nodo **function**;
- un nodo rpi-gpio-out.

e collegandoli poi tra di loro logicamente come proposto nella Figura 2-15.

I nodi inject e rpi-gpio-out devono essere configurati esattamente come nell'esempio pratico che vi abbiamo proposto nella puntata precedente del corso, così come rimane la stessa la configurazione hardware da utilizzare (il pin al quale è collegato il LED è il GPIO4).

Il nodo inject ha un'attivazione periodica e genera un messaggio ogni secondo; il contenuto di tale messaggio è irrilevante, in quanto non viene utilizzato dal nodo function, ma serve soltanto ad attivarlo.

Ad ogni attivazione, il nodo function deve generare in uscita un messaggio contenente sulla property payload lo stato del pin (e quindi del LED); questo viene realizzato tramite il codice javascript riportato nel Listato 1.

Listato 1 - Codice javascript per il LED blink

```
// Create the variable Led and load it with the context
var Led = context.get("LED");
// Invert the variable value
```

```

Led = !Led;
// Store back the inverted value into the context
context.set("LED", Led);
// Associate the payload with the variable value (Led status)
msg.payload = Led;
return msg;

```

Come è possibile vedere da tale elenco, viene creata una variabile Led, che viene caricata con il contenuto del context LED mediante l'istruzione context.get(LED). Il context ha uno scope di tipo node, dal momento che non deve essere utilizzato da nessun altro nodo del flow al di fuori del nodo function stesso.

Di fatto questo context funziona come una variabile globale che immagazzina lo stato del LED tra una invocazione del nodo function e la successiva.

Nella riga successiva lo stato della variabile Led (appena caricato) viene invertito, allo scopo di generare il lampeggio desiderato.

A questo punto il nuovo stato viene salvato sul context mediante l'istruzione context.set("LED", Led); fatto questo, non resta altro da fare che copiare il valore della variabile Led su msg.payload e "ritornare" il messaggio in uscita.

Se lanciamo il flow in esecuzione mediante il pulsante deploy, utilizzando lo stesso setup dell'esempio pratico che vi abbiamo proposto nella prima scorsa puntata, vedremo il medesimo risultato, ossia il LED collegato al GPIO4 lampeggiare alla frequenza di 1Hz.

Se colleghiamo un nodo debug all'uscita del nodo function (Figura 2-16) vedremo anche apparire un log che illustra il risultato pratico del codice che abbiamo inserito nel nodo, ossia una successione di messaggi con alternanza dello stato booleano tra true e false (che va a pilotare la linea digitale 4 della scheda Raspberry Pi), come visibile in Figura 2-17.

CONCLUSIONI

In questa seconda puntata del corso abbiamo approfondito la nostra conoscenza di Node-RED, soffermandoci maggiormente sui suoi concetti fondamentali, come i flow, i nodi e i messaggi. Abbiamo inoltre analizzato alcuni nodi fondamentali installati per impostazione predefinita con l'ambiente (si tratta dei cosiddetti Core Nodes), visto come aggiungere nodi alla paletta ed introdotto i context.

Infine, abbiamo concluso proponendo il solito esempio pratico, allo scopo di farvi "prendere la mano" con i concetti esposti e farveli fissare più facilmente nella mente .

Nella prossima puntata inizieremo a concentrare la nostra attenzione sull'interazione con il mondo esterno quindi vedremo come fare per controllare alcune periferiche della scheda Raspberry PI utilizzando le funzionalità offerte da Node-RED.

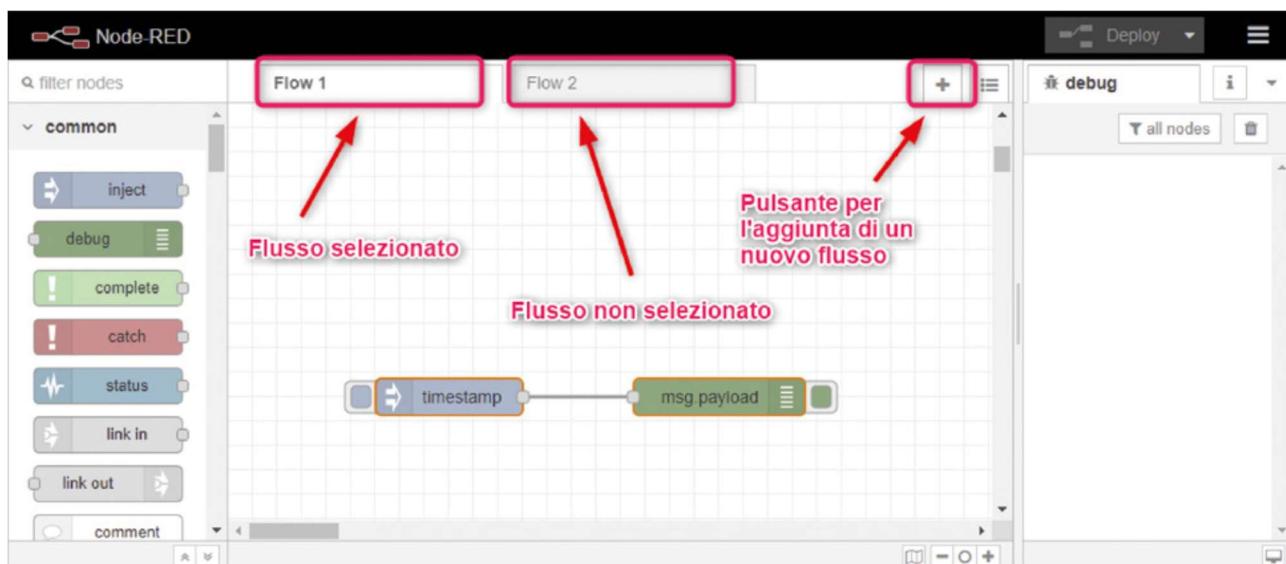


Figura 2-1 - Tab flussi ed inserimento di un nuovo flusso.

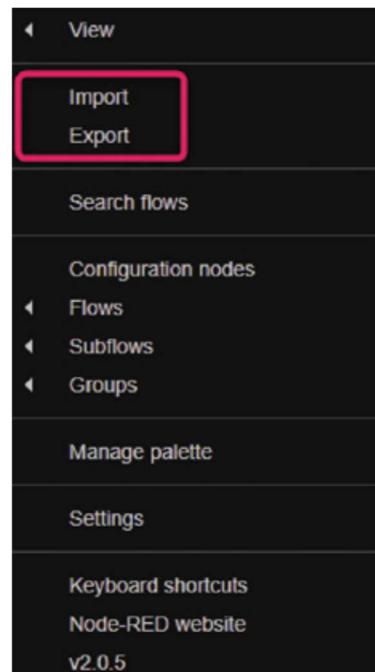


Figura 2-2 - Opzioni di Import ed Export di Node-RED.

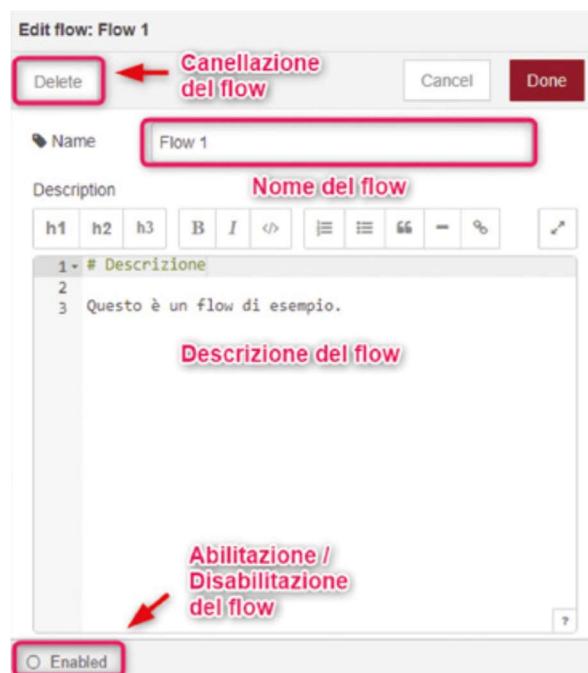


Figura 2-3 - Finestra delle proprietà di un flow.

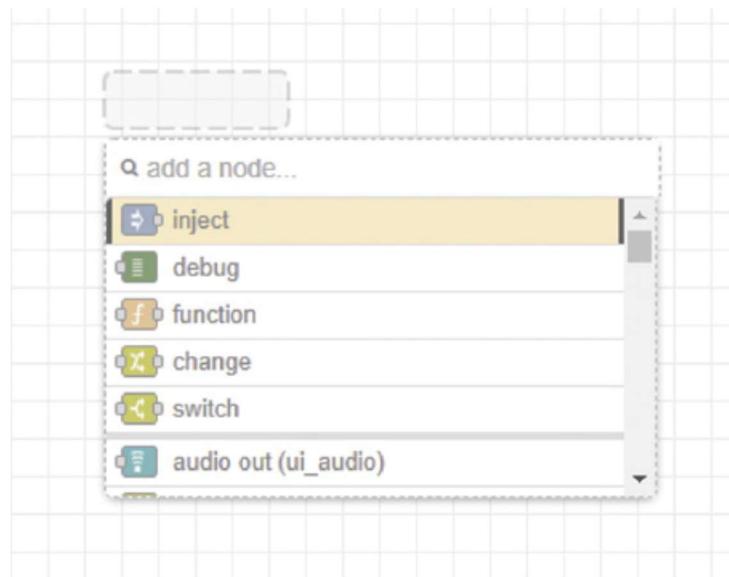


Figura 2-4 - Quick-add dialog box.

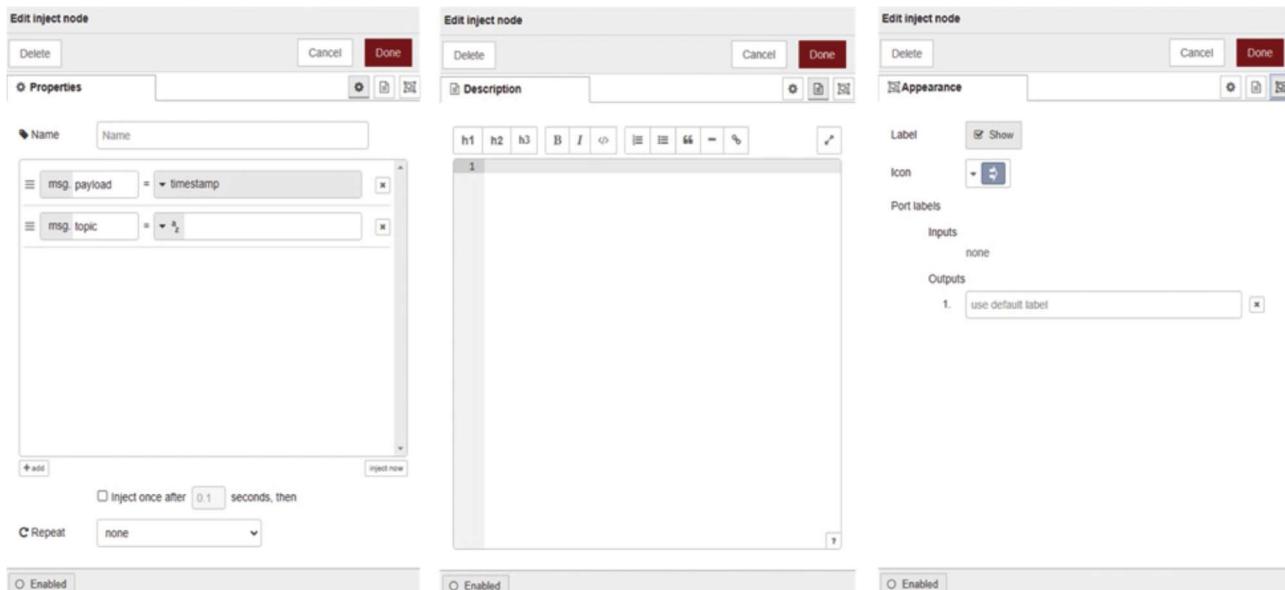


Figura 2-5 - Finestra delle proprietà (sezioni Properties, Description ed Appearance) per un nodo Inject.

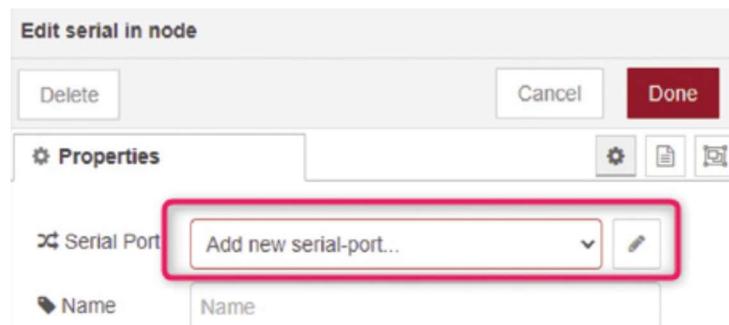


Figura 2-6 - Creazione del configuration node serial port dalla finestra di proprietà del nodo serial in.

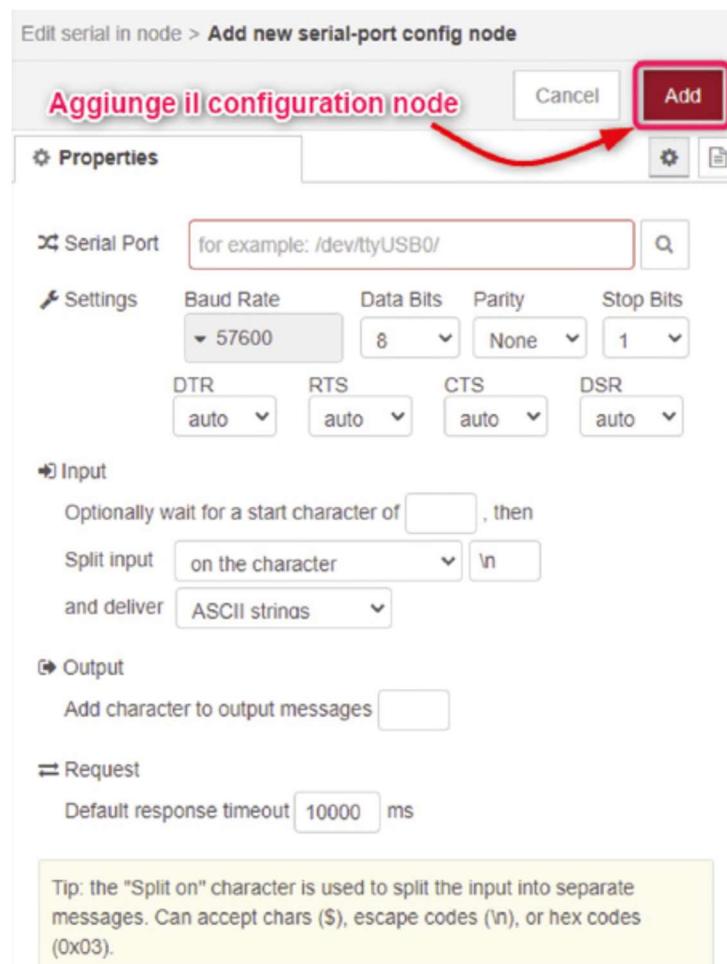


Figura 2-7 - Finestra delle proprietà del configuration node serial port.

```
{
  "_msgid": "12345",
  "payload": "..."
}
```

Figura 2-8 - Esempio di messaggio Node-RED.

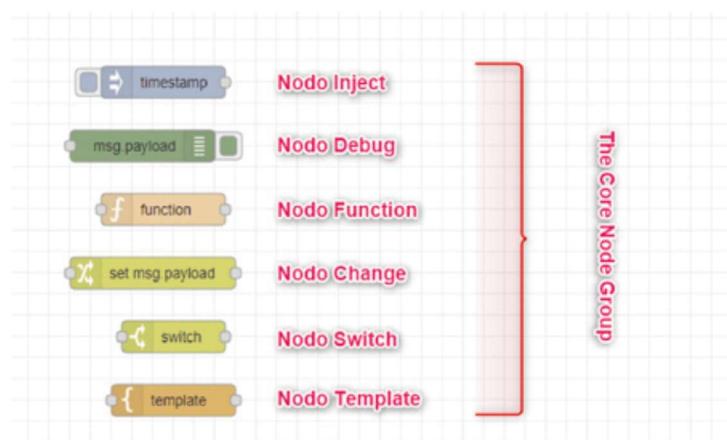


Figura 2-9 - I Core Nodes.

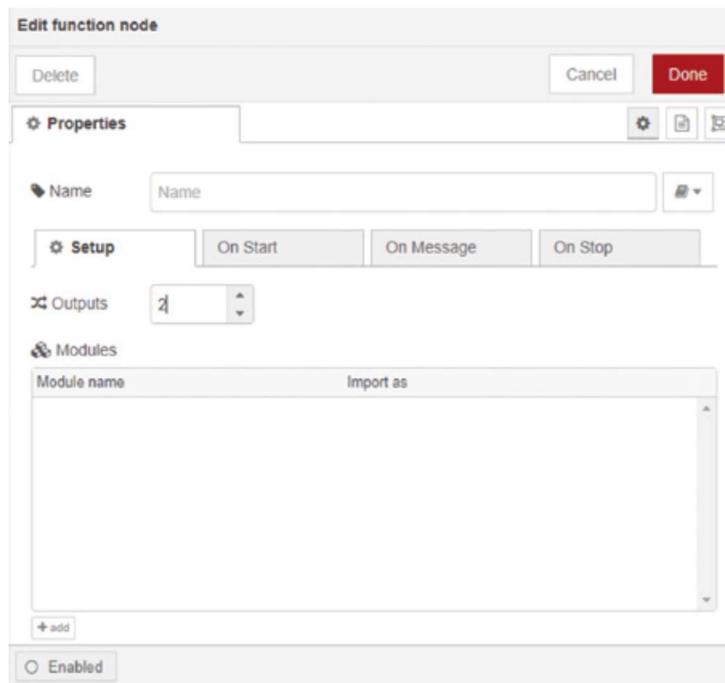


Figura 2-10 - Finestra delle proprietà del nodo Function.

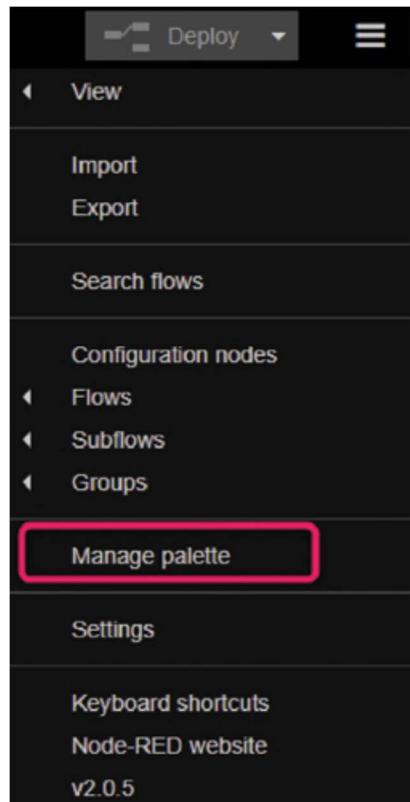


Figura 2-11 - Opzione Manage Palette tra le opzioni di Node-RED.

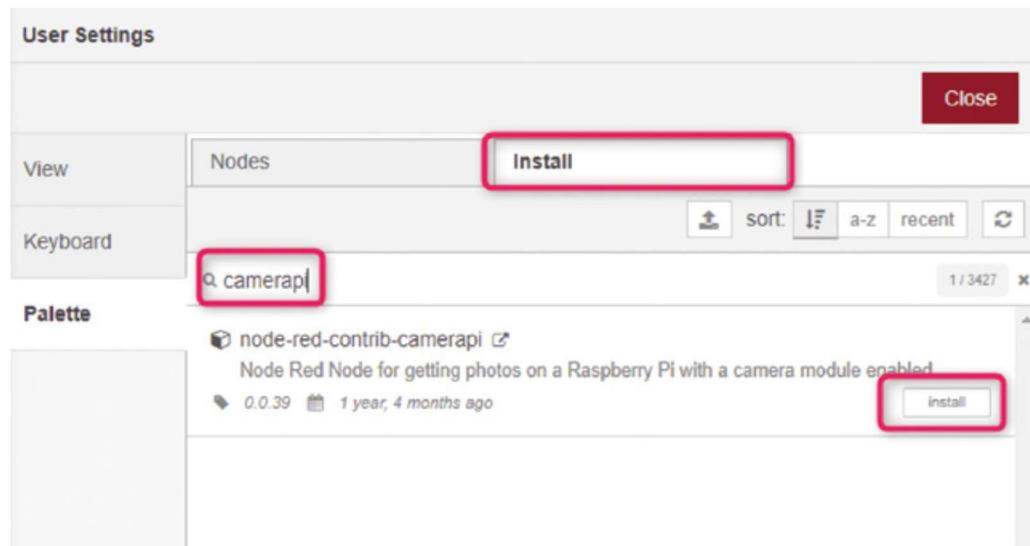


Figura 2-12 - Installazione del nodo “node-red-contrib-camerapi”.



Figura 2-13 - Nodo camerapi installato.

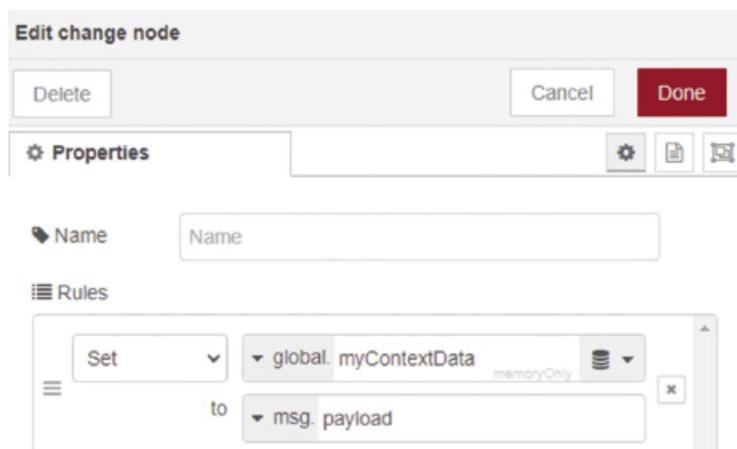


Figura 2-14 - Esempio di utilizzo di un context con Change node.

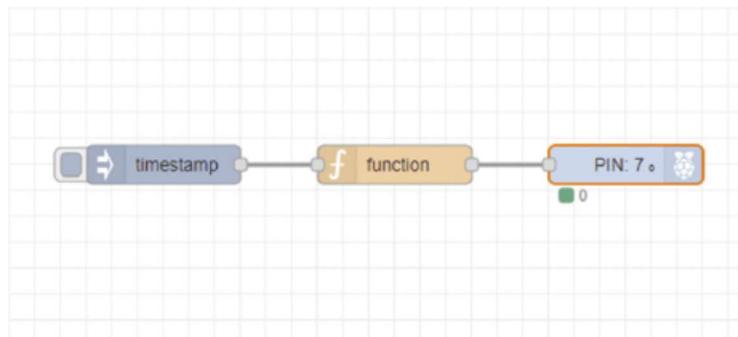


Figura 2-15 - Flow per Led Blink con nodo function.

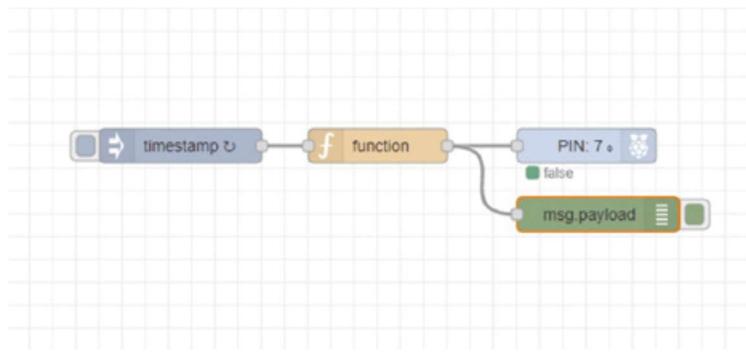
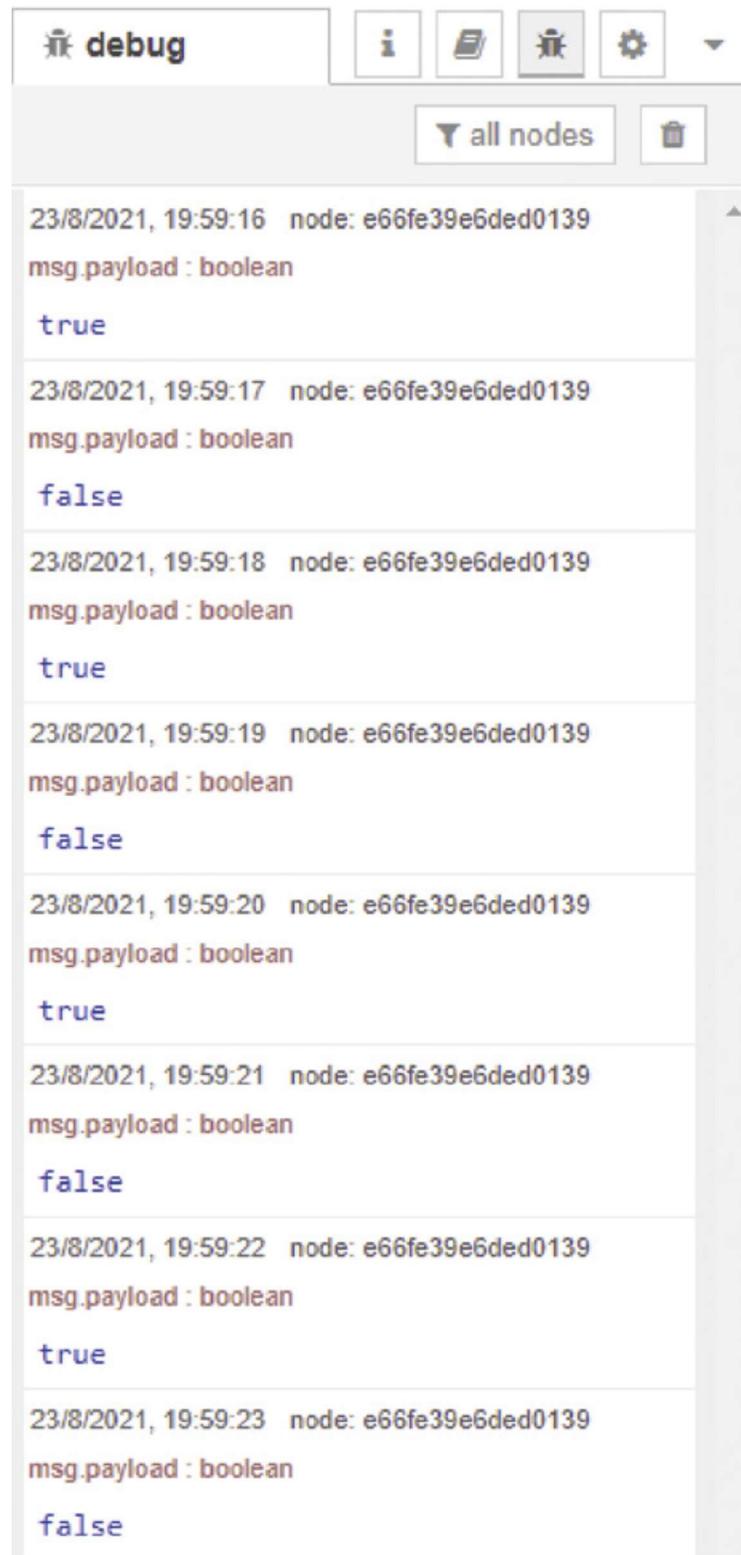


Figura 2-16 - Inserimento del nodo Debug.



The screenshot shows the Node-RED debug log interface. At the top, there is a toolbar with icons for info, copy, paste, and settings. Below the toolbar, there are buttons for 'all nodes' and a trash can. The main area displays a list of log entries:

- 23/8/2021, 19:59:16 node: e66fe39e6ded0139
msg.payload : boolean
`true`
- 23/8/2021, 19:59:17 node: e66fe39e6ded0139
msg.payload : boolean
`false`
- 23/8/2021, 19:59:18 node: e66fe39e6ded0139
msg.payload : boolean
`true`
- 23/8/2021, 19:59:19 node: e66fe39e6ded0139
msg.payload : boolean
`false`
- 23/8/2021, 19:59:20 node: e66fe39e6ded0139
msg.payload : boolean
`true`
- 23/8/2021, 19:59:21 node: e66fe39e6ded0139
msg.payload : boolean
`false`
- 23/8/2021, 19:59:22 node: e66fe39e6ded0139
msg.payload : boolean
`true`
- 23/8/2021, 19:59:23 node: e66fe39e6ded0139
msg.payload : boolean
`false`

Figura 2-17 - Debug Log.

3. Terza puntata.

Proseguiamo lo studio del tool di flow-based programming orientato all'IoT ed alla connettività. In questa terza puntata analizziamo la gestione delle periferiche della Raspberry Pi tramite Node-RED, approfittandone per l'implementazione di nuovi esempi pratici.

Nelle precedenti puntate abbiamo introdotto i concetti fondamentali di Node-RED, presentando l'ambiente grafico ed i principali nodi predefiniti, implementando anche i primi, semplici, esempi. In questa puntata muoviamo un passo avanti e iniziamo a scoprire come gestire le periferiche della Raspberry Pi tramite Node-RED. Come abbiamo già accennato in precedenza, Node-RED è un ambiente di programmazione che supporta una moltitudine di piattaforme, dal momento che le dipendenze hardware vengono completamente astratte. È chiaro, però, che tali dipendenze esistono e la gestione dell'hardware specifico varia a seconda del target utilizzato (ossia è diverso tra una Raspberry Pi e una BeagleBoard).

Tali differenze sono indirizzate in maniera diversa a seconda del nodo utilizzato; in alcuni casi i nodi sono specifici, ossia sono legati alla piattaforma hardware (è il caso di alcuni nodi Raspberry Pi). In altri casi invece i nodi sono strutturati per supportare diverse piattaforme, e fa parte della configurazione del nodo la selezione della piattaforma specifica. Nelle puntate future del corso, utilizzeremo sempre la Raspberry Pi come piattaforma hardware per i nostri test e quindi faremo spesso uso di nodi specifici.

Tenete comunque conto che tali specificità possono o meno essere portate su altre piattaforme, a seconda delle caratteristiche hardware (ad esempio una semplice interfaccia GPIO è spesso facilmente portabile, mentre interfacce più specifiche lo sono di meno o possono non essere supportate da alcune piattaforme).

PERIFERICHE RASPBERRY PI

La dotazione di periferiche della Raspberry Pi è straordinariamente ricca e questo SBC dispone sia delle periferiche evolute tipiche delle piattaforme ad alta potenza computazionale (Ethernet, USB, HDMI, CSI, DSI ecc.), sia delle periferiche di base, più vicine al mondo dei microcontrollori (IO Digitale, PWM, periferiche di comunicazione inter-circuiti, eccetera).

In particolare, nella versione 4, con il passaggio dal BCM28xx al BCM2711, è stato notevolmente arricchito il set di funzioni alternative presenti sul connettore di GPIO.

In particolare, la versione 4 dispone delle seguenti periferiche sul connettore GPIO (ovviamente *multiplexate* sul totale dei 28 pin disponibili):

- Fino a 28x GPIO Pin
- Fino a 6x UART
- Fino a 6x I2C
- Fino a 5x SPI
- Fino a 2x PWM
- Fino a 3x GPCLK outputs
- 1x SDIO
- 1x DPI (Display Parallel Port)
- 1x PCM
- 1x 1-Wire

In Figura 3-1 è riportata la disposizione delle varie funzioni alternative sul connettore GPIO a 40-pin della Raspberry Pi 4 Model B. Come si può facilmente notare le possibilità offerte dalle interfacce presenti sul connettore di GPIO sono notevoli; è possibile, infatti, controllare I/O digitali, come pulsanti o relè, interfacciare una moltitudine di sensori e attuatori usando i vari bus di comunicazione disponibili, interfacciare motori o LED tramite i canali PWM e molto altro ancora. Nei prossimi paragrafi vedremo come gestire in Node-

RED due gruppi di interfacce, il GPIO standard (ossia IO Digitali e PWM) e alcune periferiche di comunicazione (UART, I²C, 1-wire).

GESTIONE GPIO RASPBERRY PI

La dotazione di base della Raspberry Pi prevede una serie di standard GPIO (28 in totale, corrispondenti al GPIO bank 0 del BCM2711), che possono essere utilizzati come *digital input* o *digital output*.

La dinamica del segnale di queste linee digitali è 0-3,3V e ogni linea è dotata di pull-up e pull-down selezionabili via software; quindi non è necessario avere resistenze di pull-up o pull-down esterne per interfacciare pulsanti ed interruttori. Inoltre alcune linee di GPIO possono essere utilizzate come canali PWM (Pulse-Width Modulation), in particolare il BCM2711 dispone di due canali PWM (PWM 0 e 1) multiplexati con i pin GPIO 12 e 13 (o in alternativa GPIO 18 e 19).

In ogni caso tutti i GPIO possono implementare il PWM software e vedremo che questa modalità è ampiamente supportata in Node-RED. Non ci sono canali analogici nativi all'interno del BCM2711 (come anche nei predecessori), di conseguenza per acquisire segnali analogici è necessario interfacciare un convertitore A/D esterno, tramite un apposito bus di comunicazione (tipicamente SPI o I²C).

In Node-RED, per l'interazione con i GPIO della Raspberry Pi esistono diverse soluzioni; in particolare riportiamo tre moduli che possono essere installati dalla palette di Node-RED e sono tra i più diffusi.

- **node-red-node-pi-gpio:** questo è un set di nodi che troviamo installato di default su Node-RED quando viene utilizzato lo script di installazione suggerito sul sito web di Node-RED. Se utilizziamo Raspberry Pi OS, il modulo è già preconfigurato per funzionare correttamente, quindi questa è la via più semplice per poter accedere al GPIO della Raspberry Pi usando Node-RED. Il modulo implementa tutte le funzionalità di base, ossia Digital Input (con gestione pull-up/down e debounce), Digital Output e PWM.
- **node-red-node-pi-gpiod:** questo modulo si basa sul demone PiGPIOD (<http://abyz.me.uk/rpi/pigpio/pigpiod.html>) per interfacciare il GPIO della Raspberry. Di conseguenza, per poter utilizzare i suoi nodi, è prima necessario installare correttamente il demone, seguendo le istruzioni riportate sulla sezione download della pagina web di PiGPIOD (<http://abyz.me.uk/rpi/pigpio/download.html>). Non esiste un vero e proprio installer, ma bisogna scaricare un pacchetto che poi deve essere compilato localmente. Inoltre il demone deve essere preventivamente avviato (tramite riga di comando o allo startup usando rc.local o systemd) affinché i nodi funzionino correttamente. Si tratta di un sistema più complesso rispetto ai nodi predefiniti, ma che offre migliori performance e funzionalità aggiuntive (ad esempio la possibilità di controllare servo tramite le linee di GPIO).
- **node-red-contrib-gpio:** questo modulo permette di accedere al GPIO su una serie di differenti piattaforme (incluse Raspberry PI, Beagle Bone, Intel Galileo, eccetera).

Nei paragrafi che seguono utilizzeremo i nodi di default (node-red-node-pi-gpio), ma teniamo presente che ci sono anche altre strade che consentono di migliorare le performance ed accedere a funzionalità aggiuntive.

DIGITAL I/O

Come abbiamo già detto il set di nodi node-red-node-pi-gpio è preinstallato nella versione di Node-RED per Raspberry Pi. Questo set è presente nella sezione Raspberry Pi della palette (Figura 3-2) e contiene i seguenti nodi:

- **rpi-gpio out:** permette di gestire i pin di GPIO in uscita; i pin possono essere gestiti in modalità digitale o PWM;

- **rpi-gpio in:** permette di gestire i pin di GPIO in ingresso; questo nodo permette anche di controllare i setting delle resistenze di pull-up e pull-down integrate, oltre che impostare il debounce.
- **rpi-mouse:** permette di gestire un mouse USB collegato alla Raspberry (solo gestione pulsanti);
- **rpi-keyboard:** permette di gestire una tastiera USB collegata alla Raspberry Pi.

Per i nostri scopi i nodi di maggiore interesse sono i primi due (GPIO out e in), mentre non ci occuperemo dei nodi mouse e keyboard. Abbiamo già avuto modo di utilizzare il nodo GPIO negli esempi precedenti, senza però entrare nel dettaglio della configurazione ed esaminare le funzionalità aggiuntive. Partiamo quindi dall'analisi di questo nodo, per poi passare al nodo GPIO in.

GPIO out

Per posizionare il nodo sul workspace preleviamolo dalla palette e posizioniamolo su un flow, come di consueto. Come si può notare già dall'icona, il nodo ha un singolo ingresso e nessuna uscita, questo ci fa capire che riceve in input un messaggio (contenente lo stato logico o il valore del PWM duty-cycle con il quale vogliamo controllare l'uscita selezionata) e non restituisce nulla in uscita, in quanto attua direttamente la linea fisica. Cliccando due volte con il tasto sinistro del mouse sul nodo si aprirà la schermata di configurazione, riportata in Figura 3-3.

Come si può vedere dalla predetta figura, la configurazione del noto GPIO out è estremamente intuitiva. Sostanzialmente abbiamo una schermata con un controllo principale (Pin) che riproduce la composizione del connettore di GPIO a 40 Pin della Raspberry Pi, dal quale possiamo selezionare il Pin che intendiamo utilizzare. Una volta eseguita questa operazione selezioniamo la modalità del Pin (Digital output o PWM), usando il drop-down menu Type. Infine abbiamo la possibilità di impostare il valore iniziale del Pin (selezionando la checkbox "Initialize pin state") e possiamo anche dare un nome più familiare al nodo (ad esempio LED_1). Nel caso in cui si selezioni PWM anziché Digital output come modalità, la schermata cambia leggermente e presenta un nuovo controllo di tipo textbox chiamato Frequency, che serve per impostare la frequenza del PWM in Hz, come si può vedere dalla Figura 3-4.

Il controllo di questo nodo è estremamente semplice, in sostanza è sufficiente inviare un messaggio il cui msg.payload contenga lo stato dell'uscita digitale (come boolean o come duty-cycle PWM). Per testare il funzionamento del nodo si possono costruire dei semplici nodi di test, come quelli rappresentati in Figura 3-5, che fanno uso del nodo Inject per iniettare i valori digitali e di duty-cycle.

GPIO in

Il nodo GPIO in funziona in maniera del tutto duale rispetto al nodo GPIO out. Prelevandolo dalla palette e posizionandolo sul workspace si nota subito che il nodo ha una singola uscita e nessun ingresso. In maniera similare al nodo GPIO out, l'input di questo nodo è dato dal cambio di stato sulla linea digitale configurata, e il suo output è un messaggio il cui payload contiene il valore digitale che rappresenta lo stato del pin (0 o 1). Come per il nodo GPIO out, anche questo nodo dispone di una finestra delle proprietà (Figura 3-6) che può essere aperta tramite doppio click sinistro sull'icona del nodo.

Anche in questo caso abbiamo il selettore Pin che ci permette di selezionare quale GPIO utilizzare dalla lista dei GPIO presenti sul pinout del connettore a 40 pin. I controlli addizionali Resistor e Debounce permettono di configurare la resistenza di pull-up/down e l'intervallo di debounce (in ms). Questi due controlli risultano molto utili nel caso in cui si debbano interfacciare dei pulsanti. Infine la checkbox "Read initial state of pin on deploy/restart?" permette di effettuare una prima lettura all'avvio del flow, così che lo stato del pin non risulti indeterminato all'inizio. Il solito controllo di tipo textbox "Name" permette di dare un nome più appropriato al nodo (ad esempio BUTTON_1).

Se vogliamo testare il funzionamento del nodo è sufficiente (dopo averlo configurato opportunamente) collegare un nodo di debug alla sua uscita, come rappresentato in Figura 3-7 e collegare fisicamente un pulsante alla linea digitale selezionata. Con una singola pressione del pulsante il nodo dovrebbe ritornare due messaggi, corrispondenti ai due stati tra cui transita la linea digitale (ad esempio 0 e 1, pulsante premuto e poi rilasciato, se il pulsante si chiude verso massa e viene utilizzata una resistenza di pull-up sulla linea).

Per verificare il funzionamento del GPIO di base, può essere molto pratico utilizzare l'HAT I/O distribuito da Futura Elettronica (www.futurashop.it) con il codice FT1060M. Questo HAT (sostanzialmente uno shield, una cui immagine è visibile in Figura 3-8) dispone inoltre di un convertitore AD/ DA (PCF8591) I²C, in modo da poter implementare anche alcune funzioni analogiche. Ecco le caratteristiche e la dotazione dell'HAT FT160M:

- 5 LED, collegati ad altrettanti digital output;
- 2 Pulsanti e 1 interruttore, collegati ad altrettanti digital input;
- 1 LED, collegato al canale D/A del PCF8591;
- 1 Sensore di temperatura, collegato ad un canale A/D del PCF8591;
- 1 Fotoresistenza, collegata ad un canale A/D dell'integrato di data-acquisition PCF8591 on board.

PERIFERICHE DI COMUNICAZIONE

Dopo aver esaminato in dettaglio i vari nodi per la gestione del GPIO, passiamo adesso all'analisi delle più comuni periferiche della Raspberry Pi che possiamo accedere tramite Node-RED. Come abbiamo visto nella parte introduttiva dell'articolo, la Raspberry Pi dispone di diverse periferiche di comunicazione, che permettono di interfacciare diversi tipi di dispositivi, come ad esempio sensori, attuatori o anche PC o altri SBC.

Per ovvie ragioni di spazio non è possibile effettuare una trattazione esaustiva, quindi ci limiteremo a trattare alcune delle periferiche più comuni e che ci saranno utili per lo sviluppo dei nostri progetti pratici anche nel prosieguo del corso.

Seriale

Cominciamo dalla periferica di comunicazione per eccellenza, ossia la porta seriale o porta UART (Universal Asynchronous Receiver and Transmitter). La UART è una porta di comunicazione asincrona, di tipo full-duplex (ossia è possibile ricevere e trasmettere allo stesso tempo, dal momento che è dotata di due linee di trasmissione, RX e TX), ed opera nativamente a livello TTL (ossia con dinamica 0-5V). La porta seriale viene spesso utilizzata con diversi standard di livello fisico, come RS-232, RS-422 e RS-485, che permettono di ampliare le distanze di comunicazione (ad esempio tramite l'adozione di tecniche differenziali nella trasmissione del segnale) o migliorare l'indirizzamento dei dispositivi. La Raspberry Pi è nativamente dotata di due UART: la mini UART (/dev/ttys0) e la PL011 UART (/dev/ttymA0). Per impostazione predefinita, la mini UART è mappata sui pin GPIO 14 e 15 (rispettivamente TXD ed RXD) mentre la PL011 è dedicata al modulo Bluetooth on-board. Ci serviremo quindi della mini UART come interfaccia seriale per i nostri test, ma prima di poterla utilizzare con Node-RED occorre disabilitare la console, che per impostazione predefinita è attiva su questa interfaccia di comunicazione e, di conseguenza, ci impedirebbe di utilizzare questo dispositivo da Node-RED. La console sulla mini UART può essere disabilitata utilizzando l'utility raspi-config (sudo raspi-config o, più semplicemente, editando il file cmdline.txt).

Per editare il file cmdline.txt, aprete una shell (ad esempio tramite connessione SSH) e digitate il comando:

```
sudo nano /boot/cmdline.txt
```

Una volta aperto il file con nano editor, rimuovete la seguente sezione:

```
console=serial0,115200
```

ed effettuate un reboot.

```
sudo reboot
```

A questo punto la mini UART è scollegata dal terminale e possiamo utilizzarla. Node-RED offre il modulo serial port per la gestione della seriale (<https://flows.nodered.org/node/node-red-node-serialport>), composto da tre nodi (Figura 3-9):

- Serial In: permette di ricevere dati dalla porta seriale.
- Serial Out: permette di inviare dati alla porta seriale.
- Serial Request: permette di inviare un dato e si mette in attesa della risposta (combinazione del nodo serial out – serial in).

A questi si aggiunge il nodo di configurazione serial-port. Il nodo di configurazione serial-port permette di configurare il comportamento della porta seriale, impostando valori come il baud rate, il numero di data bits e stop bits, la parità, ecc. Per testare il funzionamento della porta seriale dobbiamo dotarci di un convertitore USB-Seriale e collegarlo alla Raspberry Pi; Occorre collegare i pin 14 e 15 (con una connessione di tipo crossed), oltre ad un pin di GND.

Una volta fatto il collegamento possiamo posizionare il nodo Serial out sul workspace e cliccare due volte sul tasto sinistro per aprire la finestra delle proprietà (Figura 3-10).

A questo punto clicchiamo sull'icona a forma di matita a fianco del drop down menu "Serial Port".

Questo permetterà di aprire la schermata di Figura 3-11, dalla quale è possibile configurare la porta seriale. Selezioniamo come porta /dev/ttyS0 (è possibile cliccare sul pulsante a forma di lente di ingrandimento per ottenere una lista delle interfacce seriali disponibili), impostiamo il baud-rate a 115200 baud e nella sezione output indichiamo '\n' come carattere da aggiungere ad ogni stringa (in modo da avere un ritorno a capo automatico).

Infine clicchiamo su Update e Done per confermare la configurazione e l'associazione del nodo Serial out con il nodo di configurazione serial-port appena creato.

Ora la nostra porta seriale è configurata, per utilizzarla in trasmissione è sufficiente collegare un nodo inject con la stringa che vogliamo inviare sulla porta seriale. Nell'esempio di Figura 3-12 abbiamo inviato la stringa "Ciao Mondo!!!" sulla porta seriale, se apriamo una finestra di terminale su un PC collegato al convertitore USB-seriale che abbiamo usato per interfacciare la porta UART della Raspberry Pi dovremmo ricevere la stringa inviata.

I²C

La Raspberry PI è dotata anche di una periferica I²C, assegnata per impostazione predefinita ai GPIO 2 e 3 (SDA e SCL, rispettivamente). L'I²C è una interfaccia molto utile, infatti si tratta di un protocollo seriale sincrono half-duplex, di tipo master-slave multi-drop. Questo significa che, a differenza della seriale asincrona, può indirizzare diversi dispositivi, agendo proprio come un piccolo bus di comunicazione. Tramite questa interfaccia possiamo collegare alla nostra Raspberry Pi una moltitudine di sensori ed attuatori, controllandoli facilmente tramite Node-RED. Per utilizzare l'I²C la prima operazione da eseguire è quella di abilitare l'interfaccia. Questo può essere fatto semplicemente dall'utilità raspi-config (suro raspi-config), selezionando l'opzione "Interface Options" e successivamente l'opzione "I2C". Ci verrà chiesto se vogliamo abilitare la periferica, domanda alla quale dovremo rispondere affermativamente e ad operazione conclusa dovremo vedere un messaggio come quello di Figura 3-13 che conferma l'attivazione della periferica I²C (al termine dell'operazione è necessario eseguire un reboot).

Una volta abilitata la periferica dovremo installare un modulo per la gestione dell'interfaccia I²C-Bus sotto Node-RED. Uno dei moduli più versatili, in quanto del tutto generico è il modulo node-red-contrib-i2c (<https://flows.nodered.org/node/node-red-contrib-i2c>), che come di consueto potremo installare tramite l'opzione "Manage palette → Install". Questo modulo contiene tre nodi, come evidenziato in Figura 3-14, che sono:

- **I2c Scan**: permette di effettuare una scansione del bus I²C alla ricerca di dispositivi presenti, elencandoli per indirizzo in un array fornito come risposta.

- **I2c In:** permette di effettuare una richiesta di lettura sul bus I2C.
- **I2c Out:** permette di effettuare una richiesta di scrittura sul bus I²C.

Oltre a questo set di nodi generico sono presenti diversi altri nodi per la gestione di dispositivi specifici, che semplificano ulteriormente la gestione, sollevando l'utilizzatore anche dall'onere di inviare la sequenza di comandi necessari all'interrogazione di un determinato dispositivo. Tuttavia, data la quantità di dispositivi I²C presenti sul mercato, risulta molto utile avere a disposizione anche questo set generico.

Avremo modo di utilizzare il bus I²C in seguito nel corso delle altre puntate, per adesso è sufficiente sapere come attivarlo e quali sono i nodi per gestirlo.

1-Wire

L'ultima interfaccia che analizziamo in questa puntata è l'interfaccia 1-wire. Come il nome suggerisce, la caratteristica principale di questa semplice interfaccia è quella di consentire la gestione di dispositivi esterni tramite un singolo collegamento (più ovviamente gli estremi della tensione di alimentazione). Sebbene non esistano moltissimi dispositivi compatibili con questo standard, il basso costo, la semplicità di gestione e le esigue risorse necessarie per gestirlo (un singolo pin da sacrificare per la connessione), lo rendono spesso un'ottima scelta implementativa.

Come nel caso precedente, per impostazione predefinita la Raspberry Pi ha un'interfaccia 1-wire disponibile, allocata sul GPIO 4, che va attivata, esattamente come abbiamo visto fare prima per l'I²C. Lanciamo quindi raspi-config, e selezioniamo nuovamente "Interface Option", ma stavolta selezioniamo l'opzione "1-Wire" dal menu di gestione delle interfacce, confermando l'abilitazione. Anche in questo caso, al termine della procedura avremo un messaggio di conferma e sarà necessario eseguire un reboot del sistema.

Dal punto di vista di Node-RED, non esiste ad oggi un set di nodi generico per la gestione della 1-wire, ma piuttosto una serie di nodi specifici per i singoli dispositivi.

Ad esempio, uno dei dispositivi 1-wire più diffusi è il sensore di temperatura Dallas Semiconductor DS18B20. Questo dispositivo, incapsulato in un case plastico tipo TO-92 (a tre terminali) è facilmente reperibile, ha un bassissimo costo ed una elevata precisione di misura; inoltre se ne possono collegare diversi sullo stesso bus 1-wire, in quanto ogni dispositivo è identificato da uno specifico serial number univoco. Esistono svariati nodi per la gestione di questo dispositivo in Node-RED; uno molto efficace è quello fornito dal modulo node-red-contrib-sensor-ds18b20 (<https://flows.nodered.org/node/node-red-contrib-sensor-ds18b20>), illustrato in Figura 3-15.

Il nodo è di semplicissimo utilizzo, una volta installato e portato sul workspace, facendo doppio click possiamo accedere alla finestra delle proprietà (Figura 3-16). Qui verranno già elencati, ordinati per serial number, tutti i sensori DS18B20 che la scheda Raspberry Pi è riuscita a rilevare (ovviamente dovremo averne collegato almeno uno per vedere qualcosa).

A questo punto il nodo può essere configurato per eseguire letture periodiche ad un certo rate, oppure naturalmente esegue una lettura ogni volta che viene ricevuto un messaggio sulla porta di ingresso. Il risultato della lettura, già convertito in Celsius, verrà inserito nel campo payload del messaggio in output, come di consueto.

ESEMPIO PRATICO: LETTURA DI UN SENSORE DS18B20 TRAMITE INTERFACCIA SERIALE

Arrivati a questo punto, possiamo sfruttare le conoscenze acquisite per implementare una semplice applicazione: ciò che vogliamo realizzare stavolta è un dispositivo che acquisisca una lettura da un sensore di temperatura, nel nostro caso il popolare integrato DS18B20, e la invii ad un terminale seriale in esecuzione su Personal Computer ogni volta che quest'ultima viene richiesta.

Allo scopo utilizzeremo la stringa “Temp?” per identificare una richiesta valida, mentre qualsiasi altra stringa verrà ritenuta non valida.

Per l'implementazione ci occorrono due blocchi serial: serial in (per ricevere la richiesta) e serial out (per fornire la risposta). Inoltre utilizzeremo un nodo function e un nodo switch per verificare che la richiesta ricevuta sia effettivamente formattata correttamente e altri 2 nodi function per formattare le due risposte (positiva con lettura di temperatura e negativa). Infine ci occorre il nodo sensor-ds18b20 visto in precedenza. In Figura 3-17 è rappresentato il flow che implementa il nostro semplice programma:

analizziamo nodo per nodo il flow al fine di comprenderlo meglio. Il Nodo Serial IN riceve la stringa in ingresso dalla linea seriale. Il configuration node di questo blocco è del tutto simile a quello visto nel paragrafo dedicato alla porta seriale. Il nodo function GetRequest ha il compito di verificare se la richiesta sia stata correttamente formattata: ci aspettiamo di ricevere “Temp?\n”, in quanto il configuration node della porta seriale si aspetta “\n” come carattere di terminazione di una stringa. Il controllo della formattazione è fatto con il codice javascript illustrato qui di seguito:

```
if (msg.payload == "Temp?\n")
{
  msg.payload = "TempReq";
}
else
{
  msg.payload = "InvalidReq";
}
return msg;
```

che riguarda il blocco GetRequest.

Quindi il nodo valuta la stringa e ritorna msg.payload contenente “TempReq” se la richiesta è corretta o “InvalidReq” altrimenti. A questo punto ci serviamo di un nodo Switch per creare due path alternativi nei due casi. Il nodo switch è stato configurato con le regole visibili in Figura 3-18.

A questo punto il flow ha una biforcazione: la parte marcata con ‘1’ è il flusso di risposta positivo, quindi qui viene letta la temperatura dal sensore DS18B20 e il nodo function PosResp formatta la risposta tramite il codice javascript:

```
msg.payload = "Temperature: " + msg.payload;
return msg;
```

che riguarda il blocco PosResp.

Siccome non è necessaria alcuna periodicità nella lettura (viene fornita la temperatura solo su richiesta), l'opzione periodic read è stata disabilitata nel nodo sensor-ds18b20. In questa configurazione il nodo fornisce la temperatura solo quando viene triggerato da un messaggio in ingresso.

La parte marcata invece con ‘2’ è il flusso di risposta negativo, quindi non viene letto il sensore e il nodo function NegResp ritorna una stringa contenente un messaggio di errore, tramite il seguente codice javascript:

```
msg.payload = "ERROR: Invalid Command...";
return msg;
```

relativo al blocco NegResp.

I flussi si ricongiungono convergendo sul nodo Serial OUT, che invia la risposta sulla linea seriale (il configuration node referenziato è lo stesso del nodo Serial IN).

In Figura 3-19 è possibile vedere le schermate relative ad alcune letture effettuate tramite il terminale seriale.

Per la realizzazione dell'esempio abbiamo utilizzato la sonda DS18B20 2846-SONDADS18B20 e il convertitore seriale 2846-CAVOUSBSERIALE, entrambi distribuiti da Futura Elettronica. Nel collegare il convertitore USB-Seriale si tenga presente che è sufficiente collegare solo i pin TX, RX e GND, se la Raspberry Pi è già alimentata. La sonda DS18B20 va collegata al GPIO 4 ed alimentata tramite uno dei pin a 3,3V della Raspberry Pi. Inoltre sul GPIO 4 va attivata la resistenza di pull-up interna.

CONCLUSIONI

Bene, siamo così arrivati alla conclusione di questa terza puntata, nella quale abbiamo visto come controllare le periferiche della scheda Raspberry Pi tramite Node-RED. Siamo partiti dalla semplice gestione del singolo GPIO per arrivare fino alle periferiche di comunicazione, come UART, I²C e 1-wire, per le quali Node-RED offre diversi moduli di gestione.

Nella prossima puntata introdurremo un importantissimo elemento di Node-RED: si tratta della Dashboard, che permette di creare user interfacce (UI) accessibili via web per implementare i nostri progetti.



Figura 3-1 - Composizione del connettore di GPIO della Raspberry Pi 4.

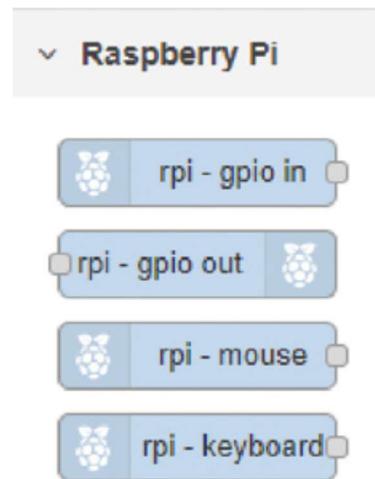


Figura 3-2 - Nodi node-red-node-pi-gpio

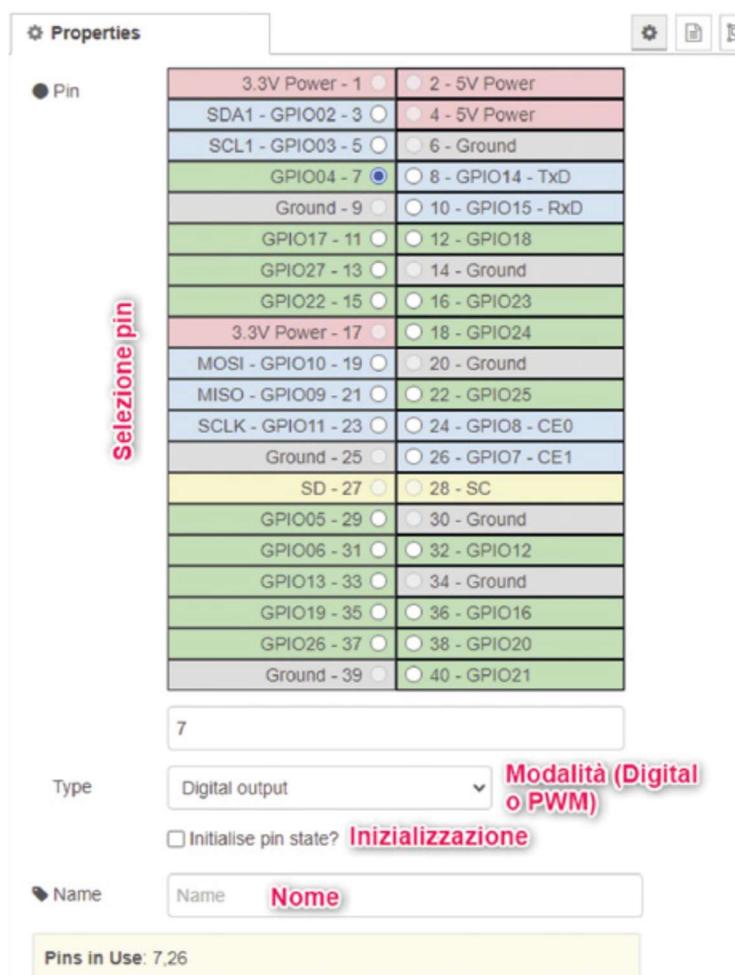


Figura 3-3 - Configurazione nodo GPIO out.



Figura 3-4 - Configurazione PWM.



Figura 3-5 - Test digital output e PWM.

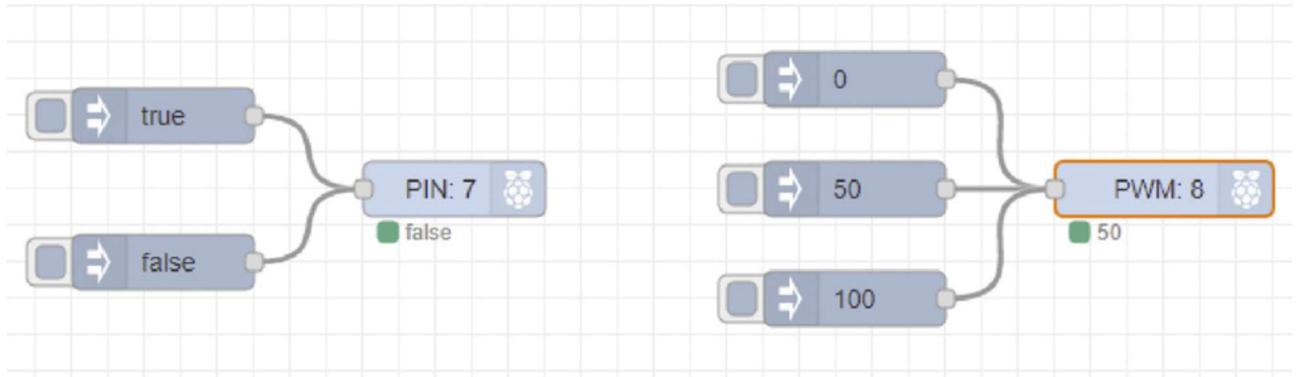


Figura 3-6 - Configurazione del nodo GPIO in.

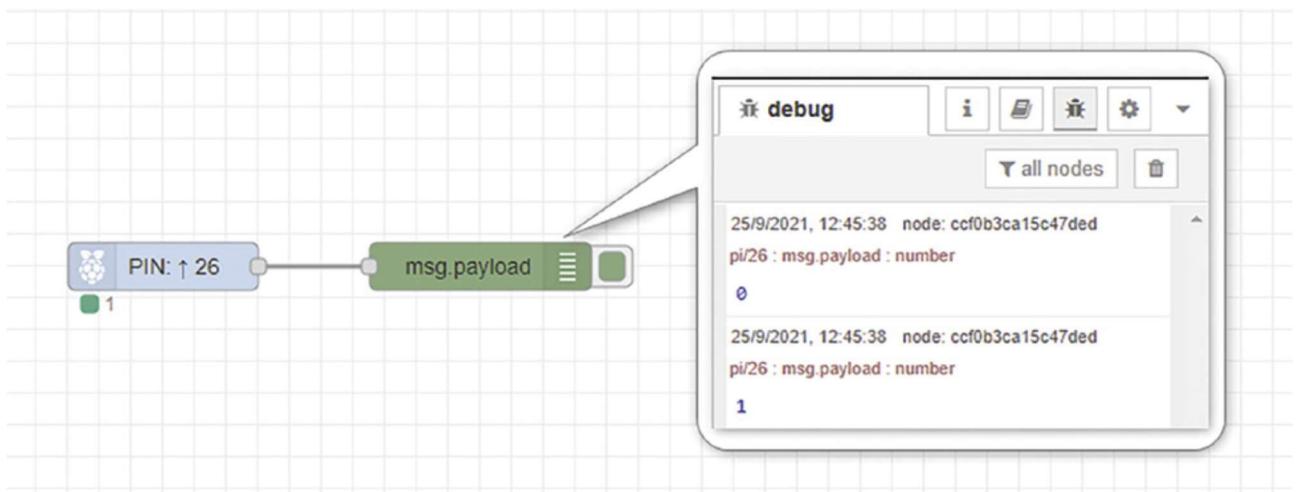
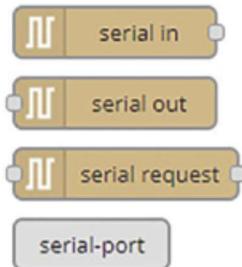
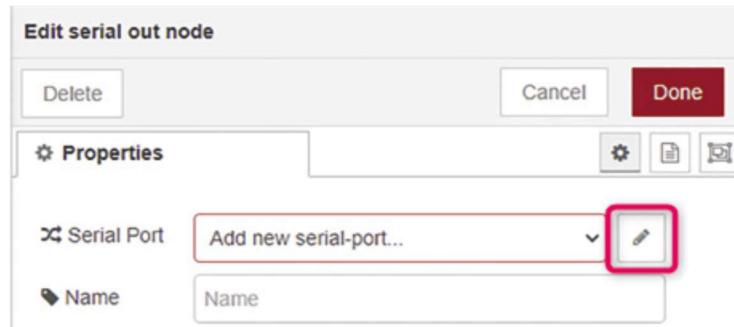
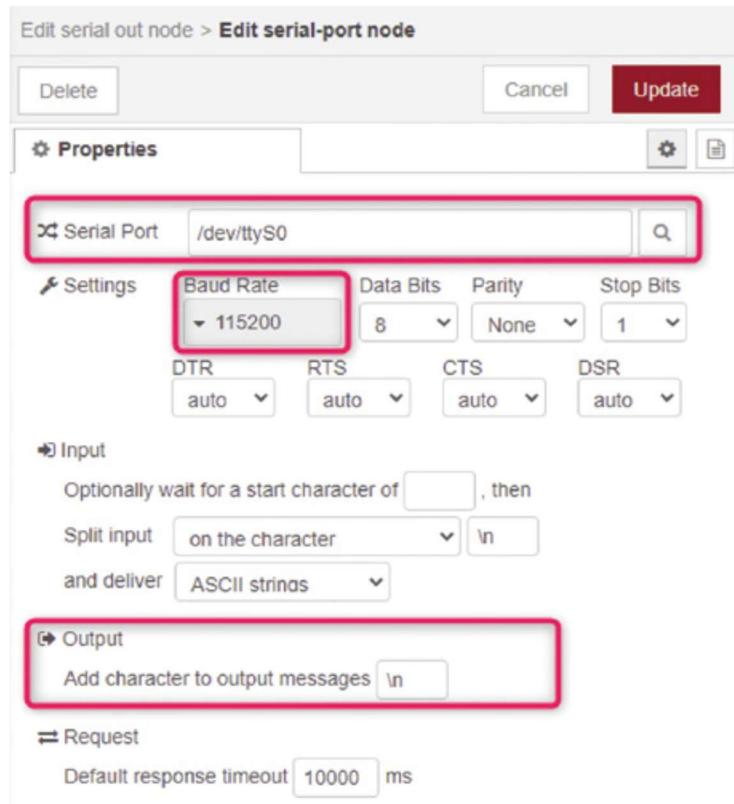


Figura 3-7 - Test del nodo GPIO in con visualizzazione della finestra di debug.



Figura 3-8 - Raspberry Pi I/O HAT FT1060M.

*Figura 3-9 - Nodi del modulo serial port.**Figura 3-10 - Finestra delle proprietà del nodo Serial Out.**Figura 3-11 - Configurazione del nodo serial-port.*

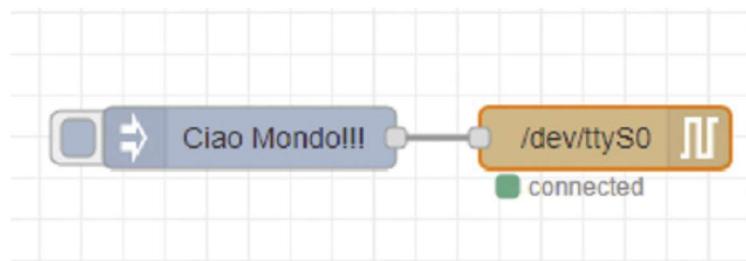


Figura 3-12 - Test Seriale.

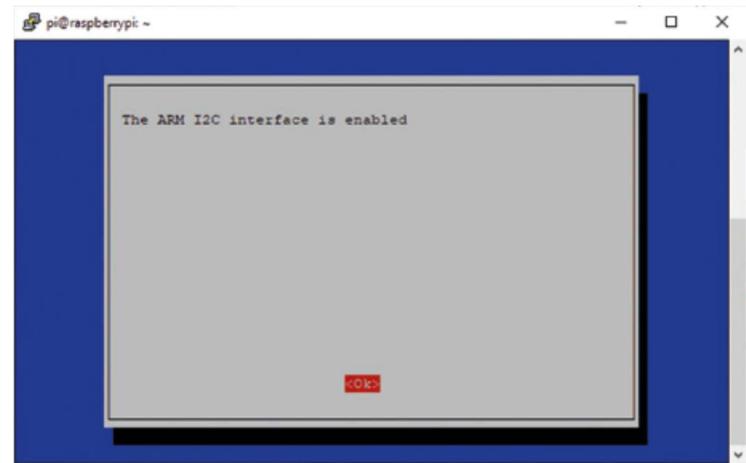


Figura 3-13 - Attivazione della periferica I2C su Raspberry Pi.



Figura 3-14 - Nodi per la gestione dell'interfaccia I2C tramite Node-RED.

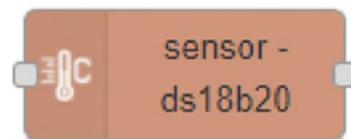


Figura 3-15 - Nodo per la gestione dei sensori di temperatura DS18B20.

A screenshot of the "Properties" window for the "sensor-ds18b20" node. The window includes fields for "Sensor ID" (set to "28-00000852f501"), "Periodic" (unchecked), "Period (secs)" (set to "1"), and "Name/topic" (set to "Name"). There are also standard window controls for "Properties", "Settings", and "Close".

Figura 3-16 - Finestra delle proprietà del nodo sensor-ds18b20.

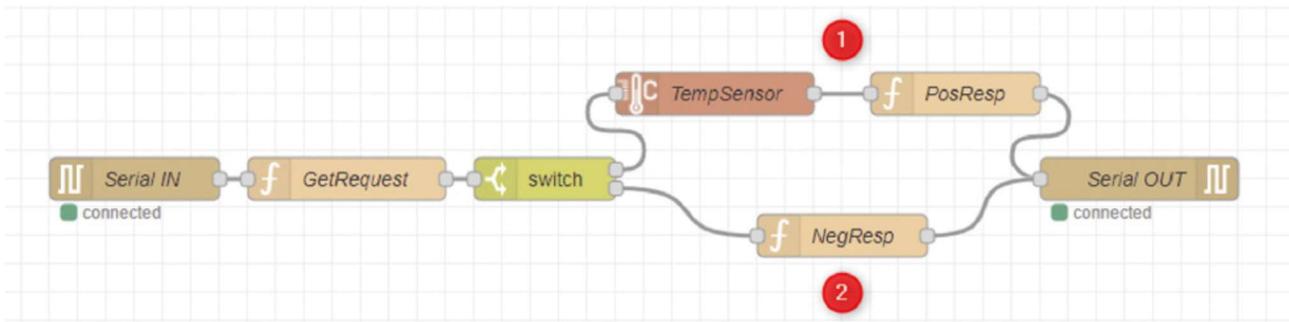


Figura 3-17 - Flow dell'esempio pratico.

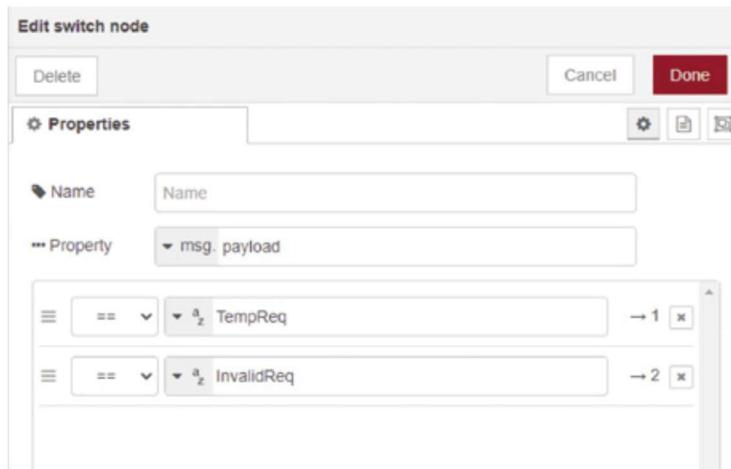


Figura 3-18 - Regole del nodo switch.

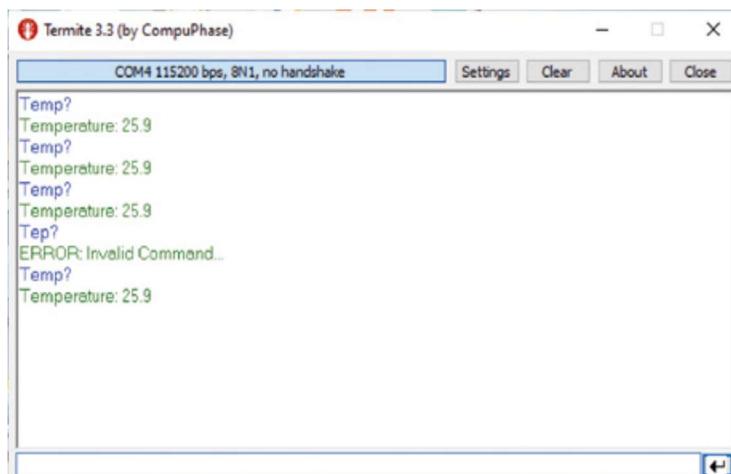


Figura 3-19 - Letture di temperatura tramite terminale seriale.

4. Quarta Puntata.

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 della parte grafica delle nostre applicazioni, sfruttando la Node-RED dashboard, uno dei pacchetti di nodi aggiuntivi maggiormente utilizzati.

Nella precedente puntata abbiamo iniziato a prendere confidenza con la gestione delle periferiche hardware tramite Node-RED, utilizzando, come di consueto per questo corso, la Raspberry Pi come piattaforma hardware. Nello sviluppo di applicazioni IoT, o comunque orientate alla connettività, l'utilizzo delle periferiche è sicuramente una faccia della medaglia, in quanto è fondamentale per l'interfacciamento dello specifico sensore o attuatore che si desidera rendere "smart" o comunque remotizzare, in modo che sia gestibile attraverso una opportuna interfaccia. La realizzazione della suddetta interfaccia è l'altro lato della medaglia e può essere declinata in diverse forme. Sicuramente una delle forme più comunemente utilizzate è quella grafica, nella modalità in cui l'istanza di Node-RED sulla nostra Raspberry Pi agisce come webserver, fornendo l'interfaccia grafica quando ci colleghiamo tramite un browser ad uno specifico indirizzo IP.

Node-RED ci viene incontro in questa specifica modalità di utilizzo con un componente estremamente interessante: la dashboard.

Tuttavia, prima di vedere di cosa si tratta nel dettaglio, dobbiamo affrontare il tema della configurazione, in particolare perché, come vedremo in seguito, alcune modalità di utilizzo dei nostri dispositivi ci espongono a seri rischi di Cybersecurity.

CONFIGURARE NODE-RED

Come abbiamo detto nella parte introduttiva, in questa puntata porremo molto l'accento sulla creazione di interfacce utente remote, tramite Node-RED, utilizzando un componente (la dashboard) che è in grado di esporre parti grafiche interattive su una specifica pagina web, creando di fatto delle webapp. Questa modalità presenta notevoli vantaggi, tra cui la semplicità di utilizzo 117 e l'estrema compatibilità. Infatti per gestire uno specifico dispositivo remoto sarà sufficiente collegarci all'indirizzo IP in cui Node-RED espone il webserver e non è importante con quale dispositivo (la webapp è compatibile con i vari smartphone in circolazione, sia Android che iOS, che anche con i vari browser utilizzati in ambiente desktop). Tuttavia se vogliamo remotizzare un'applicazione di questo tipo, uscendo dai confini della nostra rete locale, non abbiamo altra scelta se non quella di aprire una porta del nostro router. Questa azione, inevitabilmente, ci sottopone ad un rischio: se non adeguatamente protetta, la nostra interfaccia web, come anche l'editor di Node-RED, potrebbero essere sfruttati da una terza parte a scopi malevoli. Dobbiamo quindi affrontare un problema di cybersecurity.

Fortunatamente Node-RED dispone di tutta una serie di predisposizioni che, se opportunamente attivate, ci consentiranno di esporre i nostri pannelli di controllo al di fuori della nostra rete locale con la ragionevole sicurezza di poter resistere ad attacchi esterni (in cybersecurity non si parla mai di sicurezza assoluta, in quanto avendo un determinato quantitativo di tempo a disposizione, anche le interfacce più sicure possono essere forzate). Prima di addentrarci però nei dettagli relativi alla protezione della nostra istanza di Node-RED è opportuno analizzare come Node-RED affronta il problema della configurazione, che consente anche di attivare le varie features di Cybersecurity integrate.

Node-RED, al suo avvio, carica una serie di opzioni di configurazione presenti sul file settings.js, che si trova di default nella sua directory principale. Di conseguenza, per configurare il comportamento della nostra istanza di Node-RED possiamo semplicemente intervenire su questo file, attivando o modificando le varie opzioni di configurazione che

sono presenti. Le opzioni disponibili sono diverse ed impattano su innumerevoli aspetti relativi all'editor, alla gestione dei files ed al comportamento in generale di Node-RED.

La maggior parte delle opzioni sono commentate e corredate di una breve descrizione, per attivarle è quindi sufficiente de-commentare la sezione specifica e fare le modifiche del caso. Ad esempio, se volessimo visualizzare l'editor all'interno di uno specifico path, come "/editor", basterebbe attivare l'opzione `httpAdminRoot`, e modificarla in:

```
httpAdminRoot: '/editor',
```

Riavviando Node-RED (può essere consigliabile riavviare del tutto la Raspberry Pi, tramite il comando `sudo reboot`), non visualizzeremo più il nostro editor su `<localhost>:1880`, ma su `<localhost>:1880\editor`.

Il dettaglio delle varie opzioni di configurazione esula dagli obiettivi di questo corso, ma una descrizione dettagliata delle varie configurazioni possibili può essere reperita dal sito web di Node-RED al seguente link:

```
https://nodered.org/docs/user-guide/runtime/configuration.
```

OPZIONI DI SECURITY

Ora che abbiamo capito come fare per configurare opportunamente Node-RED, entriamo nel dettaglio delle configurazioni di Cybersecurity che vogliamo mettere in atto per proteggere la nostra istanza di Node-RED nel momento in cui volessimo renderla accessibile dall'esterno della nostra rete locale.

La prima cosa che vogliamo fare è proteggere l'editor, per fare in modo che solo gli utenti autorizzati possano accedervi e fare modifiche ai flows. Per garantire questo tipo di protezione di accesso, Node-RED fornisce due sistemi:

- Autenticazione basata su username/password
- Autenticazione basata su OAuth/OpenID providers (come ad esempio Twitter o GitHub)

Noi ci concentreremo solo sulla prima modalità in questa puntata. Per creare utenti con relative credenziali di accesso dobbiamo de-commentare e modificare la proprietà `adminAuth` nel file `settings.js` dell'istanza di Node-RED che vogliamo proteggere. La proprietà `adminAuth` dispone di due campi:

- **Type**: rappresenta il tipo di protezione che vogliamo implementare. La modalità username/password è definita "credentials"
- **Users**: è un array di javascript objects che rappresenta i vari utenti che vogliamo creare, con relative password e diritti di accesso.

Quindi, se vogliamo creare due utenti, un utente `admin`, con diritti di lettura e scrittura e un utente `guest` con diritti di sola lettura, dovremo modificare la proprietà come illustrato nel Listato 1.

Listato 1 – Creazione degli utenti admin e guest

```
adminAuth: {
  type: "credentials",
  users: [
    {
      username: "admin",
      password: "$2b$10$uf4IHPnuO1BO4pR...", permissions: "*"
    },
    {
      username: "guest",
      password: "$2b$10$D4vDDId7yb42kme1...",
      permissions: "read"
    }
  ]
},
```

Come si può vedere dal listato le password sono opportunamente crittografate tramite l'algoritmo di hashing bcrypt, in modo da non essere conservate in chiaro (nell'esempio gli hash delle due password sono stati ridotti per ragioni di spazio). Per generare gli hash delle password dei nostri utenti possiamo utilizzare direttamente la funzione di hash bcrypt

integrata in Node-RED (`node-red admin hash-pw`), oppure utilizzare uno dei tanti servizi presenti online, come ad esempio <https://passwordhashing.com/BCrypt>.

Una volta creati gli utenti ed assegnati gli hash delle password non ci resta che assegnare i privilegi di accesso ad ogni utente. Nell'esempio di Listato 1, all'utente admin sono stati assegnati privilegi illimitati (*), mentre all'utente guest sono stati assegnati privilegi di sola lettura (read). Una volta fatta questa modifica e riavviato Node-RED, se proviamo ad accedere all'editor, verremo re-indirizzati su una pagina di login come quella riportata in **Figura 4-2**, e solo inserendo una coppia di credenziali valide potremo effettivamente accedere.

Il meccanismo di autenticazione di Node-RED prevede la creazione di token di accesso, che hanno validità per un certo periodo. Di conseguenza non saremo costretti a re-inserire la password di accesso finché il token rimane valido, e il periodo di validità dei token può essere modificato agendo sulla proprietà `sessionExpiryTime` (il valore di default è di 7 giorni). Si faccia attenzione però a questo settaggio: periodi troppo lunghi potrebbero esporre a rischi legati alla sicurezza, mentre periodi troppo corti comportano la necessità di dover reinserire la password di accesso ogni volta che l'editor lo richiede (anche a runtime, mentre stiamo modificando i nostri flows).

Il meccanismo di protezione che abbiamo creato consente di bloccare l'accesso al nostro editor da parte di soggetti non autorizzati, ma è ancora vulnerabile, perché lo scambio di informazioni avviene su un canale HTTP non protetto. Di conseguenza le nostre credenziali di accesso sono trasmesse in chiaro dal client (il browser che stiamo utilizzando) verso il nostro server Node-RED, e di conseguenza potrebbero essere intercettate. Per eliminare questo rischio dobbiamo passare su un canale sicuro di tipo HTTPS. Per farlo è sufficiente de-commentare la proprietà `https` presente sempre sul nostro `settings.js`. Questa proprietà contiene due campi, `key` e `cert`, che rappresentano la chiave privata e la catena di certificati necessari per la creazione del canale HTTPS. I campi `key` e `cert` non sono altro che i path della chiave privata e del certificato, in formato `.pem`. Quindi, supponendo di avere una coppia chiave/certificato valida, per re-indirizzare la connessione al nostro server Node-RED su HTTPS dobbiamo inserire una configurazione simile a quella riportata nel Listato 2.

Listato 2 – Configurazione del canale HTTPS

```
https: {  
  key: require("fs").readFileSync('/home/pi/.node-red/privatekey.pem'),  
  cert: require("fs").readFileSync('/home/pi/.node-red/certificate.pem')  
},
```

Per la creazione della chiave privata e del certificato, è possibile seguire questa guida:

<http://stevesnoderedguide.com/securing-node-red-ssl>.

A questo punto, per raggiungere il nostro editor, non potremo più utilizzare una connessione HTTP, ma dovremo collegarci in HTTPS. Può essere comodo attivare anche la seguente opzione, sempre sul nostro file `settings.js`:

```
requireHttps: true,
```

che reindirizza le connessioni HTTP su HTTPS automaticamente.

A questo punto il nostro editor è ben protetto, e possiamo quindi decidere di esporlo verso l'esterno. Si faccia comunque sempre molta attenzione a questa operazione, dal momento che aprendo una porta del nostro router verso l'esterno ci esponiamo comunque ad attacchi, quindi si raccomanda sempre di evitare di inserire informazioni sensibili nei dispositivi a cui viene dato accesso e di proteggere sempre le varie interfacce che si espongono (come ad esempio SSH o simili).

NODE-RED DASHBOARD

Ora che la nostra istanza di Node-RED è protetta possiamo passare all'argomento principale di questa puntata, ossia, come accennato nella parte introduttiva, come realizzare interfacce grafiche per i nostri flows.

Dal momento che Node-RED nasce come un ambiente di sviluppo con forti connotazioni IoT, è stato sin dall'inizio pensato un componente che permetta agli utilizzatori di realizzare semplicemente interfacce grafiche: la dashboard.

La dashboard non è altro che un set di nodi (si installa esattamente come gli altri nodi e, una volta installato, compare in una sezione dedicata sulla palette) che permettono di inserire elementi grafici come pulsanti, gauges, grafici e molti altri, che ci permettono di visualizzare dati in real-time e di interagire con altri nodi presenti nei nostri flows.

Per installare la dashboard è sufficiente cercare tramite l'opzione "Manage Palette" il componente node-red-dashboard. Una volta installato vedremo nella palette il gruppo di nodi riportato in Figura 4-3.

Oltre ai vari nodi viene installato anche un particolare pannello di gestione, denominato dashboard ed accessibile dalla sidebar (vedi Figura 4-4), che ci permette di gestire layout, temi, font ed altre opzioni della parte grafica.

La dashboard è semplicissima da utilizzare, infatti, una volta installata, l'interfaccia grafica della nostra istanza di Node-RED risulta accessibile all'indirizzo:

<Node-RED server URL/IP>:1880\ui

Ossia dobbiamo semplicemente aggiungere "\ui" alla locazione a cui solitamente ci colleghiamo per accedere all'editor. Questo specifico path può anche essere modificato dall'utente, andando semplicemente a modificare la properties "ui" del file *setting.js*, in maniera del tutto simile a come abbiamo fatto per il path dell'editor nella parte iniziale di questa puntata (ricordiamoci sempre di riavviare Node-RED per rendere effettive le modifiche che facciamo al file di configurazione).

DASHBOARD NODES

Come abbiamo accennato in precedenza, la dashboard è composta da una serie di nodi, ed alla quasi totalità di questi nodi corrisponde un elemento grafico, denominato widget, con il quale sia il flow che l'utente possono interagire. E' possibile quindi visualizzare dati provenienti dal flow (ad esempio visualizzare la lettura di un sensore di temperatura su un grafico) oppure recepire input dall'utente ed eseguire azioni (ad esempio, attivare un relè alla pressione di un pulsante).

Facciamo quindi un'analisi di dettaglio dei vari widget che compongono la dashboard:

- **Audio Out:** questo widget permette di riprodurre audio (waw o mp3) o di sintetizzare vocalmente un testo passato come input.
- **Button:** questo widget crea un pulsante virtuale con il quale è possibile interagire. E' possibile impostare lo specifico *msg.payload* da inviare alla pressione del pulsante.
- **Chart:** questo widget crea un grafico, che può essere di vari tipi differenti (grafico a linee, a barre o a torta). Nel grafico vengono rappresentati i dati inviati alla porta di input del nodo, come ad esempio la lettura di un sensore.
- **Colour Picker:** questo widget crea un picker che permette di selezionare un colore.
- **Date Picker:** questo widget crea un picker che permette di selezionare una data.
- **Dropdown:** questo widget crea un dropdown menu e permette di selezionare una delle opzioni. Gli elementi del dropdown possono essere definiti dalla finestra di configurazione del nodo o anche tramite uno specifico campo (*msg.options*) inviato tramite un messaggio in ingresso.
- **Form:** questo widget crea un form i cui campi possono essere definiti tramite la finestra di configurazione del nodo.
- **Gauge:** questo widget crea un gauge che rappresenta il valore istantaneo passato alla porta di ingresso tramite *msg.payload*. Il nodo ha 4 modalità differenti per la gauge: standard, donut, compass e level.
- **Notification:** questo widget è in grado di creare notifiche per l'utente. La notifica può essere di tipo toast (popup che scompare dopo un tempo predefinito) oppure

nella forma popup con pulsante di conferma (ed in questo caso può anche essere richiesto un input all'utente).

- **Numeric:** questo widget crea un input di tipo numerico.
- **Slider:** questo widget crea una slider orizzontale controllabile dall'utente.
- **Switch:** questo widget crea uno switch, ossia un pulsante che ha due posizioni stabili, a cui corrispondono due differenti valori di `msg.payload` inviati come output.
- **Text:** questo widget crea un elemento grafico testuale di output, e può quindi essere utilizzato come label. Il contenuto della label viene impostato tramite il valore di `msg.payload` inviato sulla porta in ingresso.
- **Text Input:** questo widget crea un elemento grafico testuale di input. Può supportare diverse modalità, come password, email, telephone e vari tipi di picker.
- **Template:** la dashboard ha anche una sua specifica tipologia di template node, che permette di creare widget personalizzati, tramite HTML e Javascript.
- **Ui control:** questo nodo permette di controllare dinamicamente la dashboard.

Una volta posizionato un nodo della dashboard sul nostro flow, dovremo occuparci di come il relativo widget viene posizionato nel layout visualizzato sulla UI. La dashboard è organizzata in elementi gerarchici che prendono il nome di Tabs e Groups. I Tabs sono gli elementi posizionati gerarchicamente più in alto e rappresentano appunto i vari tab che possono essere visualizzati dalla UI (ne può essere visualizzato uno per volta). I groups invece sono contenuti nei tab e contengono al loro interno una serie di widgets. Di conseguenza, se vogliamo posizionare dei widget sulla nostra UI, dobbiamo creare almeno un tab ed un group. Questa operazione può essere eseguita sia dalla pagina di gestione della dashboard, che anche dalle singole pagine di configurazione dei vari nodi, nella sezione che permette di associare un nodo/widget ad un group.

Nella pagina di gestione della dashboard è possibile anche gestire il layout dei gruppi all'interno dei tab e dei widget all'interno dei gruppi.

ESEMPIO

Vediamo a questo punto un piccolo esempio di come funziona la dashboard, in maniera da comprenderne meglio caratteristiche e funzionalità. Ipotizziamo di voler creare un semplice pannello con un controllo di tipo slider ed uno di tipo gauge, posizionati all'interno di un gruppo su un tab, e di voler controllare il valore visualizzato dal gauge tramite lo slider. Per prima cosa creiamo gruppo e tab, andando a selezionare la sezione dashboard della sidebar e posizionandoci sul tab Layout. Per aggiungere un tab clicchiamo sul pulsante “+ tab”, come illustrato in Figura 4-5.

Una volta aggiunto il tab quest'ultimo comparirà nella sezione Tabs & Links; passandoci sopra con il mouse compariranno alla destra dell'elemento stesso tre opzioni: “+ group”, “edit” e “link”. Selezioniamo l'opzione edit per cambiare il nome al tab. Una volta selezionata l'opzione si aprirà una finestra delle proprietà dalla quale è possibile eseguire alcune operazioni, come rinominare il tab, aggiungere un'icona e altro. Cambiamo il nome da quello di default in MyTab e clicchiamo sul pulsante rosso Update per proseguire, come illustrato in Figura 4-6.

A questo punto clicchiamo su “+ group” per aggiungere un nuovo gruppo.

Similmente a quanto visto nel caso del tab, anche i gruppi hanno una opzione edit, che possiamo usare per rinominare il gruppo, cambiandone il nome in “MyGroup”.

Anche in questo caso abbiamo varie opzioni che per il momento possiamo ignorare. A questo punto la nostra UI dispone di un tab e di un gruppo, quindi possiamo inserire i widget. Selezioniamo, tra i nodi della sezione dashboard, un nodo slider ed un nodo gauge e trasciniamoli sul workspace, come illustrato in Figura 4-7.

Come si può vedere i nodi presentano le icone a forma di triangoli rossi, ad evidenziare un errore di configurazione (oltre al pallino blu ad indicare che ancora non è stato eseguito il deploy). Questo perché i nodi non sono ancora stati esplicitamente assegnati a nessun

gruppo. Questa assegnazione viene fatta dalla finestra di proprietà del nodo. Iniziamo con il nodo slider, cliccando 2 volte per accedere alla finestra properties. Da questa finestra assegniamo il gruppo tramite il controllo Group (primo dall'alto) e modifichiamo anche i seguenti 2 campi (come illustrato anche in Figura 4-8):

- *Name*: assegniamo come nome al controllo “MySlider”.
- *Tooltip*: aggiungiamo il seguente tooltip, “The slider will control the gauge”.

Lasciamo i valori di default per il resto dei controlli e clicchiamo sul pulsante Done per confermare.

Continuiamo modificando le proprietà del nodo gauge, assegnando il medesimo gruppo ed assegnando “MyGauge” come nome al nodo, come illustrato in Figura 4-9.

A questo punto, dato che vogliamo che lo slider controlli il gauge, colleghiamo con un wire l’output del nodo slider all’input del nodo gauge, ed eseguiamo infine il deploy.

Per visualizzare l’interfaccia grafica creata, possiamo collegarci al path interno in cui è visualizzata la ui, ossia, come detto in precedenza:

```
<Node-RED server URL/IP>:1880\ui
```

Dove <Node-RED server URL/IP> rappresenta l’indirizzo della nostra raspberry. In Figura 4-10 è riportato un esempio di output di questo semplice flow con interfaccia grafica.

SECURITY DELLA UI

Ora che abbiamo visto come creare una interfaccia grafica con Node-RED è anche opportuno ritornare nuovamente sul tema della security, in quanto l’opzione di protezione vista all’inizio della puntata è valida solo per l’editor, ma non per la UI. Node-RED ha un secondo set di opzioni per proteggere il pannello della dashboard ed i contenuti statici (come le immagini), se ce ne sono.

Le due opzioni, che potete trovare sempre sul file *settings.js*, sono riportate nel Listato 3.

Listato 3 – Opzioni di protezione della dashboard e dei contenuti statici

```
httpNodeAuth: {user: "admin", pass: "$2b$10$uf4IHPnuO1BO4pR...", ""},  
httpStaticAuth: {user: "admin", pass: "$2b$10$uf4IHPnuO1BO4pR...", ""},
```

Anche in questo caso si tratta di una coppia username-password, e la password è da inserire esattamente nelle stesse modalità della password per l’accesso all’editor, ossia crittografata con algoritmo bcrypt.

ESEMPIO PRATICO: STAZIONE METEO CON BME280

Passiamo a questo punto all’esempio pratico di questa puntata: quello che ci proponiamo di ottenere è di realizzare una semplice stazione meteo che prenda dati di un sensore di temperatura, umidità e pressione e li visualizzi su una serie di gauge su una pagina web. Come hardware utilizzeremo ovviamente una Raspberry Pi ed il sensore BME280. Tale sensore può essere acquistato presso lo shop di futura elettronica ed ha come codice prodotto la sigla 1606-GY-BME280 (una immagine del sensore è riportata in Figura 4-11).

La realizzazione è semplicissima: partiamo installando il nodo specifico che ci permette di comunicare con il sensore. Come di consueto usiamo la pagina di ricerca (manage palette, accessibile dal menu con il simbolo delle tre linee orizzontali in alto a destra), cerchiamo il nodo *node-red-contrib-bme280* ed installiamolo.

Una volta installato il nodo, posizioniamolo sul workspace ed apriamo la sua pagina delle proprietà. Dobbiamo:

- Rinominarlo in BME Sensor
- Assegnare come bus 1
- Assegnare come I2C Address 0x76
- Assegnare come topic bme280
- Spuntare l’opzione extra data (ci permetterà di leggere anche la temperatura apparente) come illustrato in Figura 4-12.

Il nodo ha un funzionamento semplicissimo, ogni volta che riceve un messaggio in ingresso, ritorna un object con 4 campi:

- Temperatura (temperature_C)
- Umidità (humidity)
- Pressione barometrica (pressure_hPa)
- Temperatura apparente (heatIndex)

Aggiungiamo adesso gli altri nodi necessari.

Ci occorre:

- un nodo inject per triggerare la lettura periodicamente (decidiamo di leggere 1 volta al secondo)
- 4 nodi change (ci serviranno per estrarre i campi dal messaggio in output dal nodo BME280)
- 3 nodi function (ci serviranno per formattare l'output di umidità, pressione e temperatura apparente, che hanno molte cifre decimali)
- 4 nodi Gauges (per la visualizzazione)

Creiamo anche l'infrastruttura della dashboard, composta da un Tab e da un Group, ed associamo i 4 gauges allo stesso gruppo, ed eseguiamo le seguenti configurazioni.

- Gauge 1:
 - -Nome: Temperature
 - -Range: -20 – 80
 - -Units: °C
- Gauge 2:
 - -Nome: Humidity
 - -Range: 0 – 100
 - -Units: %
- Gauge 3:
 - -Nome: Pressure
 - -Range: 500 – 1500
 - -Units: hPa
- Gauge 4:
 - -Nome: Heat Index
 - -Range: -20 – 80
 - -Units: °C

In Figura 4-13 è rappresentata la configurazione del gauge che fornisce l'indicazione relativa alla temperatura.

A questo punto configuriamo i nodi change. Lo scopo di questi nodi è quello di estrarre un campo specifico dal messaggio contenente l'object complessivo e passarlo al nodo function che lo formerà correttamente per la visualizzazione (o direttamente al nodo gauge nel caso del dato di temperatura).

Per farlo ci serviamo della regola set e impostiamo nel campo "to" il valore `msg.payload.<campo specifico>`. Dove il valore <campo specifico> individua il campo che vogliamo estrarre dal messaggio. In Figura 4-14 è rappresentata la configurazione del nodo change che estrae il dato di temperatura.

A questo punto non ci rimane che configurare i 3 nodi function. Questi nodi hanno la funzione di formattare correttamente i dati di alcuni campi (umidità, pressione, heat index), che hanno una precisione eccessiva (svariate cifre decimali) per la tipologia di dato da visualizzare. Tramite i nodi function, servendoci del metodo `toFixed()` portiamo la precisione a 2 cifre decimali. Lo snippet di codice Javascript che esegue questa operazione è riportato in Figura 4-15 ed è lo stesso per tutti i 3 i dati da manipolare (la temperatura viene già riportata dal nodo BME280 con una precisione di 2 cifre decimali). Ora non ci resta che configurare anche il nodo inject per fornire la periodicità. Come abbiamo visto

anche nelle puntate precedenti, è sufficiente settare l'opzione interval ed indicare 1 secondo come periodicità (Figura 4-16). A questo punto la configurazione è conclusa, e possiamo collegare i nodi del nostro flow come riportato in Figura 4-17. Ora possiamo eseguire il deploy del nostro flow ed effettuare i collegamenti tra la nostra Raspberry Pi ed il sensore. La breakout BME280 ha 4 pin: Alimentazione, GND, SDA ed SCL. Colleghiamo il pin di alimentazione ad una delle uscite a 5V della Raspberry Pi e il GND ad uno dei GND. I pin SDA ed SCL invece vanno collegati ai rispettivi pin della porta I2C1 della Raspberry Pi (pin 3 e 5 del connettore GPIO). A questo punto, se tutto è stato eseguito correttamente, collegandoci alla pagina:

```
<Node-RED server URL/IP>:1880\ui
```

Dove <Node-RED server URL/IP> rappresenta l'indirizzo della nostra raspberry, dovremmo visualizzare la dashboard rappresentata in Figura 4-18.

CONCLUSIONI

Siamo arrivati alla conclusione di questa quarta puntata in cui abbiamo visto come gestire la security e il componente dashboard in Node-RED. Le opzioni di security ci permettono di proteggere adeguatamente il nostro editor e le nostre interfacce grafiche qualora decidessimo di esporle sulla rete, mentre la dashboard ci permette di creare in maniera estremamente semplice interfacce grafiche hostate dal server, che quindi possono essere raggiungibili tramite diversi dispositivi, come PC, tablet e smartphone, ma è anche possibile visualizzarle su un dispositivo di output locale, come un monitor o un display, e creare quindi interfacce uomo-macchina molto evolute.



Figura 4-1 - Cybersecurity.



Figura 4-2 - Protezione dell'accesso all'editor di Node-RED con username e password.

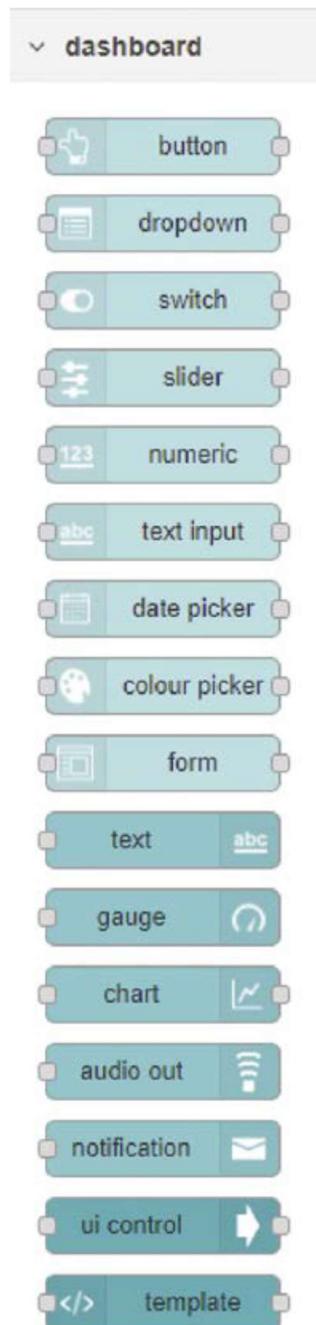


Figura 4-3 - Nodi della dashboard.

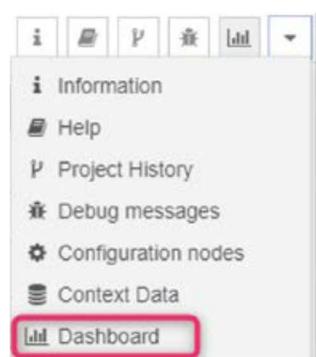


Figura 4-4 - Pannello di gestione della dashboard

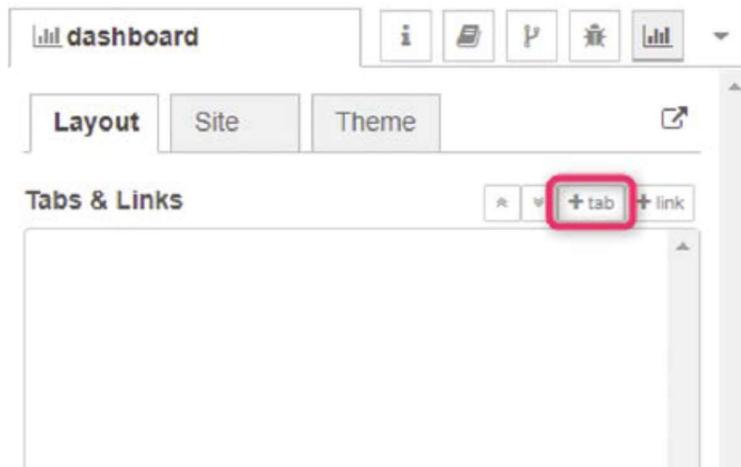


Figura 4-5 - Aggiunta di un Tab.

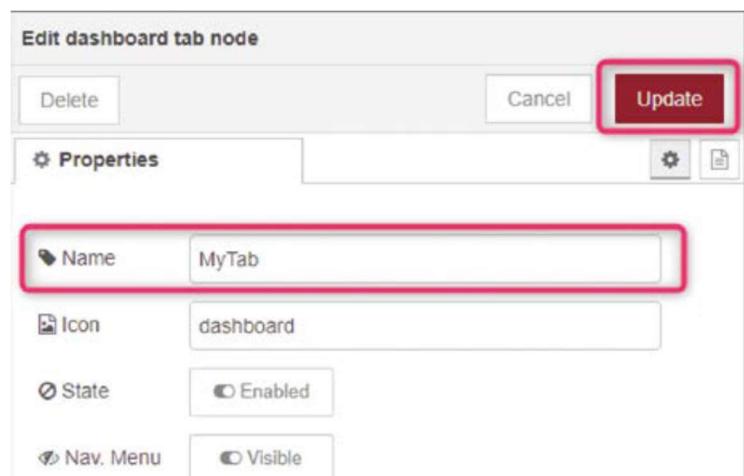


Figura 4-6 - Rinominare il TAB.

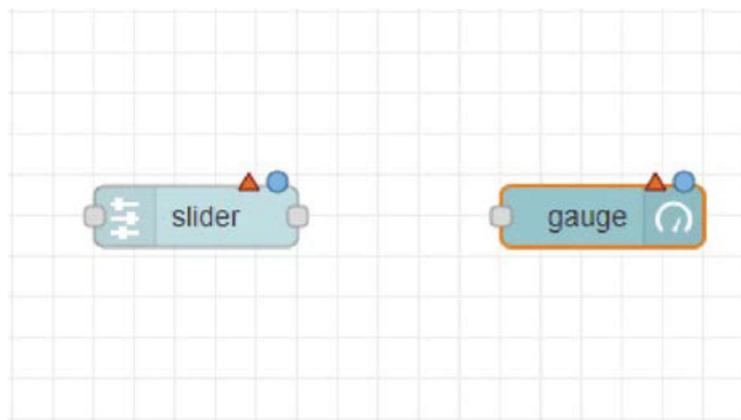


Figura 4-7 - Posizionamento di un nodo slider e un nodo gauge sul workspace.

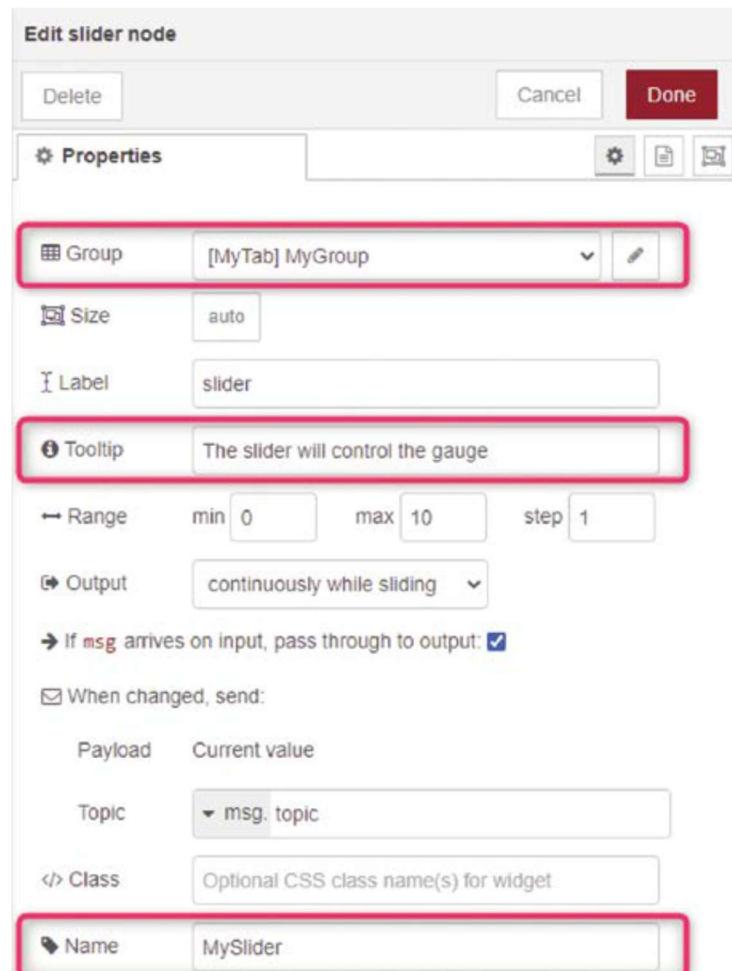


Figura 4-8 - Proprietà del nodo slider.

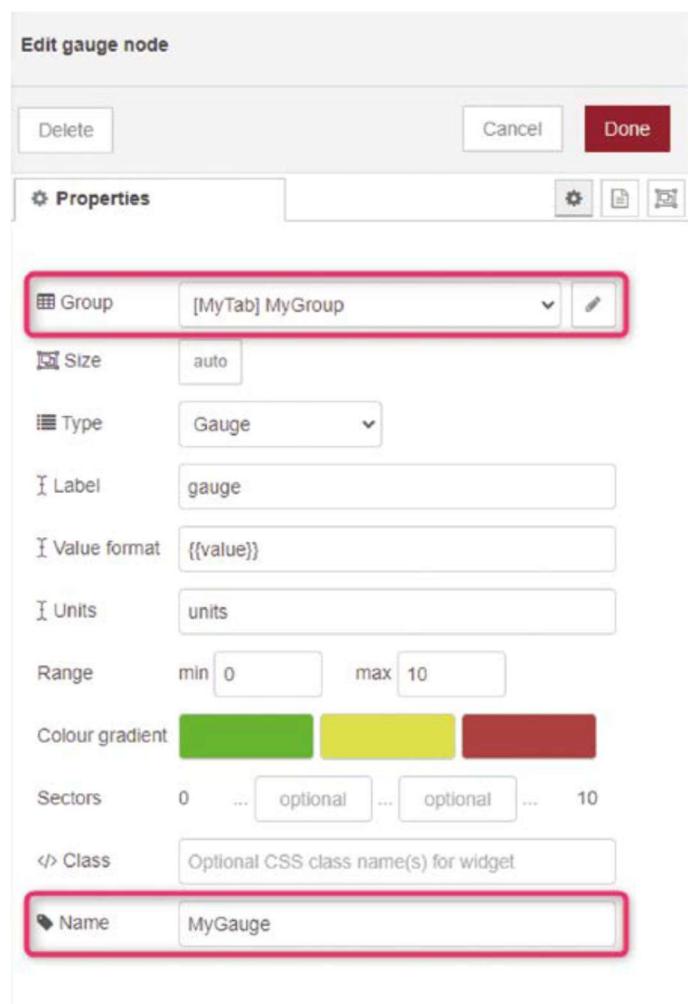


Figura 4-9 - Proprietà del nodo gauge.

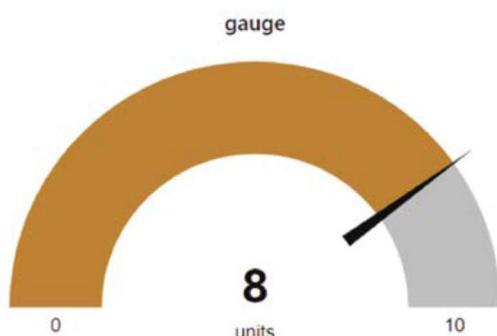


Figura 4-10 - Dashboard ottenuta dall'esempio.

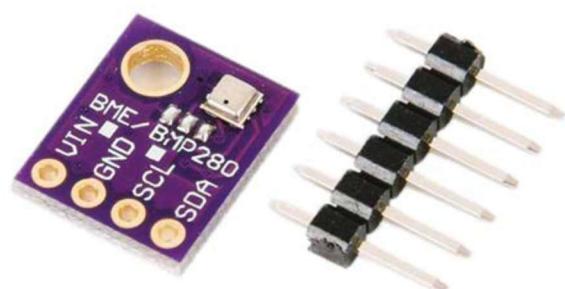


Figura 4-11 - Sensore BME280.

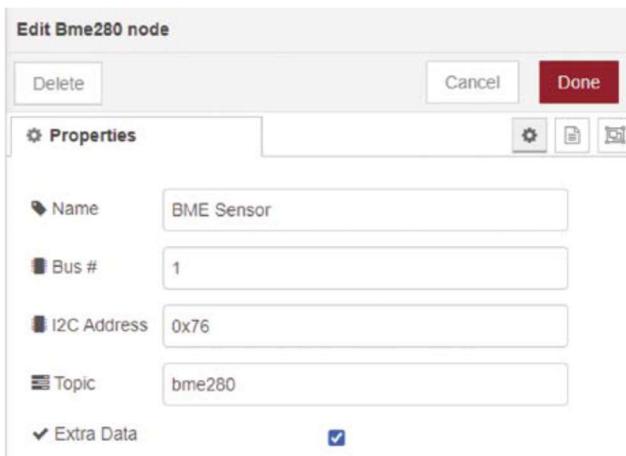


Figura 4-12 - Configurazione nodo BME280.

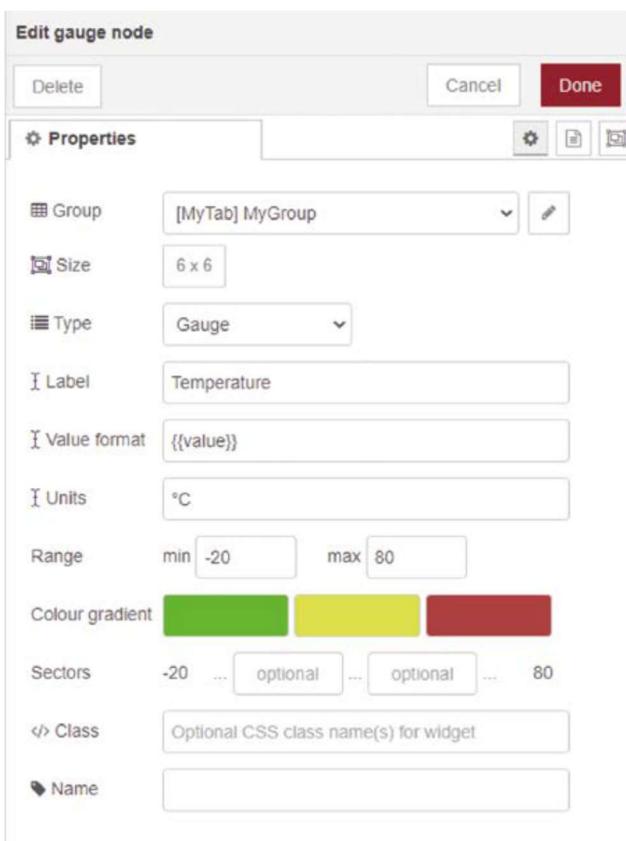


Figura 4-13 - Configurazione del gauge Temperature.

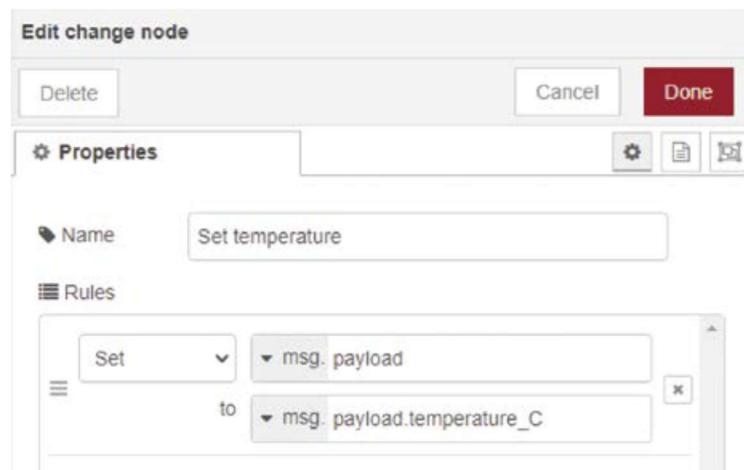


Figura 4-14 - Configurazione del nodo Change che estrae il dato di temperatura.

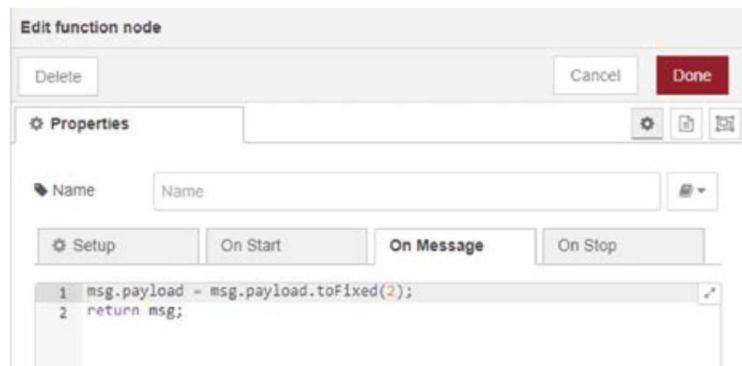


Figura 4-15 - Uno dei nodi function usati per correggere la precisione.



Figura 4-16 - Configurazione del nodo inject.

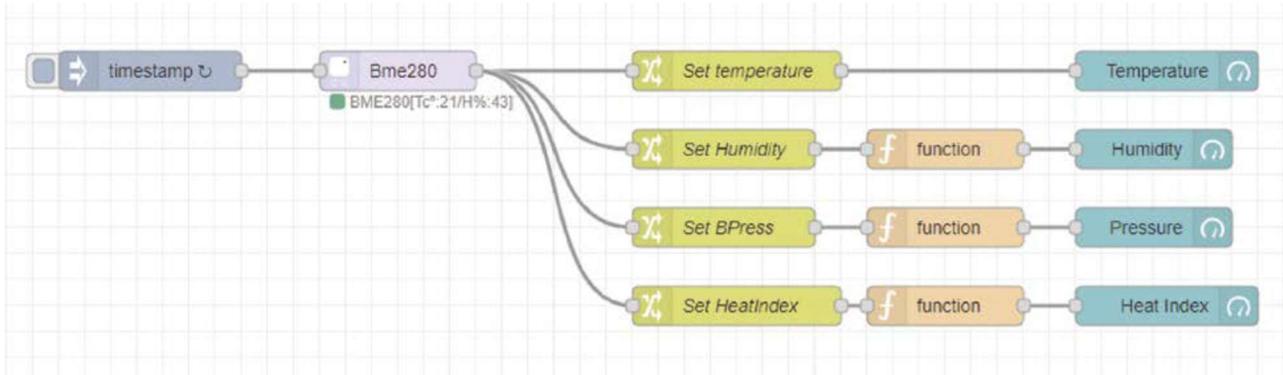


Figura 4-17 - Flow con i collegamenti completi.

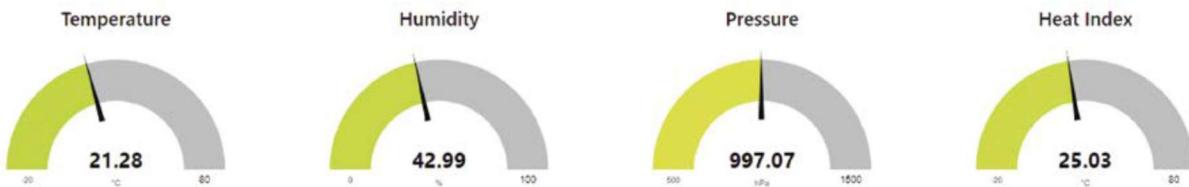


Figura 4-18 - Dashboard della stazione meteo.

5. Quinta Puntata.

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.

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 Figura 5-1 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.

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 si 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 Figura 5-2.

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.

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 simile, 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 Figura 5-3. 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

La struttura del flow è rappresentata in Figura 5-4.

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 request, 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 Figura 5-5.

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 Figura 5-6.

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 nodi 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 Figura 5-7.

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 configurazione da fare è settare la proprietà Type come “Reply to TCP”, come illustrato in Figura 5-8.

I nostri flow lato client e lato server sono ora pronti e possiamo semplicemente testare questa implementazione iniettando alcune stringhe nel flow ed esaminando ciò che viene ricevuto sulla finestra di debug (Figura 5-9), ossia l'eco della stringa inviata.

UDP

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

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.

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 Figura 5-10.

- **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. È 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 Figura 5-11.

Vediamo il dettaglio della configurazione dei due nodi UDP, partendo dal nodo UDP send, la cui finestra delle proprietà è riportata in Figura 5-12.

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 Figura 5-13.

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 Figura 5-14.

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 Figura 5-15 mostra l'output ottenuto nella finestra di debug di Node-RED.

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.
- 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 (Figura 5-16):

- **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 Figura 5-17:

Analizziamo la configurazione dei Nodi http, aiutandoci con la finestra delle proprietà del nodo Relay ON (Figura 5-18).

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`.

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 Figura 5-19, 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 Figura 5-20.

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 Figura 5-21.

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.

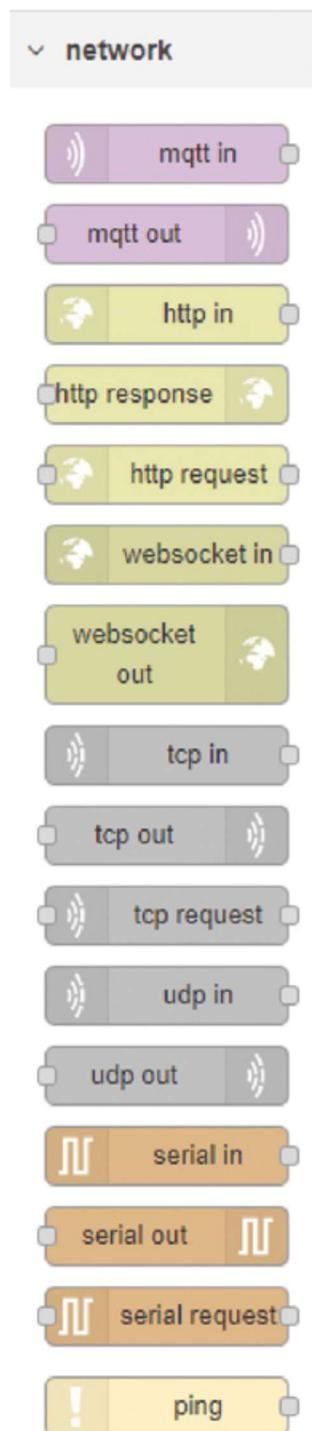


Figura 5-1 - Nodi della sezione Network.

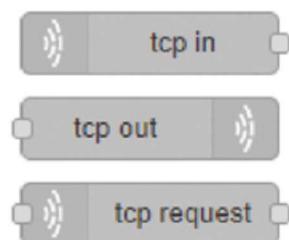


Figura 5-2 - Nodi TCP

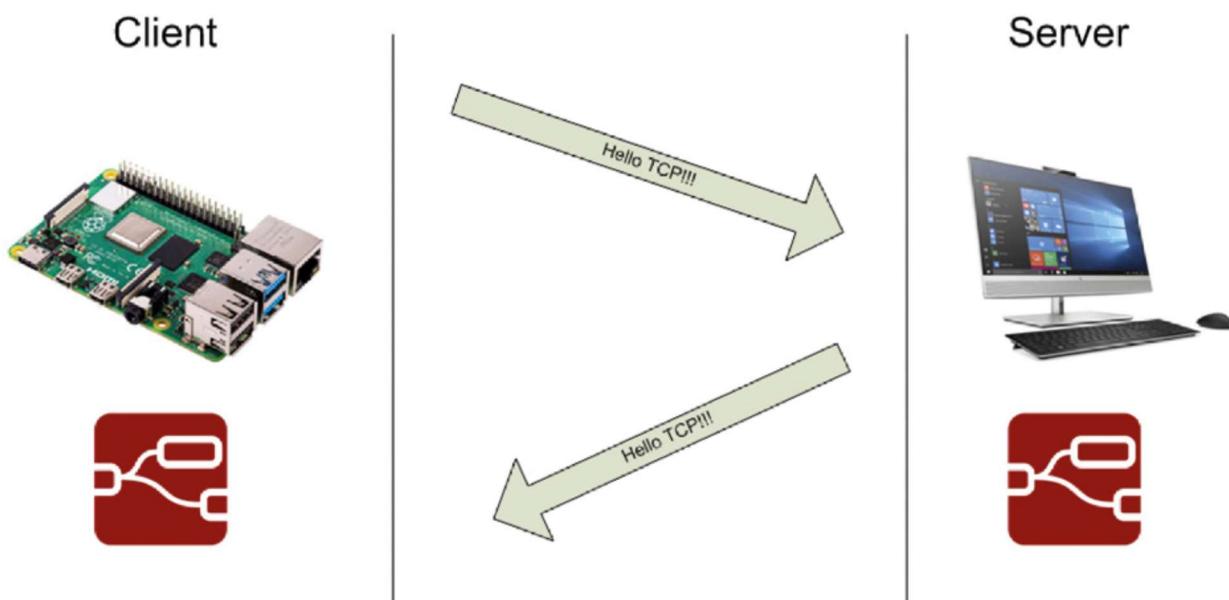


Figura 5-3 - Setup utilizzato per l'esempio TCP.



Figura 5-4 - Struttura del flow lato client.

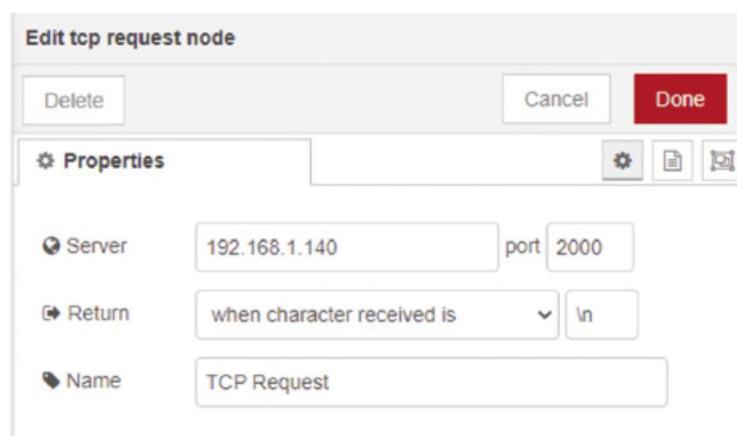


Figura 5-5 - Configurazione del nodo TCP request.

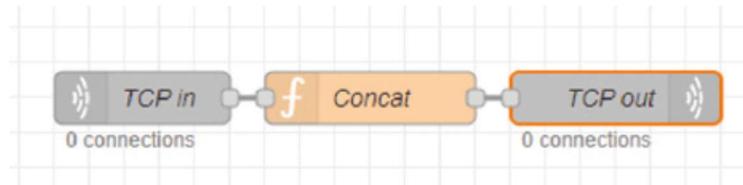


Figura 5-6 - Flow del TCP server.

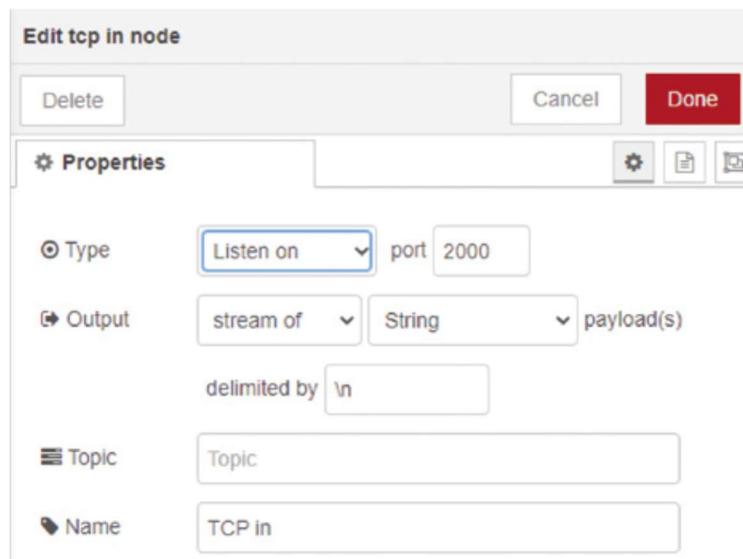


Figura 5-7 - Configurazione del nodo TCP in.

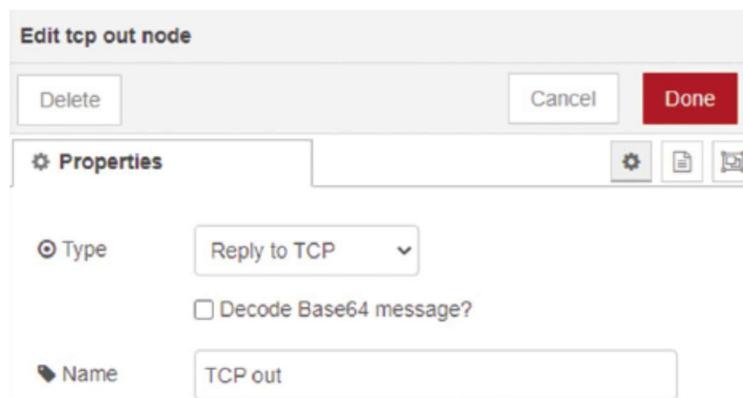


Figura 5-8 - Configurazione del nodo TCP out.

```

9/1/2022, 12:12:34 node: 925bd713e0a60017
msg.payload : string[13]
▶ "Hello TCP!!!"
```



```

9/1/2022, 12:12:36 node: 925bd713e0a60017
msg.payload : string[13]
▶ "Hello TCP!!!"
```



```

9/1/2022, 12:12:36 node: 925bd713e0a60017
msg.payload : string[13]
▶ "Hello TCP!!!"
```



```

9/1/2022, 12:12:36 node: 925bd713e0a60017
msg.payload : string[13]
▶ "Hello TCP!!!"
```

Figura 5-9 - Finestra di debug (lato client) che mostra l'eco ricevuto dal server.

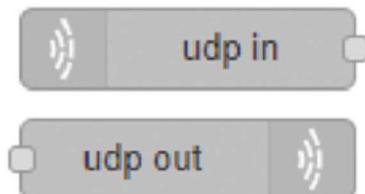


Figura 5-10 - Nodi UDP di Node-RED.

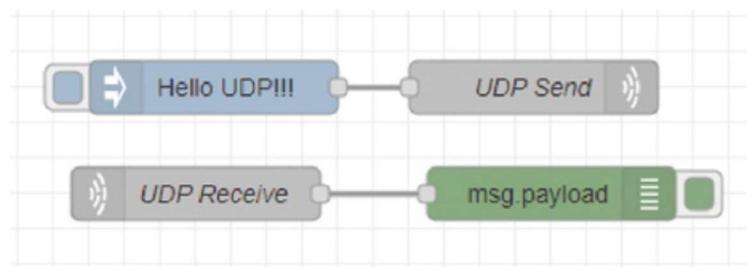


Figura 5-11 - Flow che implementa il nodo A.

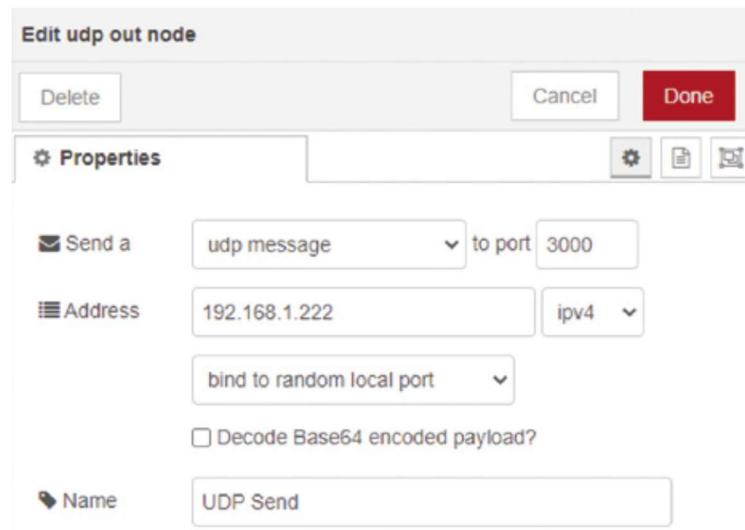


Figura 5-12 - Finestra delle proprietà del nodo UDP send.

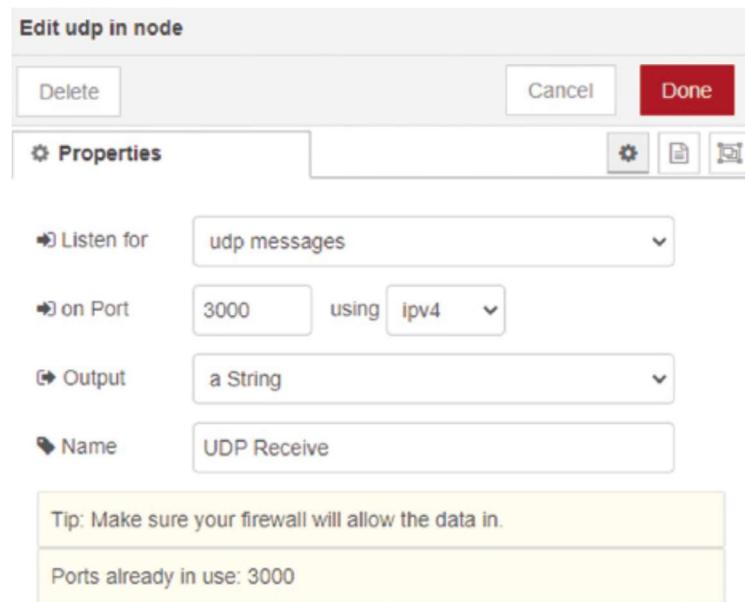


Figura 5-13 - Finestra delle proprietà del nodo UDP receive.

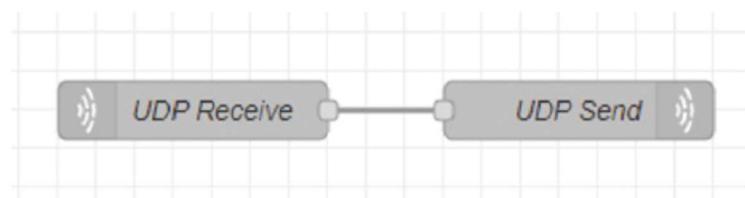


Figura 5-14 - Flow del nodo B.

```

9/1/2022, 16:28:10 node: 742ce5f54a8a035a
msg.payload : string[12]
"Hello UDP!!!"

9/1/2022, 16:28:10 node: 742ce5f54a8a035a
msg.payload : string[12]
"Hello UDP!!!"

9/1/2022, 16:28:10 node: 742ce5f54a8a035a
msg.payload : string[12]
"Hello UDP!!!"

9/1/2022, 16:28:10 node: 742ce5f54a8a035a
msg.payload : string[12]
"Hello UDP!!!"

```

Figura 5-15 - Output della finestra di debug del nodo A.

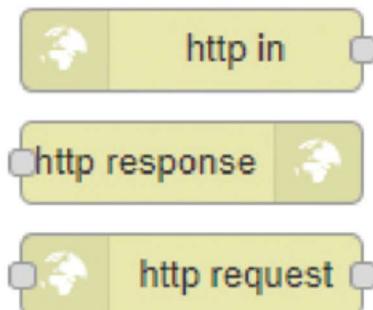


Figura 5-16 - Nodi Node-RED per la gestione del protocollo HTTP.

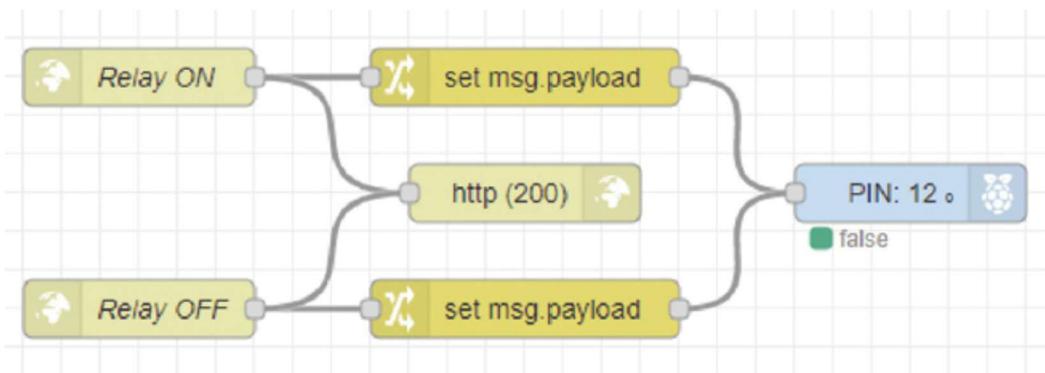


Figura 5-17 - Flow che implementa il webserver per il controllo di un relè tramite http.

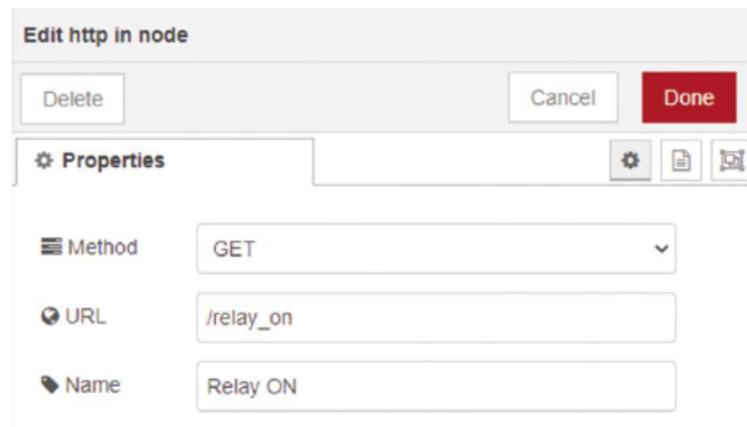


Figura 5-18 - Proprietà del nodo Relay ON (http in).

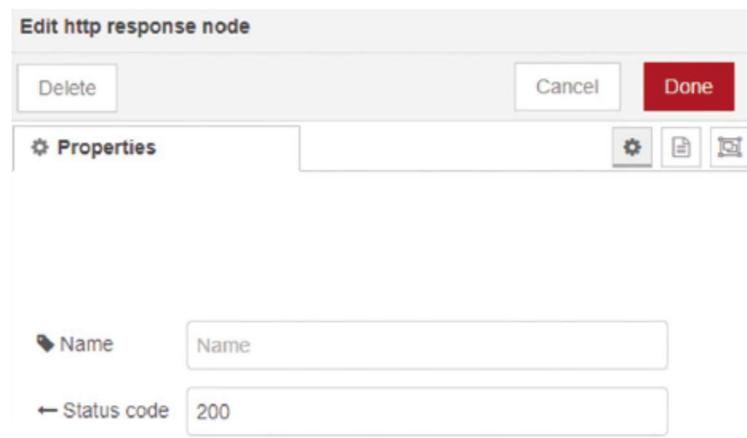


Figura 5-19 - Finestra delle proprietà del nodo http response.

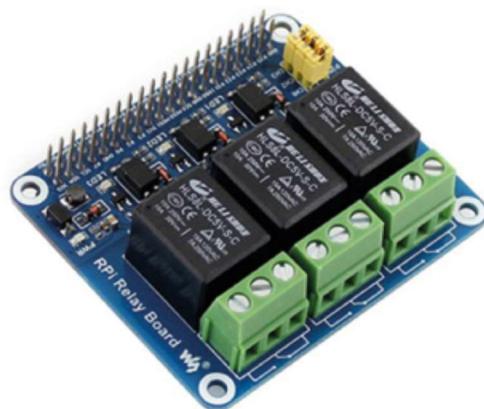


Figura 5-20 - HAT RPIRELLEBOARD.

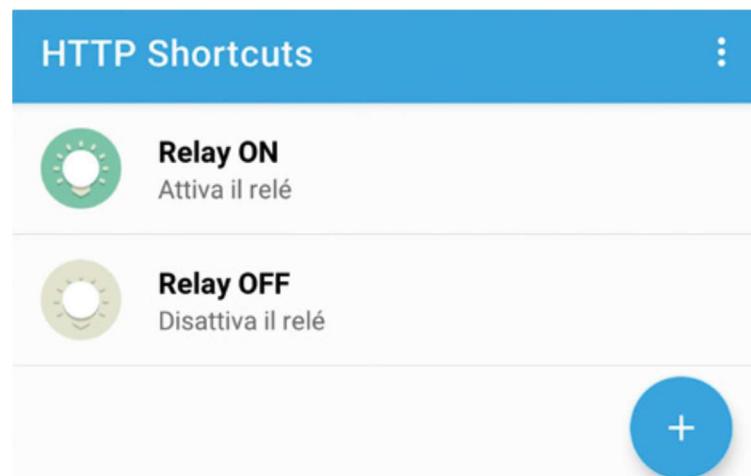


Figura 5-21 - Pannello per la gestione del relè realizzato tramite HTTP shortcuts.

6. Sesta Puntata.

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.

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 Figura 6-1.

- **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*.
- **websocket 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 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 Figura 6-2 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/](http://www.demo.piesocket.com/v3/) è possibile inserire un path interno che verrà utilizzato per lo scambio dati (nel nostro esempio */Elettronicaln*), 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 Figura 6-3.

A questo selezioniamo dal dropdown menu Type la voce “Connect To” e clicchiamo sull’icona a forma di matita per configurare il websocket client. La configurazione del websocket client è riportata in Figura 6-4.

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 (Figura 6-5) 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 proprietà è riportata in Figura 6-6.

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 Figura 6-7.

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 i 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 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 sottoscrittente) ad uno specifico topic non debba essere necessariamente un client singolo, ma il broker potrebbe inoltrare il messaggio anche ad un gruppo di nodi sottoscrittori lo stesso topic (quindi è nativamente supportato anche il broadcasting). La Figura 6-9 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 protocollo MQTT. Il 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 finché 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 Figura 6-10.

- **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 Figura 6-11 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>

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 (Figura 6-12). 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 implemeteremo con una istanza di Node-RED su PC) invia periodicamente lo stato di un LED and un nodo subscriber che implemeteremo 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 Figura 6-13, 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 Figura 6-14. 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
- **QoS:** 2
- **Retain:** false

Se tutto è stato fatto correttamente, dovremmo vedere una finestra delle proprietà come quella di Figura 6-15 (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 Figura 6-16.

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 Figura 6-17.

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 Figura 6-18.

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.



Figura 6-1 - Nodi Websocket in Node-RED.



Figura 6-2 - PieSocket test server.

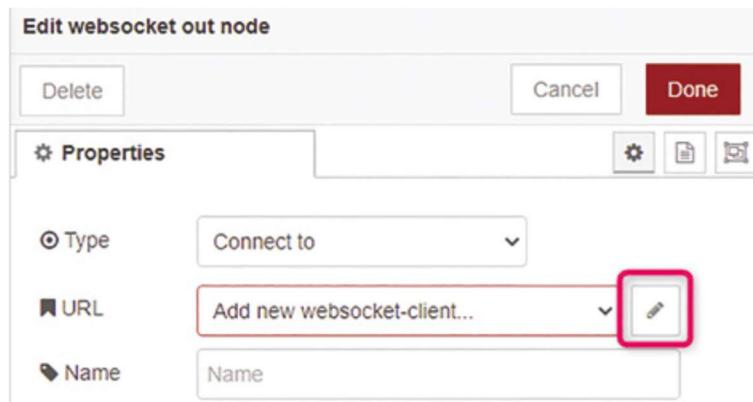


Figura 6-3 - Finestra proprietà del nodo websocket out.

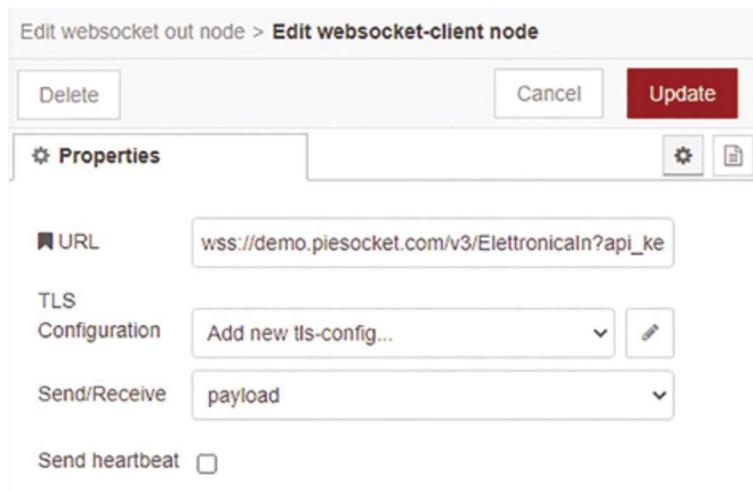


Figura 6-4 - Configurazione del websocket client node.

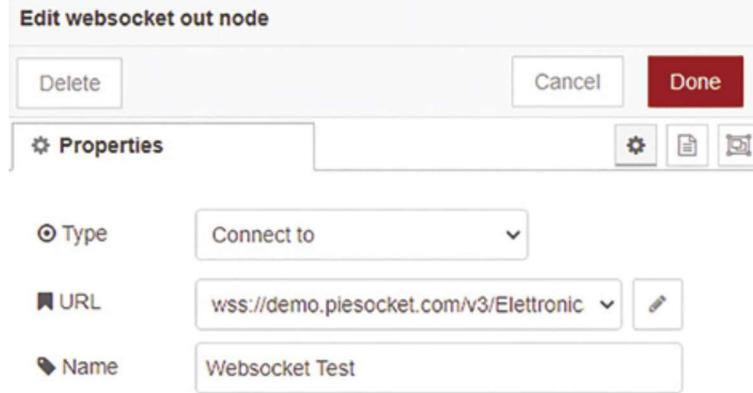


Figura 6-5 - Configurazione del nodo websocket in.

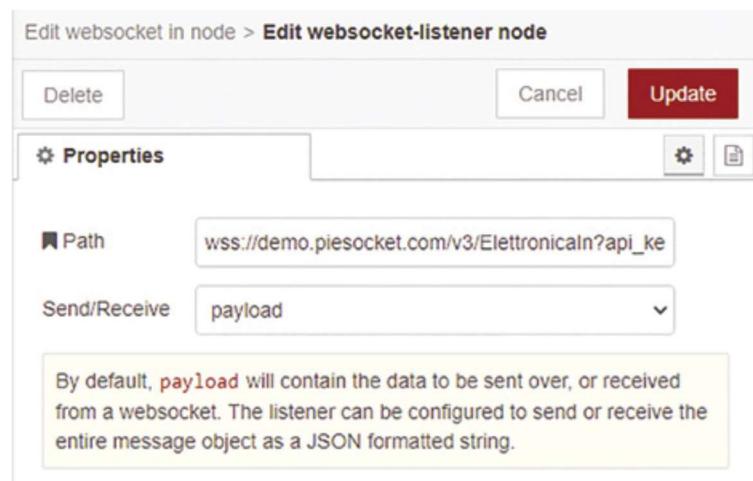


Figura 6-6 - Finestra proprietà del configuration node websocket-listener.

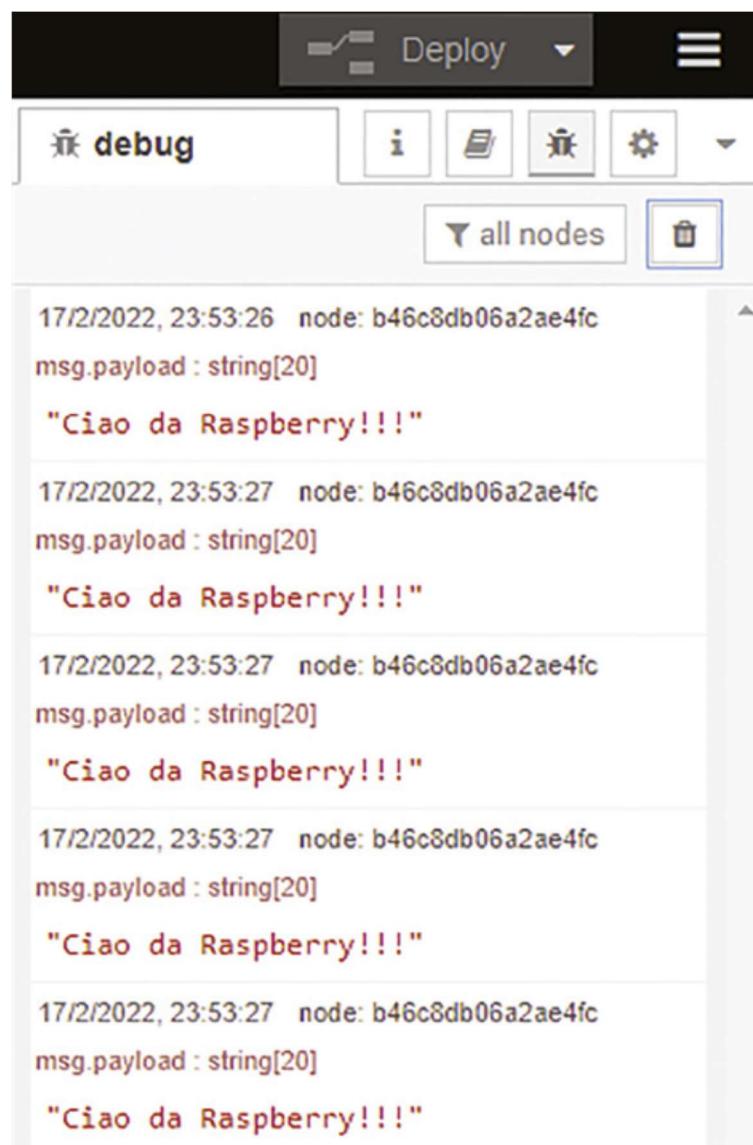


Figura 6-7 - Finestra di debug del nodo ricevitore.



Figura 6-8 - Logo del protocollo websocket.

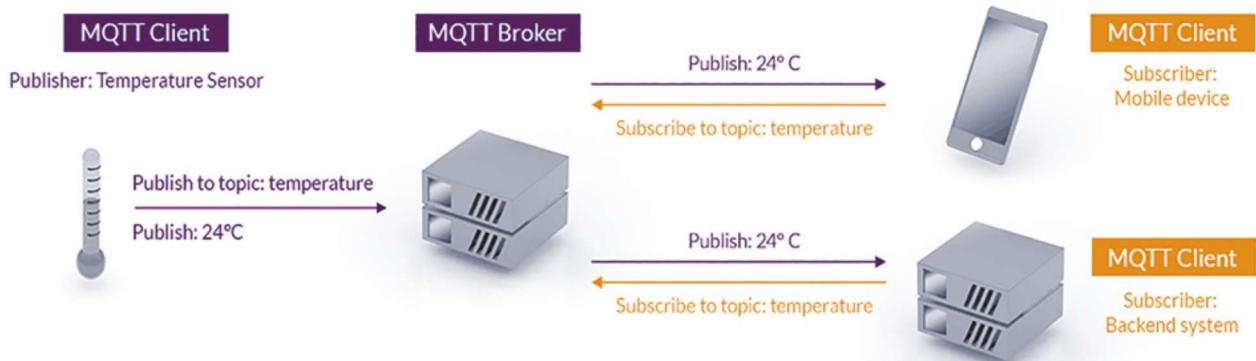


Figura 6-9 - Schematizzazione del modello pubblicazione-sottoscrizione utilizzato dal protocollo MQTT.



Figura 6-10 - Nodi MQTT in Node-RED.

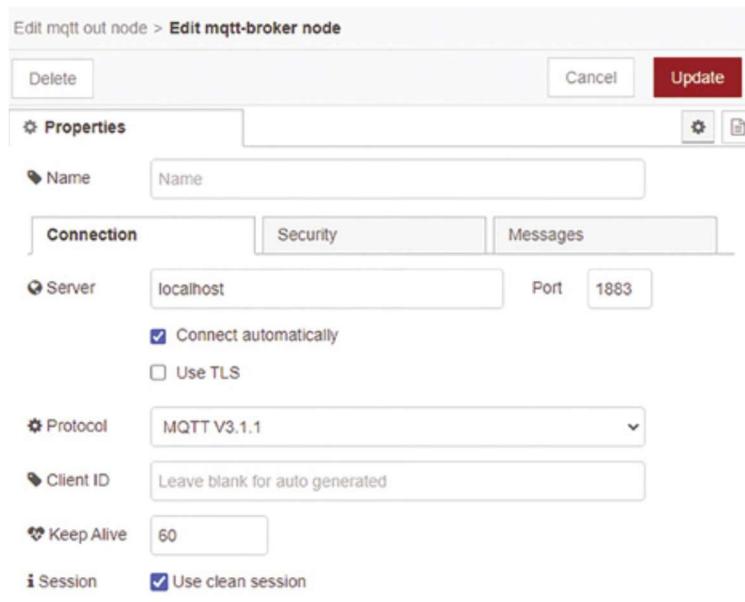


Figura 6-11 - Schermata di configurazione del nodo mqtt-broker.



Figura 6-12 - Nodo Aedes.

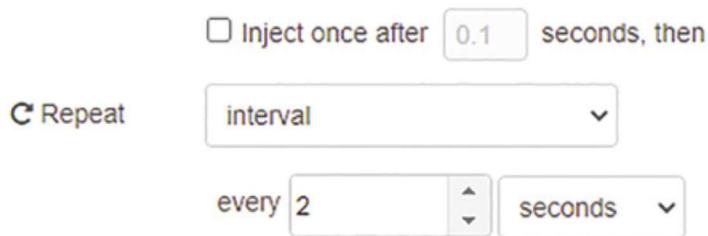


Figura 6-13 - Configurazione della periodicità del nodo Inject.

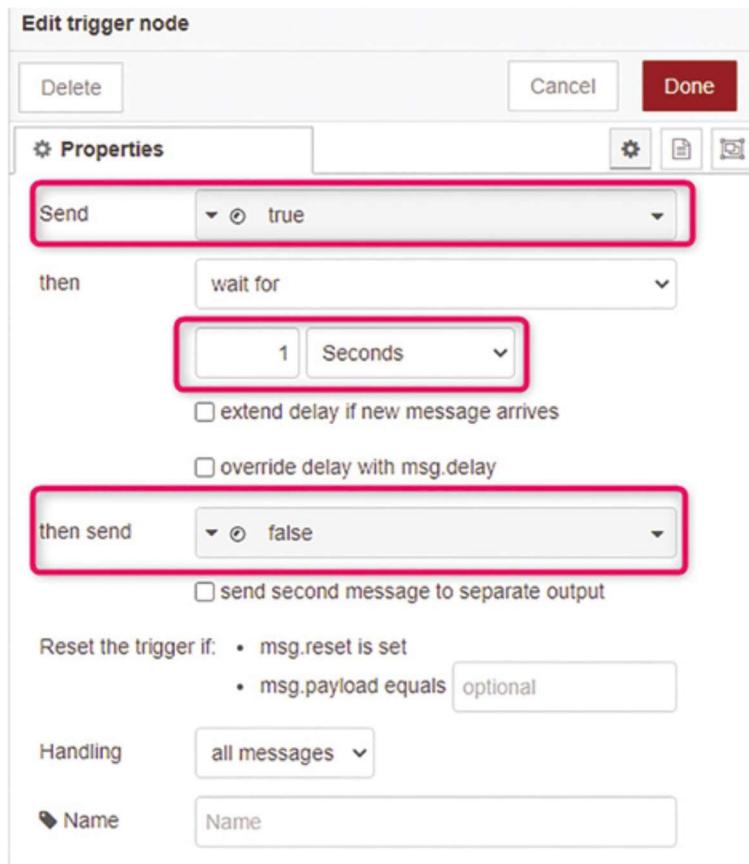


Figura 6-14 - Configurazione del nodo trigger.

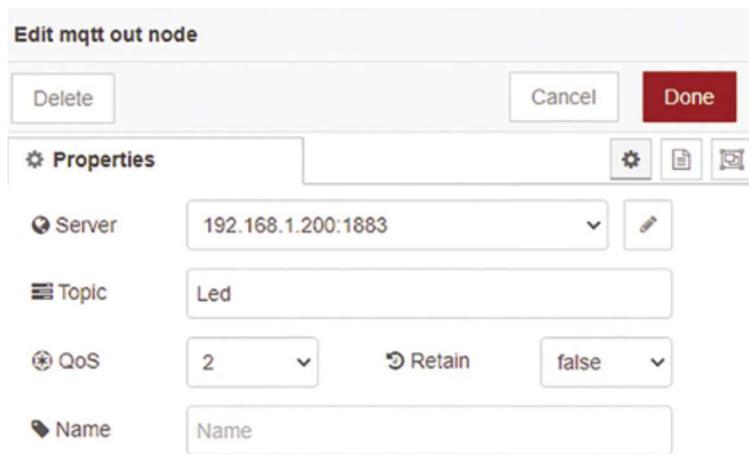


Figura 6-15 - Configurazione del nodo mqtt out.

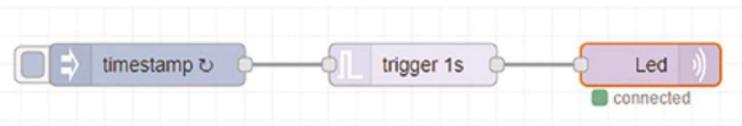


Figura 6-16 - Flow che implementa il nodo publisher.

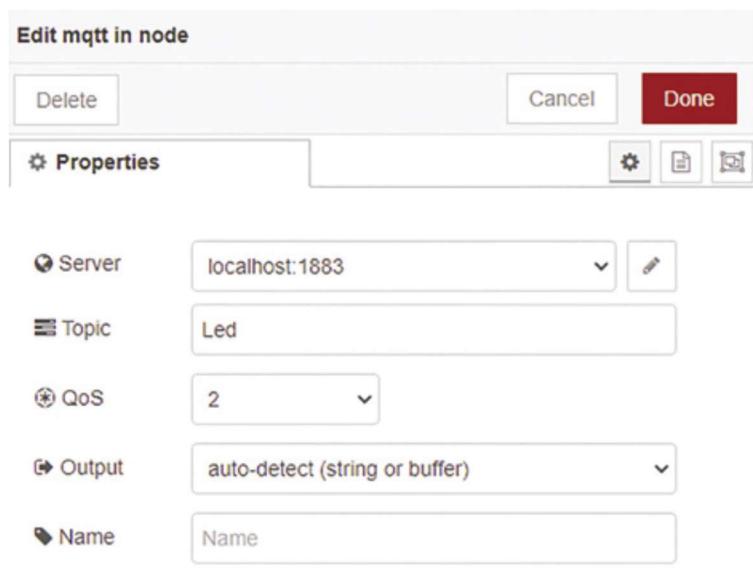


Figura 6-17 - Finestra delle proprietà del nodo mqtt in.

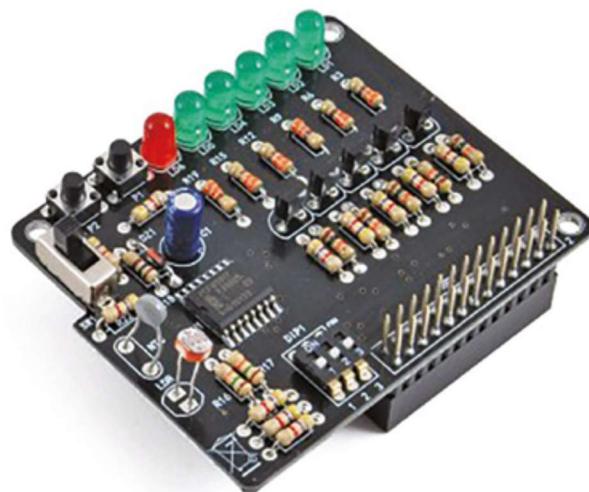


Figura 6-18 - Shield Raspberry Pi I/O di Futura Elettronica.

7. Settima Puntata.

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 parliamo dell'integrazione di funzionalità di messaggistica, sia tradizionale usando le email, che istantanea, facendo uso di applicazioni come Whatsapp e Telegram.

Abbiamo più volte posto l'accento, nelle puntate precedenti, sul fatto che Node-RED sia un ambiente con una forte predisposizione all'implementazione di applicazioni di Internet of Things e di come disponga di una serie di estensioni che ne facilitano di molto l'utilizzo in questi ambiti. Al giorno d'oggi, qualsiasi applicazione IoT che si rispetti non può più prescindere da alcune funzionalità fondamentali che ne permettano l'integrazione con delle applicazioni che siamo sempre di più abituati ad utilizzare quotidianamente sia con i nostri PC che con i nostri smartphone: le applicazioni di messaggistica. Noi siamo tipicamente abituati ad utilizzare questa tipologia di applicazioni per la comunicazione tra le persone, ma di fatto le applicazioni di messaggistica, grazie alle loro caratteristiche intrinseche di affidabilità, robustezza e semplicità d'uso, costituiscono anche un ottima interfaccia per la comunicazione uomo-macchina. Non potendo trattare, per ragioni di spazio, l'integrazione di Node-RED con tutte le applicazioni di messaggistica esistenti, tra cui si annoverano, oltre ai capostipiti WhatsApp e Telegram, anche Facebook Messenger, Viber, Signal, solo per citarne alcune, nella puntata odierna ci focalizzeremo esclusivamente su:

- Invio e ricezione di email tramite Node-RED
- Integrazione di Node-RED con WhatsApp
- Integrazione di Node-RED con Telegram

I campi di applicazione sono molteplici: la messaggistica ci consente di interagire con Node-RED usando, ad esempio Telegram o WhatsApp direttamente dal nostro smartphone, permettendoci quindi di controllare attuatori o acquisire dati usando queste applicazioni.

INVIARE EMAIL TRAMITE NODE-RED

Uno dei metodi più comuni per inviare messaggi è quello di utilizzare le email, e prima della comparsa delle applicazioni di messaggistica istantanea era anche uno dei più diffusi, oltre ai classici SMS, oramai caduti completamente in disuso perché soppiantati proprio dall'instant messaging, più economico e pratico. Le email tuttavia continuano ad essere un sistema interessante per scambiare informazioni tra un dispositivo ed una persona. Node-RED non fornisce nativamente nodi per la gestione dei protocolli di invio e ricezione di email, ma esistono svariati nodi tramite i quali è possibile aggiungere questo supporto.

Noi proponiamo il modulo node-red-node-email, che, come abbiamo visto già diverse altre volte, può essere installato direttamente dalla palette, utilizzando il nome del nodo stesso come chiave di ricerca.

Questa libreria fornisce l'accesso a 3 nodi specifici (**Figura 7-1**), che sono:

- **Email MTA:** Mail Transfer Agent. Questo nodo è un mail trasfer agent di test, che in questa puntata non tratteremo.
- **Email Send:** questo nodo permette di inviare il contenuto del *msg.payload* come email. Il contenuto di *msg.topic* può essere utilizzato per impostare l'oggetto del messaggio.
- **Email Receive:** questo nodo controlla periodicamente se sono state ricevute email e se ne trova di nuove invia un messaggio in uscita che contiene l'oggetto della mail su *msg.topic* e il body su *msg.payload*.

L'utilizzo di questi nodi è molto semplice, per inviare una email possiamo usare il nodo **Email Send**, del quale dovremo opportunamente configurare la finestra delle proprietà. Questo set di nodi non dispone di *configuration node*, quindi le configurazioni sono locali alla specifica istanza.

Le opzioni da configurare sono (si veda anche Figura 7-2, in cui è stato fatto un esempio con degli account non reali):

- **To:** identifica l'email del destinatario (nell'esempio la mail *destinatario@gmail.com*).
- **Server:** identifica il server di mail in uscita da utilizzare (di default *smtp.gmail.com*, ma può essere modificato con qualsiasi altro server smtp).
- **Port:** identifica la porta da utilizzare (di default la porta 465, ma può essere modificata in cui si utilizzi un server di posta diverso da quello di default).
- **UserId:** la user ID dell'account email dal quale si desidera inviare la mail (in questo caso la mail d'esempio *mario.rossi@gmail.com*).
- **Password:** la password dell'account.
- **Use secure connection e TLS option:** opzioni di sicurezza, entrambe abilitate di default.

Una volta configurato il nodo, per inviare una email, sarà sufficiente inviare sulla porta di ingresso del nodo un messaggio in cui il *msg.topic* costituisce l'oggetto del messaggio stesso e il *msg.payload* il suo contenuto. Per gli utenti gmail sarà necessario abilitare l'accesso da app meno sicure dai settings dell'account di google per consentire al nodo di funzionare, oppure abilitare una application password.

Vediamo invece adesso come gestire le email in ricezione. In questo caso il nodo da utilizzare è naturalmente Email Receive e le opzioni da configurare nella sua finestra delle proprietà sono le seguenti (si veda anche Figura 7-3):

- **Get Mail:** questa opzione permette di decidere come si vuole interrogare la casella di posta elettronica. E' offerta sia la possibilità di farlo quando il nodo riceve un messaggio in ingresso (ed in questo caso viene creata una opportuna porta), oppure automaticamente dopo un determinato intervallo di tempo (every X seconds).
- **Protocol:** identifica il protocollo utilizzato (IMAP o PO3)
- **Use SSL e Start TLS:** opzioni di sicurezza (di default l'SSL è attivato, mentre non viene eseguito il TLS).
- **Server:** l'indirizzo del server da utilizzare (di default *imap.gmail.com*, ma modificabile)
- **Port:** identifica la porta da utilizzare (di default la porta 993, ma può essere modificata nel caso in cui si utilizzi un server di posta diverso da quello di default)
- **UserId:** la user ID dell'account email dal quale si desidera ricevere la mail (in questo caso la mail d'esempio *mario.rossi@gmail.com*).
- **Password:** la password dell'account.
- **Folder:** identifica il folder di posta da controllare (di default è il folder INBOX, ma potrebbe eventualmente essere modificato per controllare un folder differente).
- **Disposition:** permette di decidere cosa fare del messaggio che viene letto (nulla, segnarlo come letto o cancellarlo).
- **Criteria:** permette di determinare la politica con la quale i messaggi vengono inviati in uscita (ossia se inviare quelli non letti, quelli senza risposta, etc...).

Come nel caso del nodo *send*, il *msg.topic* del messaggio di output contiene l'oggetto del messaggio, mentre il *msg.payload* ne contiene il testo. Sono presenti anche informazioni aggiuntive come la data di ricezione ed il mittente, presenti su altri campi del messaggio in uscita (Figura 7-4).

INTEGRAZIONE CON WHATSAPP

Dopo aver visto come gestire le email passiamo all'integrazione con la prima app di messaggistica istantanea: WhatsApp. Ci sono diverse possibilità per integrare Node-RED con WhatsApp, noi segnaliamo la libreria *node-red-contrib-whin*, che ci è sembrata la più completa tra quelle attualmente disponibili e anche quella con le migliori performance. Per installarla, come di consueto, usiamo l'opzione "Manage Palette" del menù principale e cerchiamo la nostra libreria.

Una volta installata noterete che la libreria Whin mette a disposizione 3 nodi (si veda anche Figura 7-5):

- **whin - send**: permette di inviare un messaggio WhatsApp al numero indicato nella configurazione.
- **whin - receive**: permette di ricevere messaggi WhatsApp dal numero indicato nella configurazione.
- **whin - confirm**: permette di inviare una richiesta con risposta "chiusa" (Yes o No) al numero indicato nella configurazione.

Oltre a questi tre nodi è presente anche il classico *configuration node whin-config* per le configurazioni generali.

Il set di nodi della libreria Whin usa uno shared gateway per la comunicazione, ed è quindi necessario registrarsi al servizio inviando uno specifico messaggio ad un numero, che successivamente rilascerà una API key da utilizzarsi per accedere alla chat.

I dettagli per la registrazione sono indicati nella sezione "Set-up and Usage" della seguente pagina web:

<https://flows.nodered.org/node/node-red-contrib-whin>.

Il numero utilizzato per la registrazione è anche quello da cui riceveremo e a cui invieremo i messaggi (potete registrarli in rubrica con il nome che preferite).

Una volta ottenuto il token possiamo posizionare un nodo whin – send sul workspace e procedere al setup del configuration node. Facciamo quindi doppio click sul nodo whin – send e poi clicchiamo sull'icona a forma di matita per accedere alla finestra delle proprietà del *configuration node*. Qui inseriamo un nome per la configurazione (nell'esempio Whin – Test) e poi il numero di telefono dal quale abbiamo fatto la registrazione al servizio e il Token ottenuto in precedenza. E' importante ricordare che il numero di telefono va inserito con il relativo country code, ma senza il carattere '+' all'inizio e che il numero di telefono va inserito senza simboli '-' o spazi all'interno.

Una volta configurato correttamente il config node possiamo salvare ed eseguire il deploy.

Per testare il nodo possiamo usare un semplice nodo inject contenente una stringa qualsiasi (ad esempio "Hello from Node-RED!!!") collegata alla porta di ingresso del nodo whin – send, come illustrato nel flow di Figura 7-6.

Una volta cliccato sul pulsante del nodo inject il messaggio verrà inviato sulla chat whatsapp che abbiamo con il numero tramite il quale abbiamo registrato il servizio.

INTEGRAZIONE CON TELEGRAM

Passiamo ora all'integrazione di node-RED con un'altra diffusissima applicazione di messaggistica istantanea: Telegram.

Anche in questo caso, esistono diverse librerie in Node-RED che permettono l'interfacciamento con Telegram; quella che noi proponiamo in questa puntata è la *node-red-contrib-telegrambot*, che fa uso di un bot telegram.

La libreria fornisce i seguenti nodi, riportati anche in Figura 7-7:

Telegram Receiver: questo nodo invia un messaggio di output quando dei messaggi sono ricevuti nella chat. I dettagli del messaggio sono contenuti nei seguenti campi di msg.payload:

- Content: il contenuto del messaggio
- Type: la tipologia di dati
- msgId: il numero che rappresenta l'ID del messaggio
- chatId: il numero che rappresenta l'ID della chat

Telegram Command: questo nodo invia un messaggio quando uno specifico comando è ricevuto dal bot sulla chat. Esempi di comando sono **/On** o **/Temperature**.

Telegram Event: questo nodo invia un messaggio quando uno specifico evento è ricevuto dal bot sulla chat

Telegram Sender: questo nodo invia un messaggio nella chat quando riceve un messaggio internamente. Il contenuto di *msg.payload.content* viene scritto sulla chat. Deve inoltre essere trasmesso la corretta chat ID, contenuta nel campo *msg.payload.chatId*.

Telegram Reply: questo nodo invia un messaggio quando nella chat viene inviata una risposta ad un messaggio specifico.

Oltre a questi nodi è presente anche un *configuration node* chiamato **Telegram Bot** che permette di impostare alcune proprietà del bot con cui si desidera comunicare (come ad esempio la API Key) ed altri dettagli della comunicazione.

Prima di utilizzare la libreria è necessario creare un bot telegram, in modo da recuperare la API Key necessaria per collegarsi al bot.

Per creare un bot su Telegram è necessario richiederne la creazione ad uno speciale bot, denominato BotFather (può essere trovato cercando **@BotFather** sulla barra di ricerca di telegram). BotFather ha uno specifico comando, **/newbot**, che può essere utilizzato per creare un nuovo bot. Una volta inviato questo comando bisogna seguire le istruzioni del BotFather e dare al bot un nome confidenziale (può essere qualsiasi cosa, nel nostro esempio è NrBot) e poi uno username, che invece deve essere univoco e terminare con suffisso "bot". Qui potranno essere necessari alcuni tentativi e il nome finale potrebbe essere stravagante, in quanto esistono molti bot Telegram e quindi ci sono molti username.

BotFather vi indicherà pazientemente se lo username che avete selezionato è già in uso, fino a quando non ne troverete uno. Trovato il nome giusto BotFather vi fornirà l'API key necessaria per comunicare con il bot.

Un esempio di creazione di un bot è riportato in Figura 7-8.

A questo punto va recuperata la chatId; per farlo possiamo semplicemente posizionare sul workspace un nuovo Telegram receiver ed impostare le proprietà del configuration node Telegram bot. Clicchiamo quindi 2 volte sul nodo Telegram receiver che abbiamo posizionato sul workspace e poi sull'icona a forma di matita presente a fianco del campo Bot, come illustrato in Figura 7-9. A questo punto possiamo configurare le proprietà del nostro Telegram Bot, inserendo le due stringhe salvate in precedenza, ossia lo username del bot e la API key, come illustrato in Figura 7-10.

In questa finestra delle proprietà è anche possibile impostare degli altri parametri relativi alla gestione del bot, ma per questo primo semplice esempio possiamo mantenere i valori di default.

Possiamo ora cercare il nostro bot su Telegram (sempre tramite lo username), avviarlo ed inviare un semplice messaggio al nostro nodo *Telegram Receiver* su Node-RED.

Se intercettiamo l'output del nodo con un nodo debug (come evidenziato in Figura 7-11), otterremo un output simile a quello di Figura 7-12, dal quale possiamo ricavare la chatID, che ci servirà per la parte di trasmissione.

ESEMPIO PRATICO: CONTROLLARE UN LED TRAMITE TELEGRAM

A questo punto abbiamo acquisito sufficienti informazioni per implementare un semplice progetto pratico in cui controlleremo un LED attraverso Telegram, facendo in modo che ad un opportuno set di comandi scritto sulla chat con il nostro bot il LED venga acceso e spento. Faremo anche in modo che il nostro flow Node-RED restituisca anche indietro una stringa che indica lo stato del LED.

Per implementare questo semplice flow ci occorrono:

- 2 nodi Telegram Command
- 2 nodi Change
- 1 nodo Rpi Gpio out
- 1 nodo Function
- 1 nodo Telegram sender

Iniziamo con la parte del flow dedicata alla ricezione, posizionando i due nodi Telegram Command e i due nodi change ed il nodo rpi gpio out come illustrato in Figura 7-13. Il nodo rpi gpio out deve controllare il Pin che collegheremo al LED. Se usiamo nuovamente lo shield Rpi I/O (codice FT1060) di Futura Elettronica, possiamo connettere questo nodo al GPIO 12 che è, a sua volta, collegato al primo LED.

A questo punto configureremo la finestra delle proprietà dei due nodi Telegram Command, impostando come Bot il bot creato in precedenza e come comando in un caso / ledon (per accendere il LED) e nell'altro /ledoff (per spegnerlo).

Il resto delle opzioni di questi due nodi può essere lasciato al valore di default. In Figura 7-14 è rappresentata la finestra delle proprietà del nodo che riceve il comando di accensione del LED. Passiamo adesso ai nodi change, che ci occorrono per modificare il valore del campo payload per adattarlo a quello necessario ad accendere e spegnere il LED tramite il nodo rpi gpio out.

I due nodi devono avere ciascuno una semplice regola che setta il *msg.payload* a true in corrispondenza del comando /ledon ed a false in corrispondenza di /ledoff.

In Figura 7-15 è riportato l'esempio sempre nel caso di accensione del LED.

A questo punto la parte di ricezione ed attuazione del comando è completa e possiamo già eseguire un primo test inviando sulla chat del nostro bot i comandi /ledon e /ledoff per accendere e spegnere il LED.

Tuttavia ci manda ancora la ricezione del feedback sulla chat, che ci indica che il comando è stato ricevuto ed eseguito. Aggiungiamo quindi il nodo function ed il nodo Telegram Sender al flow e colleghiamoli come illustrato in Figura 7-16. Per fornire correttamente la risposta dobbiamo prima di tutto configurare il nodo Telegram Sender, referenziando il solito configuration node che contiene la configurazione del nostro bot selezionandola dal campo bot della finestra delle proprietà del nodo. Poi dobbiamo inserire un piccolo snippet di codice javascript nel function node, in quanto il nodo sender si aspetta di ricevere un messaggio con un oggetto javascript con una specifica formattazione, contenente, all'interno del campo msg.payload:

I bot di Telegram

In questa puntata abbiamo introdotto l'integrazione di Node- RED con applicazioni di messaggistica istantanea, tra cui Telegram, e di conseguenza di siamo imbattuti in una delle caratteristiche peculiari di questa applicazione: i 'bot'. Ma cosa sono precisamente i bot di Telegram? I bot, come il nome suggerisce, non sono altro che degli utenti artificiali di Telegram (dei programmi quindi) che possono interagire sia nelle chat singole che nelle chat di gruppo. Un bot è facilmente riconoscibile in telegram, in quanto il suo username termina sempre con il suffisso 'bot'. Questi programmi interagiscono con gli utenti essenzialmente rispondendo a dei comandi, che possono essere impartiti tramite il

carattere speciale '/'. Tipicamente ogni bot può listare la sua lista di comandi rispondendo al comando /help. Nel corso degli anni sono stati sviluppati molteplici bot per Telegram, una lista completa e divisa per categorie che può essere consultata è lo StoreBot (<https://storebot.me/bots/>).

Alcuni bot sono fondamentali per il funzionamento dell'ecosistema, come il BotFather (@botfather), che permette di registrare nuovi bot, o il TelegramStoreBot (@store_bot), che funge da motore di ricerca per i bot e permette di visualizzare, direttamente nella chat di Telegram, l'elenco completo dei bot presenti sulla piattaforma. Altri bot sono invece simili a programmi di utilità, che possono quindi semplificarcici alcuni aspetti della nostra vita quotidiana, come ad esempio TrackBot (@TrackBot), un bot che permette di tenere sotto controllo le nostre spedizioni, offrendo un servizio di monitoraggio dei corrieri italiani e stranieri, o TranslateBot (@Translate_Bot), che permette di tradurre istantaneamente delle parole scritte all'interno della chat. Infine esistono anche bot pensati per l'intrattenimento, come Enigmibot (@enigmibot), un bot che sottopone degli enigmi, oppure 'culturali' come DottorBot (@dottor_bot), un bot dedicato alla cultura tecnico-scientifica.

- un campo content (string), che rappresenta il contenuto del messaggio;
- un campo type (string), che rappresenta la tipologia di messaggio, nel nostro caso useremo la stringa "message" che identifica un semplice messaggio di testo;
- un campo chatId (number), che contiene la chatId che abbiamo recuperato in precedenza.

Lo snippet da inserire nel nodo function è riportato nel Listato 1. A questo punto il nostro flow è completo e possiamo ritestarlo. Noterete, a questo punto, che in corrispondenza dei comandi /ledon e /ledoff, oltre all'attuazione del LED riceveremo anche la risposta, come illustrato in Figura 7-17.

Listato 1 – Snippet di codice Javascript da inserire nel nodo Function

```
if (msg.payload == true)
{
  msg.payload = {
    content : "The Led is ON",
    type : "message",
    chatId : 738459487
  };
}
else
{
  msg.payload = {
    content : "The Led is OFF",
    type : "message",
    chatId : 738459487
  };
}
return msg;
```

CONCLUSIONI

Siamo arrivati alla conclusione di questa puntata in cui abbiamo visto come integrare Node-RED con varie applicazioni di messaggistica, partendo dalle semplici email, fino ad arrivare all'integrazione di applicazioni più complesse e di recente introduzione come Whatsapp e Telegram. Nella prossima puntata ci occuperemo dell'integrazione di Node-RED con un'altra tipologia di dispositivi che sta iniziando sempre di più a diffondersi nelle nostre case: gli assistenti vocali.

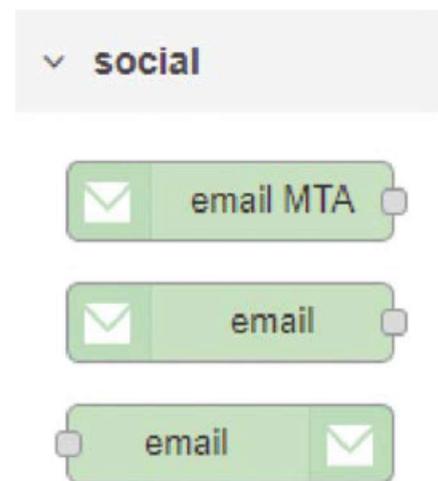


Figura 7-1 - Nodi email Node-RED.

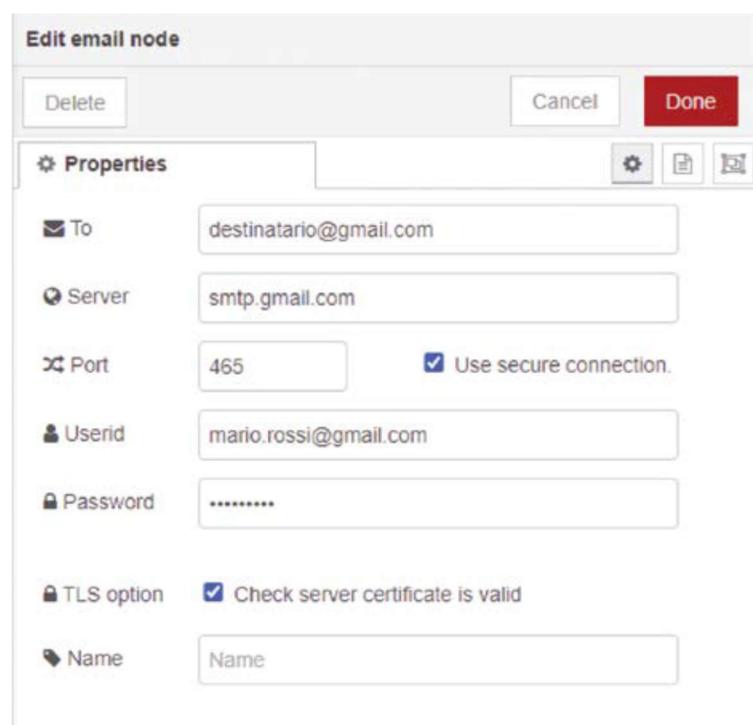


Figura 7-2 - Esempio di configurazione del nodo email send.

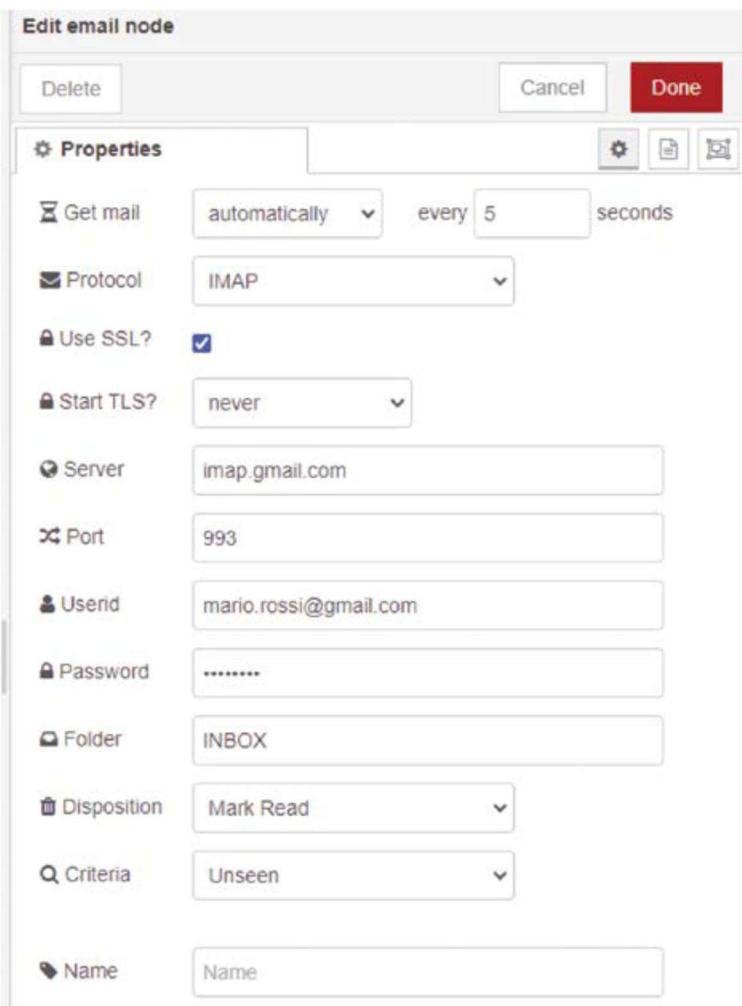


Figura 7-3 - Esempio di configurazione del nodo email receive.

```

27/3/2022, 12:13:24  node: a6fcc122d711c768
Ciao : msg : Object
  ▼ object
    ▶ payload: "Mail di test"
      topic: "Ciao"
      date: "2022-03-27T10:13:13.000Z"
    ▶ header: object
      ▶ html: "<div dir='auto'>Mail di test</div>""
      from: (redacted)
      attachments: array[0]
      _msgid: "b6d5a71454df543b"
  
```

Figura 7-4 - Esempio di complete msg object inviato dal nodo Email receive quando viene ricevuto un nuovo messaggio di posta.



Figura 7-5 - Nodi della libreria Whin.

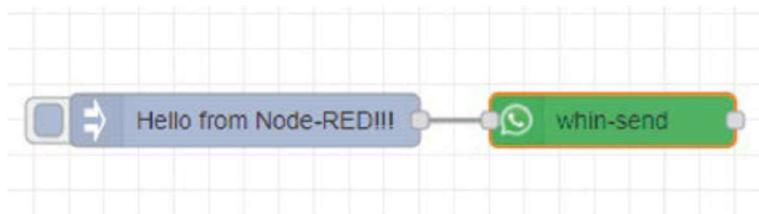


Figura 7-6 - Flow di test del nodo whin-send.



Figura 7-7 - Nodi della libreria Telegrambot.

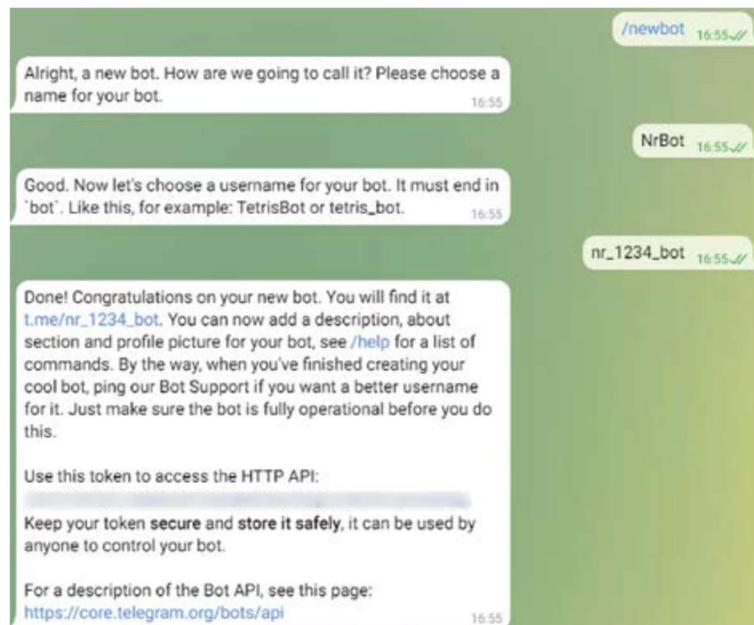


Figura 7-8 - Creazione di un nuovo bot.

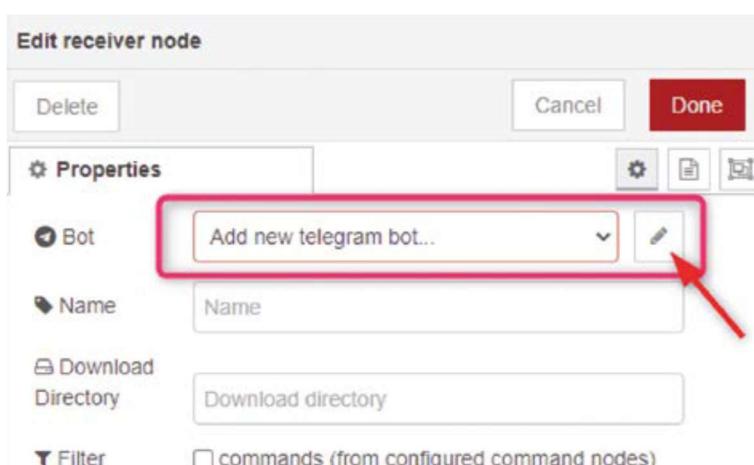


Figura 7-9 - Accesso alla finestra delle proprietà del configuration node Telegram Bot.

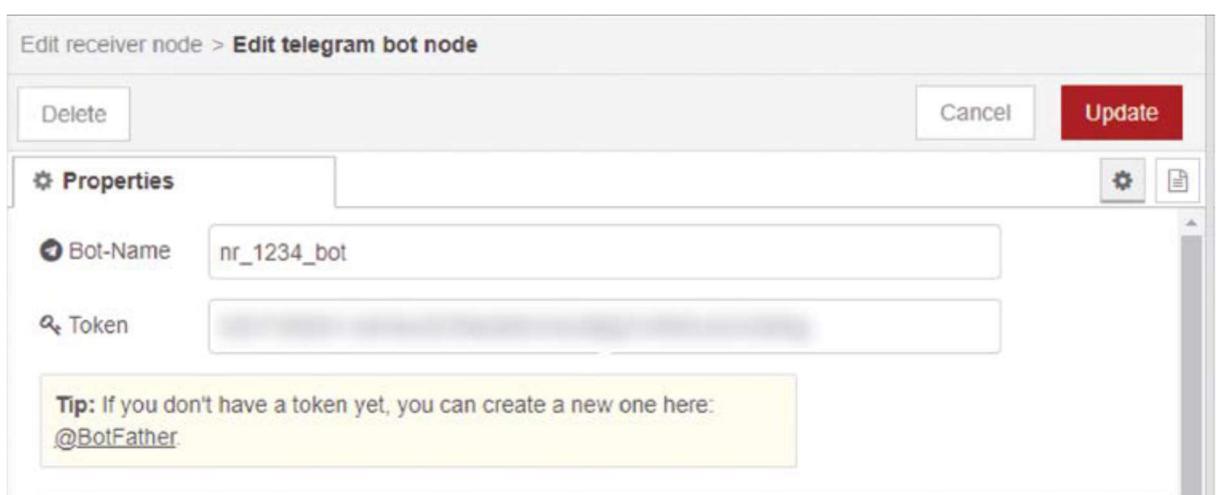


Figura 7-10 - Inserimento dello username e della API key nella finestra delle proprietà del configuration Node Telegram Bot.

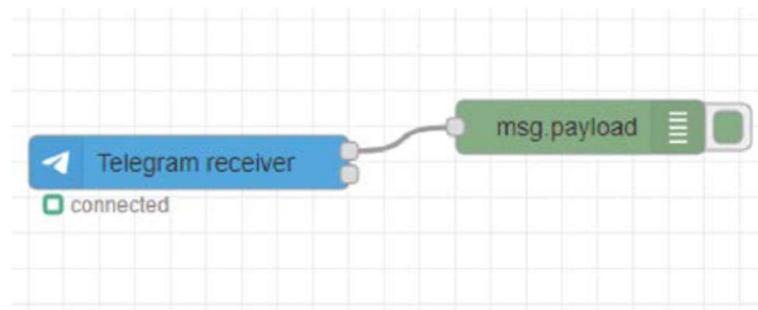


Figura 7-11 - Ricezione di un messaggio per ottenere la ChatId.

```
27/3/2022, 16:59:14  node: 06022c6581885800
msg.payload : Object
  ↴ object
    chatId: 738459487
    messageId: 2
    type: "message"
    content: "ciao"
    date: 1648393155
```

Figura 7-12 - Log del messaggio ricevuto tramite nodo Telegram receiver.

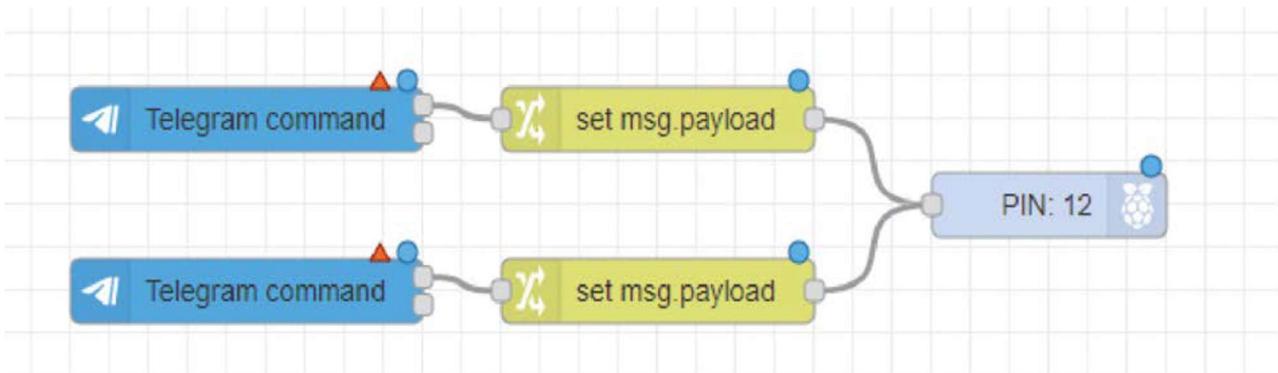


Figura 7-13 - Posizionamento dei nodi per la ricezione ed attuazione del comando.

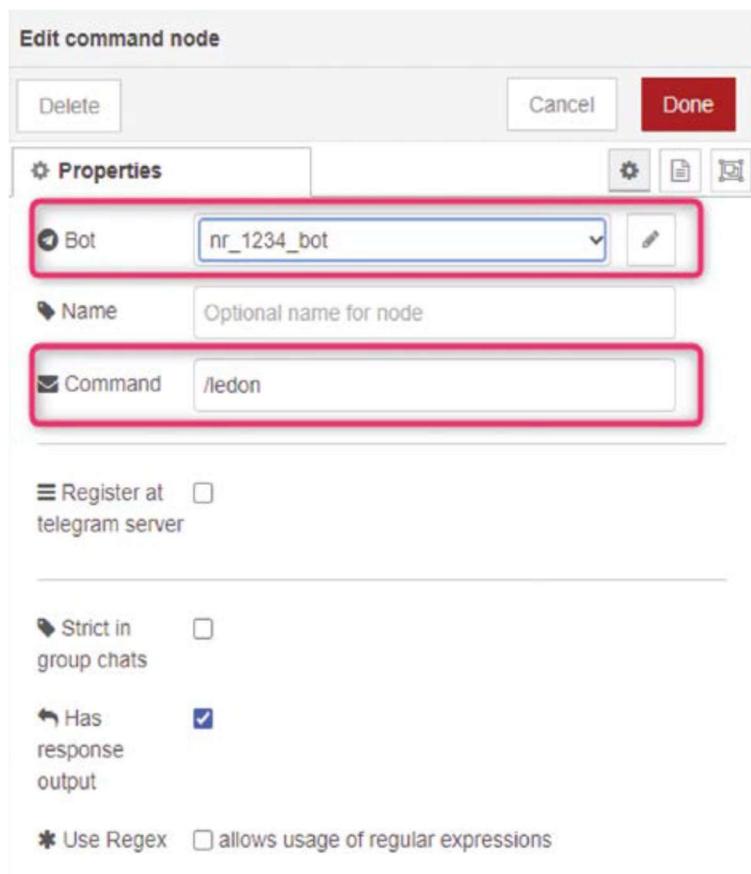


Figura 7-14 - Finestra delle proprietà del nodo Telegram command per l'implementazione del comando di accensione.

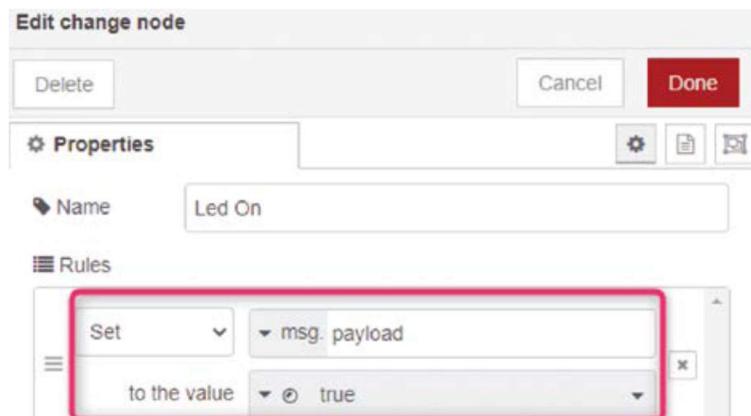


Figura 7-15 - Finestra delle proprietà del nodo change per la modifica del valore di msg.payload.

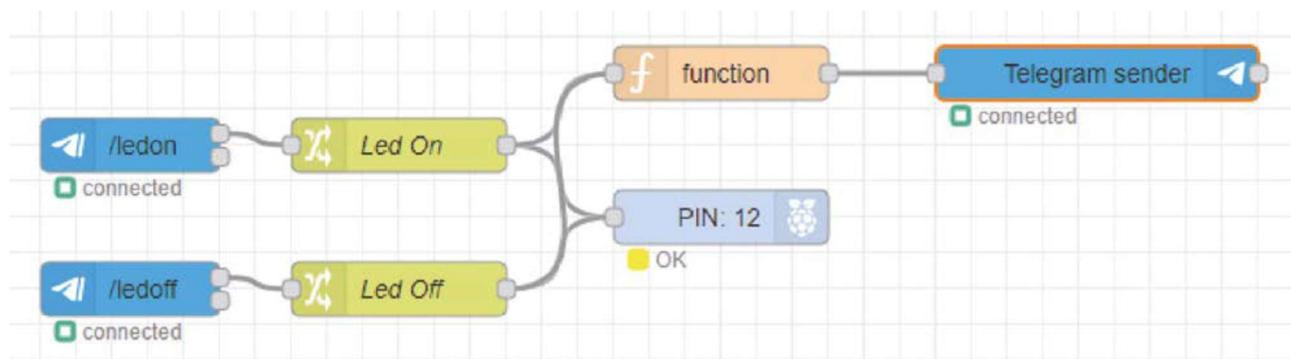


Figura 7-16 - Inserimento della risposta.

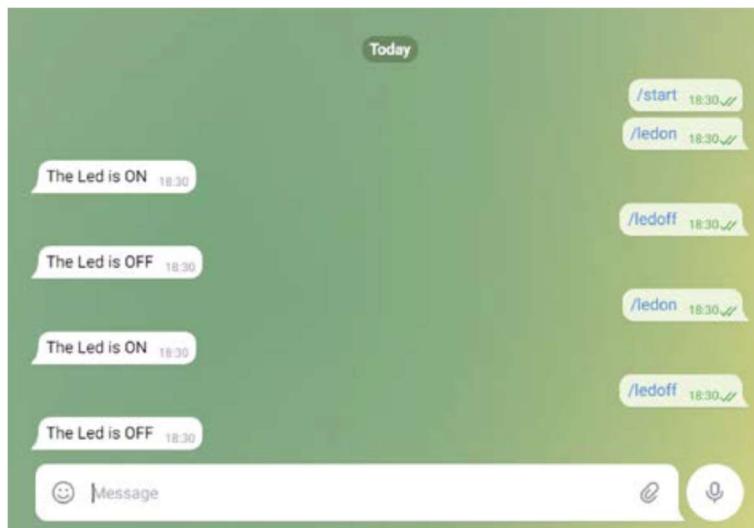


Figura 7-17 - Test finale del Flow.

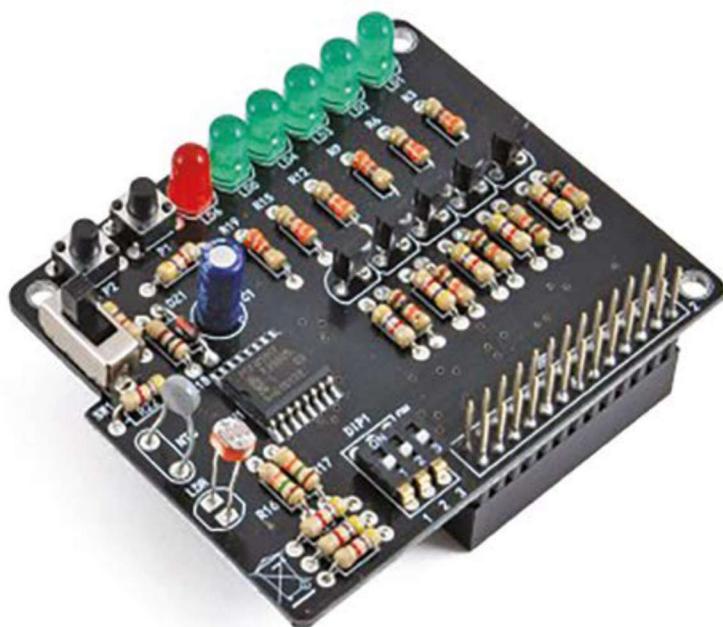


Figura 7-18 - Shield Raspberry Pi I/O di Futura Elettronica.

8. Ultima Puntata.

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 parliamo dell'integrazione degli assistenti vocali con Node-RED.

Come abbiamo già visto ampiamente nella puntata precedente, Node-RED è facilmente interfacciabile con strumenti di utilizzo quotidiano, come ad esempio, per citarne solo alcuni, email, strumenti social ed applicazioni di messaggistica istantanea. Un altro oggetto con cui possiamo interfacciare agevolmente Node-RED sono gli assistenti vocali, un segmento di mercato di oggetti IoT che ha avuto una rapidissima crescita negli ultimi anni. In Italia parliamo di un valore di circa mezzo miliardo di euro, destinato ad aumentare vertiginosamente dal momento che gli assistenti vocali sono un enabler per un settore più ampio che possiamo genericamente definire come Smart Home, che vede la connessione di vari sensori ed attuatori all'interno delle nostre case per abilitare una gestione sempre più Smart dei dispositivi casalinghi.

Esistono ormai diversi assistenti vocali disponibili sul mercato, sostanzialmente molto simili tra di loro, sebbene ognuno con le sue peculiarità; i più comuni sono sicuramente gli assistenti vocali dei colossi Amazon e Google, ossia Alexa e Google Home. In questa puntata noi ci soffermeremo principalmente sui dispositivi della famiglia Alexa, tralasciando il caso di Google Home, che è comunque supportato da Node-RED.

INTEGRAZIONE DI ALEXA IN NODE-RED

Esistono diversi nodi che possono essere utilizzati per integrare Alexa in Node-RED, e tutti questi nodi possono essere divisi sostanzialmente in due categorie:

- Nodi che permettono di emulare dispositivi controllabili via Alexa in Node-RED
- Nodi che permettono di interagire con le API di Alexa

I primi consentono di sfruttare le feature di controllo vocale di Alexa per controllare dei flussi. Si tratta essenzialmente di nodi di input, che, a fronte di una determinata configurazione, permettono di triggerare dei flussi tramite controllo vocale. Questa famiglia di nodi è perfetta per realizzare applicazioni domotiche in quanto permette di controllare luci, relè, interruttori, tapparelle elettriche e via dicendo. Il secondo gruppo di nodi permette invece di interagire con la moltitudine di APIs messe a disposizione dall'ecosistema Alexa, quindi sono più orientati alla gestione dei dispositivi Amazon da Node-RED. Con questa famiglia di nodi è possibile quindi controllare Alexa come text-to-speech device (facendo quindi riprodurre vocalmente una stringa), o controllare la riproduzione di file musicali, oppure ancora inviare notifiche push ad un dispositivo. La lista è decisamente non esaustiva, dal momento che le operazioni eseguibili sono davvero tante, quindi non perdiamo tempo e passiamo alla descrizione di una libreria del primo gruppo, che ci consentirà di controllare tramite comandi vocali Node-RED e, di conseguenza, le interfacce hardware del dispositivo su cui lo stiamo facendo girare, ossia nel nostro caso, la nostra fida Raspberry Pi.

NODE-RED COME ALEXA DEVICE

Come accennato precedentemente, esiste una famiglia di nodi che permette sostanzialmente di emulare dispositivi Alexa tramite Node-RED. Esistono diverse librerie in questa famiglia, ognuna con le sue caratteristiche peculiari ed i suoi pregi e difetti; noi ne consigliamo una che riteniamo decisamente valida e che si basa su una Alexa skill e su un servizio di Bridging per garantire l'interfacciamento.

Il nome della libreria è **node-red-contrib-alexahome-skill** (realizzata da Ben Hardill), e può essere installata, come di consueto, cercandone il nome tramite la l'opzione "manage palette" di Node-RED.

Affinchè la libreria funzioni correttamente è necessario eseguire i seguenti passi:

1. Installare la libreria in Node-RED
2. Registrarsi al servizio di bridge con la skill Alexa
3. Aggiungere le descrizioni dei dispositivi che si desidera implementare
4. Installare la skill Node-RED sull'app Alexa

Iniziamo quindi installando la libreria, che è composta da due nodi più un configuration node (vedi Figura 8-2).

Descriviamo brevemente i nodi che compongono la libreria:

- **alexa-home**: questo è il nodo base che permette di intercettare un comando vocale per uno specifico dispositivo. I dispositivi devono essere preventivamente configurati tramite il servizio di bridge con la skill, prima di poter essere utilizzati. La configurazione contiene, oltre al nome del dispositivo stesso, anche le sue proprietà, tra cui ad esempio la tipologia (lampada, Interruttore, tapparella, etc..) ed i comandi vocali ai quali il dispositivo può obbedire.
- **alexa-home-resp**: questo nodo permette di inviare una risposta personalizzata in merito all'esecuzione di una determinata operazione (ad esempio per confermare che l'attuazione di una luce è andata a buon fine). Non ci soffermeremo molto su questo nodo, in quanto le medesime operazioni possono essere eseguite in maniera più semplice e completa, tramite alcuni nodi appartenenti al secondo gruppo.
- **alexa-home-conf**: questo è il classico nodo di configurazione che permette di collegare un determinato account del servizio di bridge ad una specifica istanza di Node-RED, e quindi permettere di accedere ai dispositivi configurati per quello specifico account.

Una volta installata la libreria, prima di iniziare ad utilizzarla sarà necessario registrarsi al seguente sito web: <https://alexa-node-red.bm.hardill.me.uk/>, la cui home page è rappresentata in Figura 8-3. Una volta registrati sul sito possiamo iniziare ad aggiungere i dispositivi che desideriamo emulare tramite Node-RED. Come primo semplice esempio possiamo aggiungere un LED della nostra Raspberry Pi, che poi andremo ad attivare tramite un comando vocale. Spostiamoci quindi nella sezione "Devices" del sito a cui ci siamo appena registrati e clicchiamo sul pulsante "Add Device" per configurare un nuovo dispositivo. Verrà aperta una semplice maschera di configurazione che ci permetterà di configurare il nostro device. Trattandosi di un semplicissimo esempio, possiamo inserire la configurazione riportata in Figura 8-4.

Al termine dell'operazione, se tutto è andato a buon fine, il dispositivo comparirà nella lista dei dispositivi, come riportato in Figura 8-5. A questo punto, per far riconoscere i dispositivi ad Alexa, è necessario installare una skill nell'app Alexa (o tramite il vostro account Amazon) e collegare l'account del servizio di bridging in cui abbiamo configurato il nostro dispositivo. La skill da installare si chiama Node-RED (di Ben Hardill), ed una immagine della pagina descrittiva della skill è visibile in Figura 8-6.

La skill si può facilmente trovare cercando la voce "Node-RED" dall'app Alexa o dal vostro account Amazon. Una volta installata la skill va attivata e, dopo l'attivazione, verranno richieste le credenziali che avete inserito al momento della registrazione al servizio. A questo punto il dispositivo che è stato precedentemente aggiunto può essere riconosciuto da Alexa. Possiamo utilizzare il comando vocale "Alexa, trova nuovi dispositivi" per consentire l'associazione. Dopo qualche istante Alexa dovrebbe confermare che un dispositivo denominato LED è stato trovato ed associato.

Possiamo tornare su Node-RED ed andare avanti con la creazione di un semplice flow per testare il nuovo dispositivo. Nella palette cerchiamo i nodi della categoria Alexa ed aggiungiamo un nodo alexa-home, oltre ad un nodo rpi-gpio-out, come illustrato in Figura 8-7.

Configuriamo, come di consueto, il nodo rpi-gpio-out per controllare un GPIO, e impostiamolo per controllare il GPIO 18, che, usando lo Shield Raspberry I/O (7100-FT1060M), è collegato al primo LED.

Passiamo ora alla configurazione del nodo alexa-home, facendo un doppio click sul nodo stesso per accedere alla finestra delle proprietà. Qui clicchiamo sull'icona a forma di matita per accedere alla configuration node alexa-home-conf, come illustrato in Figura 8-8.

Il configuration node richiede semplicemente di inserire le credenziali dell'account creato in precedenza, quindi eseguiamo e clicchiamo su "Update". Una volta tornati sulla finestra delle proprietà del nodo alexa-home clicchiamo su "Done" ed eseguiamo un deploy. A questo punto, apprendo nuovamente la finestra delle proprietà, dovremmo poter vedere i nostri dispositivi (in questo caso, se non ne sono stati aggiunti altri, il dispositivo LED) sul drop-down control "Device". Selezioniamo LED, mettiamo il segno di spunta su Auto Acknowledge e clicchiamo nuovamente su "Done", come illustrato in Figura 8-9.

A questo punto possiamo collegare con un wire l'uscita del nodo alexa-home e l'ingresso del nodo rpi-gpio-out (Figura 8-10) e testare il nostro flow.

Se tutto è stato fatto correttamente, usando il comando vocale "Alexa, accendi LED" si dovrebbe accendere il LED sullo shield, mentre con il comando "Alexa, spegni LED" il LED dovrebbe spegnersi. Dal momento che è stata utilizzata l'opzione "Auto Acknowledge", Alexa fornirà automaticamente un feedback sul risultato dell'operazione ("OK" se tutto è andato per il verso giusto o un qualche tipo di errore nel caso contrario). Come avrete notato non è necessario nessun tipo di adattamento del messaggio di uscita in questo caso, in quanto il nodo alexa-home restituisce un boolean su msg.payload che riporta lo stato booleano associato al comando vocale impartito (true nel caso di "accendi LED" e false nel caso di "Spegni LED"). Naturalmente nel caso di attuazioni più complesse rispetto ad un semplice LED potrebbe essere necessario fare qualche adattamento al messaggio in uscita da alexa-home, ma sostanzialmente partendo da questa semplice base è possibile controllare (almeno con dei comandi di base) virtualmente qualsiasi dispositivo interfacciabile con Node-RED.

INTERFACCIAMENTO CON ALEXA API

Ora che abbiamo sperimentato con il primo gruppo di nodi, passiamo ad analizzare il secondo, ossia i nodi che ci consentono di interfacciarcici con le API di Alexa. In questa famiglia di nodi, il rappresentante più noto è sicuramente la libreria node-red-contrib-alexaremote2-applestrudel, che ha subito vari fork, passando da essere chiamata alexa-remote and alexa-remote2, cakebed ed infine alexa-remote2-applestrudel, che è la versione che vi raccomandiamo di installare. Una volta installata questa libreria mette a disposizione un ampio set di nodi, riportati in Figura 8-11. Come di consueto facciamo una rapida carrellata, per poi passare a qualche dettaglio sulla configurazione e ad un esempio pratico.

- **alexa-remote-init**: nodo utilizzato per effettuare il controllo manuale sull'inizializzazione. Non è strettamente necessario utilizzarlo in quanto esiste una opzione di auto-init nel configuration node.
- **alexa-remote-event**: nodo che permette di intercettare eventi, come errori, modifiche ad alcuni settaggi come il volume ed altro ancora. Non tutti gli eventi vengono però notificati da questo nodo.
- **alexa-remote-smarthome**: nodo per il controllo dei dispositivi di tipo casa intelligente. Questo nodo permette sia di controllare dispositivi standard, come lampadine intelligenti e similari, sia i dispositivi emulati da Node-RED (come il LED dell'esempio precedente).
- **alexa-remote-routine**: questo nodo permette di triggerare vari tipi di routine, tra cui:
 - **Speak**: riproduce una stringa di testo.
 - **Speak at volume**: riproduce una stringa di testo ad uno specifico volume.
 - **Text Command**: permette di inviare tramite una stringa un comando vocale.
 - **Wait, Stop**: utilizzati per il controllo di esecuzione nelle sequenze.

- **Promt, Phrase:** permette di far eseguire ad Alexa propt o frasi standard (ad esempio fornire notizie sul tempo o sul traffico in tempo reale o utilizzare formule di saluto standard).
- **Sound:** riproduce un suono (ad esempio un allarme o il verso di un animale).
- **Volume:** permette di controllare il volume.
- **Music:** Permette di riprodurre musica.
- **SmartHome:** permette di accedere al controllo dei dispositivi per la casa intelligente (sostanzialmente la stessa funzione del nodo alexa-remote-smarthome).
- **Launch Skill:** permette di lanciare una skill di Alexa (naturalmente la skill dev'essere installata).
- **Routine:** permette di eseguire una routine predefinta in Alexa.
- **Push Notification:** permette di inviare una notifica push al dispositivo associato all'account.
- **Node:** permette di implementare una sequenza complessa contenente, in numero ed ordine arbitrario, le funzioni elencate in precedenza.
- **alexa-remote-echo:** questo nodo permette di gestire alcune funzioni fondamentali dei dispositivi, sia amministrative che non (come ad esempio rinominare o cancellare un dispositivo o gestire il pairing o la connessione di dispositivi bluethooth)
- **alexa-remote-other:** questo nodo fornisce un'interfaccia con tutte le altre funzioni (ad esempio varie funzioni per il reperimento di informazioni o la gestione delle notifiche).
- **alexa-remote-list:** questo nodo fornisce un'interfaccia dedicata per la gestione delle liste.
- **alexa-remote-account:** questo è il configuration node che permette di interfacciare il nostro account Amazon con Node-RED ed impostare alcune configurazioni di base.

Ora che abbiamo dato uno sguardo ai vari nodi messi a disposizione, passiamo alla configurazione e ad un semplice esempio. Per cominciare trasciniamo sul workspace un nodo “Routine” e facciamo doppio click per accedere alla finestra delle proprietà. Come di consueto, la prima cosa da fare sarà impostare le proprietà del configuration node, quindi clicchiamo sull'icona a forma di matita (**Figura 8-12**).

Qui impostiamo:

- **Nome:** il nome che preferite per la vostra configurazione (nell'esempio MyNrConfig).
- Auth method: Proxy
- **This IP:** inserite l'IP locale della Raspberry sulla quale state facendo girare Node-RED (nell'esempio 192.168.1.10)
- **Port:** lasciate pure 3456
- **File Path:** qui va indicato il path di un file di testo locale nel quale verranno salvate le credenziali per gli accessi successivi
- Refresh: 0
- **Service Host:** alexa.amazon.it
- **Page:** amazon.it
- **Language:** it-IT

Per il resto lasciate sempre i valori di default. Salvate ed eseguite un deploy. A questo punto dovreste notare che sotto al nodo routine che avevate trascinato sul workspace compare il seguente messaggio: “open 192.168.1.10:3456 in your browser” (**Figura 8-13**). Naturalmente l'indirizzo IP sarà quello che avete inserito in fase di configurazione, ossia quello della vostra Raspberry nella vostra rete locale.

Se aprite il browser all'indirizzo indicato dovreste trovare una pagina di autenticazione sulla quale dovrete inserire le credenziali del vostro account amazon.

Se tutto viene eseguito correttamente dovreste visualizzare un messaggio sul browser che conferma il successo dell'operazione e visualizzare la scritta "ready" sotto il nodo routine. A questo punto i nodi alexa remote sono operativi e possono essere utilizzati.

Per testarli possiamo utilizzare lo stesso esempio utilizzato in precedenza, completandolo con una conferma customizzata alle operazioni di accensione e spegnimento del LED. Per farlo trasciniamo sul workspace un altro nodo "Alexa Routine" ed un nodo Switch.

Configuriamo il nodo switch con due regole, rispettivamente "is true" ed "is false" (vedi **Figura 8-14**), per creare due output alternativi che vengono triggerati in funzione dell'operazione richiesta (ricordate che il nodo alexa-home ritorna appunto un boolean come risultato della richiesta di accensione o spegnimento del LED). Ora configuriamo i due nodi Alexa routine per riprodurre sul dispositivo che vogliamo rispettivamente le due frasi "LED Acceso" e "LED spento" e colleghiamoli ai due rami "is true" ed "is false" del nodo switch. Ovviamente il nodo che riproduce la frase "LED acceso" sul ramo "is true" e l'altro su "is false". Infine colleghiamo l'uscita del nodo alexa-home all'ingresso del nodo switch. Infine rimuoviamo la spunta sulla checkbox "Auto Acknowledge" del nodo alexa-home. In Figura 8-15 è riportata la finestra delle proprietà di uno dei nodi Alexa Routine ed in Figura 8-16 un'immagine del flusso completo. A questo punto se ritestate il vostro flow noterete che le risposte di Alexa sono cambiate, infatti invece di rispondere sempre "OK" (funzione di auto acknowledge, che abbiamo disabilitato), risponde con le frasi "LED acceso" e "LED spento" a conferma delle due operazioni. Come avrete potuto facilmente notare, quelli che abbiamo presentato sono solo due semplicissimi esempi delle potenzialità derivanti dalla combinazione di Node-RED ed Alexa, lasciamo ai lettori più interessati ed intraprendenti l'esplorazione delle varie funzioni aggiuntive.

CONCLUSIONI

Siamo arrivati alla conclusione del corso Node-RED.

Durante questa avventura, dopo aver gettato le necessarie basi teoriche sulla FBP (Flow Based programming), siamo passati dai primi, semplici, esempi di gestione dell'I/O (GPIO, I2C, UART, 1-wire), ad esempi sempre più complessi, come la connettività (TCP, UDP, http, Websocket ed MQTT), l'interfacciamento con le app di messaggistica (Whatsapp e Telegram) ed infine l'integrazione con gli assistenti vocali. Ci sarebbero molti altri temi, che per questioni di tempo, non possono essere trattati in questo corso, come il Machine Learning e l'AI, l'interfacciamento con i social (Facebook, Twitter) ed altro ancora, insomma Node-RED è uno strumento potente e flessibile, ma allo stesso tempo, alla portata di tutti, quindi non vi resta altro da fare che prendere una Raspberry (se non ne siete già in possesso) ed iniziare a sviluppare i vostri flow e le vostre automazioni.



Figura 8-1 - Assistenti vocali di Amazon.



Figura 8-2 - Nodi della libreria node-red-contrib-alexa-home-skill. 119



Node-RED Alexa Home Skill Bridge

This site is the registration point for using the node-red-contrib-alexa-home-skill Node-RED module.

This node and service allows you to create your own "devices" to work with the Amazon Echo's Home Skill system, giving you voice control over basically anything you can interface with Node-RED. There is support for not only things like Belkin WeMo and Philips Hue, but also a whole range of homebrew devices. You can also use Alexa to kick off sequences of events not just discrete on/off type actions



Figura 8-3 - Home page del servizio di bridge.

Add New Device

Name:
LED

Description:
This device will control a RPI LED.

Actions

On: **Off:**

%: **+%:** **-%:**

°C/F: **+°C/F:** **-°C/F:**

Figura 8-4 - Inserimento di un nuovo dispositivo.

Devices

Name	Description	Actions
LED	This device will control a RPI LED.	
Add Device		

After making any changes remember to ask Alexa to update devices it knows about by saying "Alexa, discover devices"

Figura 8-5 - Lista dei dispositivi. 120



Figura 8-6 - Skill Alexa per l'interfacciamento con il servizio di Bridge.

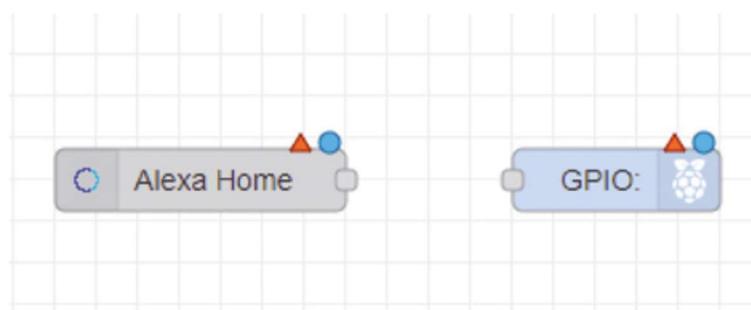


Figura 8-7 - Inserimento di un nodo alexa-home e di un nodo rpi-gpio-out.

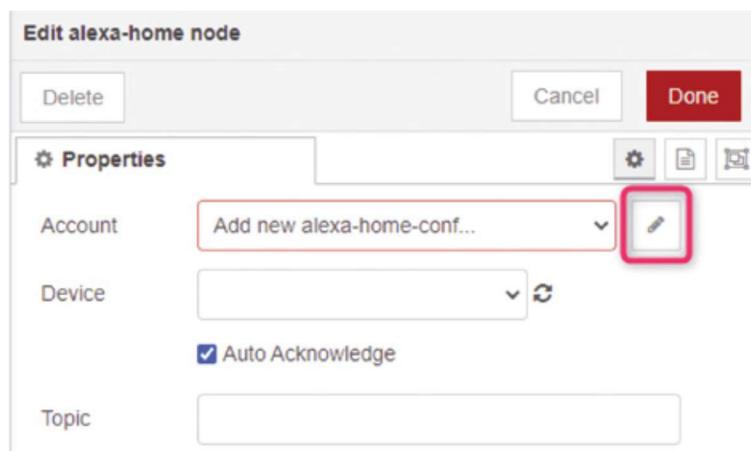


Figura 8-8 - Accesso al configuration node.

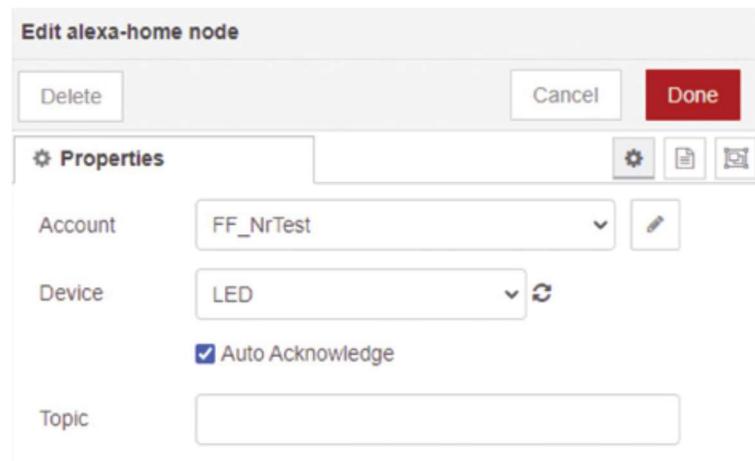


Figura 8-9 - Configurazione del nodo alexa-home.

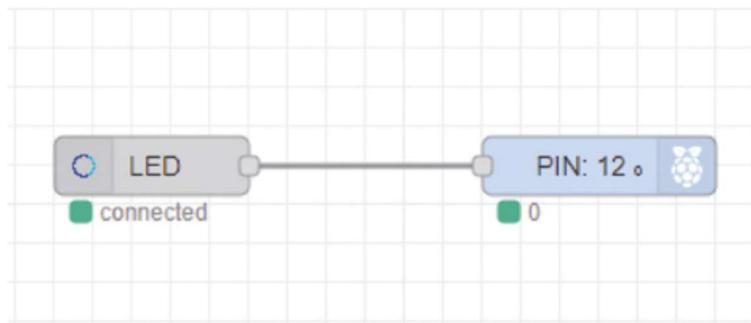


Figura 8-10 - Flow per il controllo di un LED tramite comando vocale con Alexa.

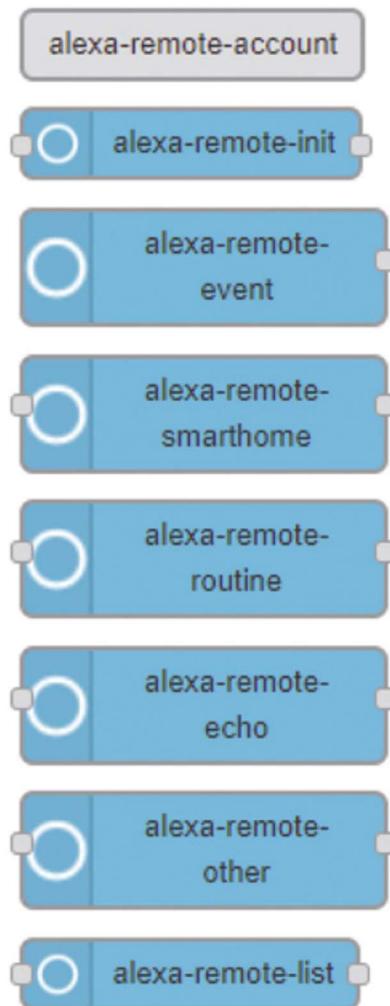


Figura 8-11 - Nodi della libreria `node-red-contrib-alexa-remote2-applestrudel`.

Edit Alexa Account node

Delete Cancel Update

Properties

Name: MyNrConfig

Auth Method: Proxy

This IP: 192.168.1.10

Port: 3456

File Path: /home/pi/alexa.txt

Refresh: 0 days

Auto Init: On

Events: On

Service Host: alexa.amazon.it

Page: amazon.it

Language: it-IT

"On" Keyword: on

User Agent: Optional

Figura 8-12 - Finestra di configurazione del configuration node di alexa remote2 applestrudfel.

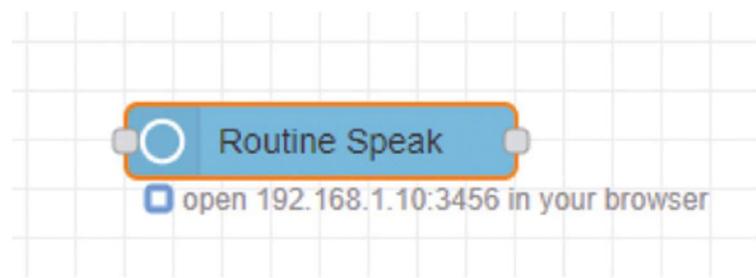


Figura 8-13 - Messaggio con l'indirizzo IP della pagina di autenticazione.

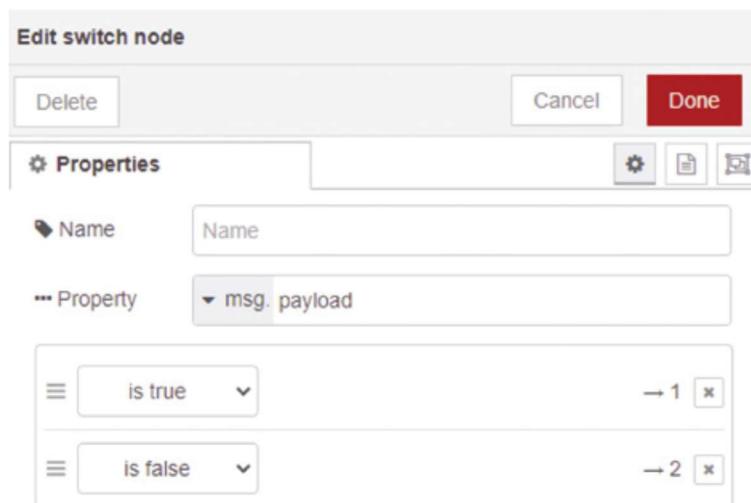


Figura 8-14 - Configurazione del nodo Switch.

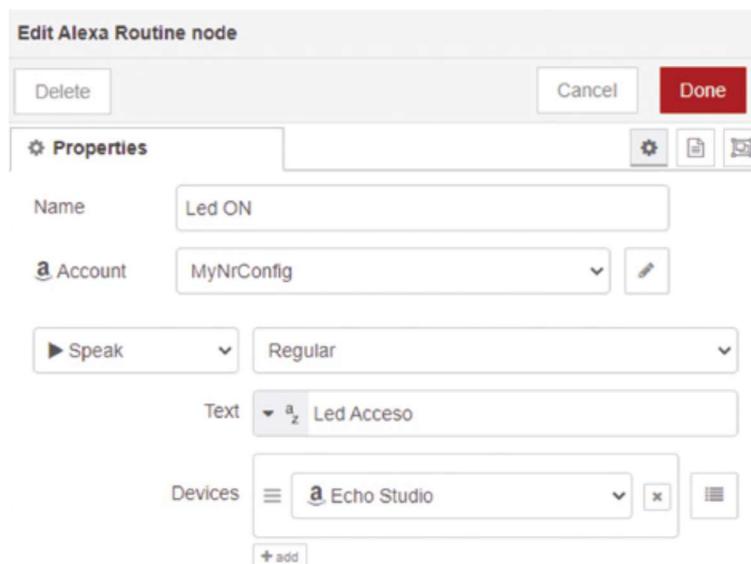


Figura 8-15 - Configurazione del nodo Alexa Routine usato per riprodurre il comando vocale "LED acceso".

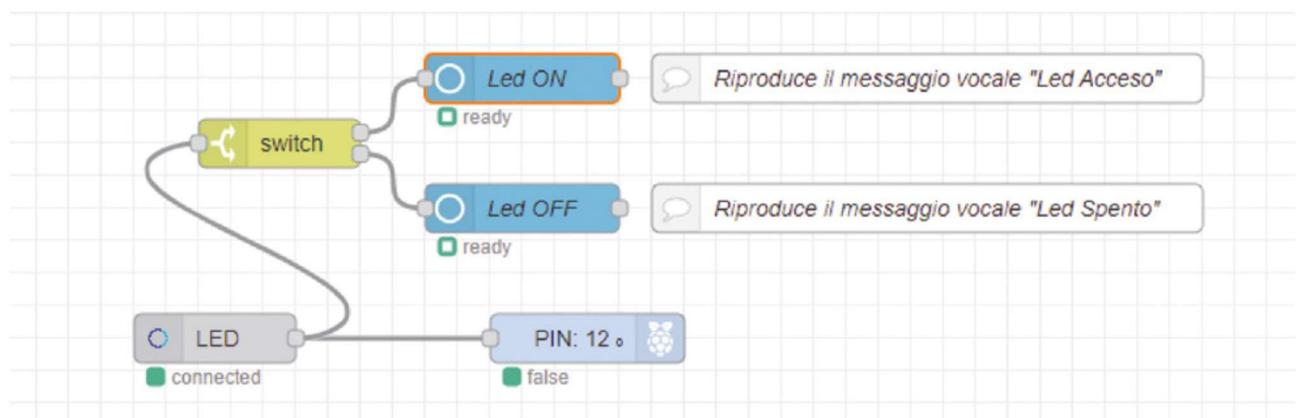


Figura 8-16 - Flusso completo con risposte customizzate.

Node.JS

Node.js è un runtime system open source multiplattforma orientato agli eventi per l'esecuzione di codice JavaScript, costruito sul motore JavaScript V8 di Google Chrome ed originariamente creato da Ryan Dahl nel 2009.

Normalmente gli script JavaScript vengono utilizzati lato client, generalmente incorporati all'interno dell'HTML di una pagina web, venendo interpretati da un engine incorporato direttamente all'interno di un Browser. Node.js consente invece di utilizzare JavaScript anche per scrivere codice da eseguire lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche prima che la pagina venga inviata al Browser dell'utente. Node.js in questo modo permette di implementare il cosiddetto paradigma "JavaScript everywhere" (JavaScript ovunque), unificando lo sviluppo di applicazioni Web intorno ad un unico linguaggio di programmazione (JavaScript).

Node.js ha un'architettura orientata agli eventi che rende possibile l'I/O asincrono. Questo design punta ad ottimizzare il throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output, è inoltre ottimo per applicazioni web real-time (ad esempio programmi di comunicazione in tempo reale o browser game).

Node.js è un progetto di sviluppo open source distribuito gestito dalla Node.js Foundation e facilitato tramite il programma di progetti collaborativi della Linux Foundation.

Le aziende che supportano il programma includono GoDaddy, Groupon, IBM, LinkedIn, Microsoft, Netflix, PayPal, Rakuten, SAP, Voxer, Walmart e Yahoo!

Javascript Object Notation (JSON)

Il formato JSON (JavaScript Object Notation) è un formato di interscambio molto utilizzato in applicazioni client/server. È basato sul linguaggio JavaScript Standard ECMA-262 3^a edizione (dicembre 1999), ma ne è indipendente, trattandosi di un formato completamente indipendente dallo specifico linguaggio di programmazione utilizzato, sebbene le convenzioni che utilizza siano familiari alla famiglia dei linguaggi basati sul C.

La semplicità di JSON ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX.

Il suo uso tramite JavaScript è particolarmente semplice, infatti l'interprete JavaScript è in grado di eseguirne il parsing tramite la funzione `JSON.parse()`. Questo lo ha reso velocemente molto popolare a causa della diffusione della programmazione in JavaScript nel mondo del Web.

La maggior parte dei linguaggi di programmazione possiede un typesystem molto simile a quello definito da JSON per cui sono nati molti progetti che permettono l'utilizzo di JSON con altri linguaggi quali, per esempio: ActionScript, C, C#, Adobe ColdFusion, Common LISP, Delphi, E, Erlang, Java, JavaScript, Lua, ML, Objective Caml, Perl, PHP, Python, REBOL e Ruby. 124

Intervista a Nick O'Leary



Nick O'Leary è uno dei creatori di Node-RED, lo abbiamo raggiunto via mail per fargli qualche domanda sul passato, presente e futuro del tool che ha creato.

Elettronica In: Ciao Nick, come puoi immaginare molti dei nostri lettori sono curiosi riguardo a come lo sviluppo di Node-RED ha avuto inizio. Puoi darci qualche dettaglio a riguardo?

Nick: Certamente. Nel 2013 stavo lavorando presso l'IBM Emerging Technology Group. A quel tempo lavoravo su progetti per diversi clienti, spesso mettendo insieme diversi elementi tecnologici per dare vita ai progetti più disparati. Come parte del ruolo eravamo attivamente incoraggiati a spendere parte del nostro tempo usando nuove tecnologie e mettendo insieme proof-of-concepts di nuove idee. Avendo un background in MQTT ho realizzato una semplice demo di un tool che permetteva di connettere insieme nodi che rappresentavano topics MQTT per mezzo di "wires". Non c'è voluto molto ad aggiungere un pulsante che permetteva di generare il codice in grado di implementare il sistema MQTT progettato. A quel punto un collega che stava per iniziare un nuovo progetto per un cliente mi suggerì che sarebbe stato molto utile avere la possibilità di mappare messaggi MQTT su una porta seriale, e così il gruppo di nodi Serial venne aggiunto al PoC. Da quel punto in avanti ogni nuovo progetto aggiunse un nuovo set di requisiti e un nuovo set di nodi al tool. In breve diventò chiaro che non si trattava più solo di MQTT, ma avevamo tra le mani un tool per la progettazione dei sistemi più disparati. La prima versione di Node-RED era nata a quel punto.

Elettronica In: Avresti mai immaginato che Node-RED potesse avere un simile successo?

Nick: Nei primi mesi siamo stati molto impegnati a consegnare progetti per i clienti, spesso usando Node-RED per realizzarli. Sapevamo di avere qualcosa di utile tra le mani, ma la vera domanda era cosa fare con quel qualcosa. Sin dall'inizio abbiamo deciso che la migliore cosa da fare fosse di renderlo open-source, per far sì che la vasta community degli sviluppatori IoT potesse metterci le mani sopra. Dopo essere passati per il processo interno della compagnia abbiamo ricevuto l'approvazione a renderlo open-source e così il progetto Node-RED su GitHub vide la luce. Non avrei mai potuto immaginare quando la community potesse trovarlo utile e quanto potesse farlo crescere, ma è successo. Continuo a volte a vederlo come il mio piccolo side-project, ma da molto tempo ormai è molto più di quello.

Elettronica In: Quali sono i tuoi programmi adesso? Abbiamo visto che hai lasciato IBM, lo hai fatto perché vuoi dedicarti al 100% a Node-RED nel prossimo futuro?

Nick: Sono stato incredibilmente fortunato a trovare un ruolo in IBM, che mi ha concesso di guidare il progetto di Node-RED e renderlo open-source. Avrei potuto continuare felicemente in quel ruolo, ma all'inizio del 2021 mi è stata data l'opportunità di costruire qualcosa di più grande con Node-RED. Quindi in Aprile ho lasciato IBM ed ho fondato la mia compagnia, FlowForge Inc., che ha lo scopo di costruire una piattaforma attorno a Node-RED, così che possa essere più facile per altre compagnie adottarlo.

Elettronica In: Quali compagnie fanno uso di Node-RED?

Nick: C'è una crescente lista di compagnie che hanno adottato Node-RED. E' molto interessante osservare da quali settori provengono. Molte di loro sono società che lavorano nell'IoT, come ci si potrebbe aspettare, ma ci sono altre compagnie che lo usano e che lavorano in altri settori, come l'healtcare, l'insurance e l'automotive.

Elettronica In: Quali sono le funzionalità più interessanti della release corrente (Node-Red 2.0)? E quali sono le più promettenti feature future che puoi anticiparci?

Nick: Nella release 2.0 ci sono due nuovi tools piuttosto interessanti: un flow debugger ed un Flow Linter.

Il debugger permette di inserire breakpoints nei flows, esattamente come si farebbe con un tradizionale debugger a livello di codice. Il Linter è un analizzatore che controlla il flow usando una serie di regole e permette di evidenziare potenziali problemi nel flow. Siamo ancora agli albori per entrambi questi tools, ma in futuro forniranno agli utilizzatori nuovi superpoteri per aiutarli a creare applicazioni con un elevato livello di qualità. Un nuovo elemento sul quale abbiamo iniziato a lavorare è un Flow Testing Framework. Quello che vorremmo fare è fornire agli sviluppatori uno strumento che gli consenta di creare facilmente test per i loro flow, in modo da garantire che funzionino come ci si aspetta.