SAPIENZA
UNIVERSITÀ DI ROMA

Design e Sviluppo del sistema di End User Development in SeismoCloud

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di laurea in Informatica

Edoardo Ottavianelli
1756005

Relatore
Prof. Emanuele Panizzi

A/A 2019/2020

Sommario

L'oggetto dello studio della tesi è la progettazione e la creazione di un sistema di End User Development per **SeismoCloud**. Uno strumento di EUD (Sviluppo dell'utente finale) consente agli utenti di programmare delle azioni che avverrano in modo automatico alla verifica di determinate condizioni, senza conoscere un linguaggio di programmazione o avere elevata conoscenza informatica.

SeismoCloud è un sistema di Earthquake Early Warning, cioè una rete di sismometri e acceleromentri a basso costo distribuiti su un determinato territorio (in questo caso l'Italia) che consente di avvertire le persone interessate quando è in corso una possibile scossa sismica o addirittura, data l'elevata velocità della rete, prima che il sisma le abbia raggiunte.

Dato che l'End User Development è pensato per un pubblico di non esperti, il lavoro si concentra principalmente sulla facilità d'uso dell'applicativo e sulla minima conoscenza informatica durante la configurazione delle azioni da programmare. Quindi è necessario un lavoro di astrazione dai tecnicismi della materia semplificando in maniera più forte possibile le funzionalità offerte. Questo però non deve contrastare le potenzialità del sistema, gli utenti devono avere **completa libertà nelle scelte** da loro ideate, quindi i più esperti devono comunque riuscire a programmare delle azioni complesse.

Oltre all'esperienza utente viene esaminata la sicurezza garantendo che: i dati che il sistema produce o di cui fa uso non vengano corrotti dall'esterno o utilizzati in maniera impropria; la privacy degli utenti che utilizzano la piattaforma deve essere garantita.

La tesi è suddivisa in quattro macrocapitoli: la **progettazione** del sistema e le scelte effettuate in fase di disegno del progetto; l'**implementazione** delle funzionalità inserite all'interno del sistema concentrata sull'astrazione e sulla semplificazione del processo di automatizzazione dei compiti; uno **studio della sicurezza** del sistema finalizzata alla ricerca ed alla risoluzione, *ove possibile*, di vulnerabilità; una **fase di test** per assicurare l'esito positivo del lavoro e/oppure sottolineare eventuali criticità ancora presenti.

Questo lavoro di tirocinio è stato riassunto in un paper creato insieme al Professore Emanuele Panizzi ed il Dottorando Enrico Bassetti, il quale è stato sottomesso, accettato e presentato al Workshop EMPATHY 2020 (Empowering People in Dealing with Internet of Things).

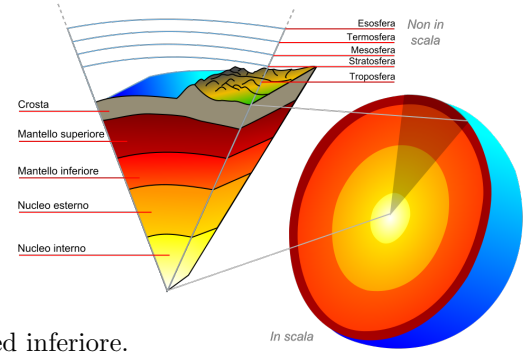
Indice

1. Introduzione a SeismoCloud e obiettivi del progetto

1.1 I terremoti e la loro natura

La terra è composta a strati, o meglio da involucri concentrici (Figura 1.1) ed ognuno di essi ha diverse caratteristiche e particolarità. Al centro della Terra c'è il **nucleo interno**, ossia un ammasso viscoso composto quasi esclusivamente da ferro avente un raggio di circa 1250 km. Si raggiungono temperature molto elevate, circa 5000-6000°C. A seguire abbiamo il **nucleo esterno**, principalmente composto per il 20% da ferro e la restante parte nichel, raggiunge circa i 3000°C. Comprendendo anche il nucleo interno, ha un raggio di circa 3500 km.

Figura 1.1: Composizione della Terra



Continuando verso l'esterno, troviamo il **mantello terrestre**, che si divide in superiore ed inferiore. È composto da diversi metalli ed è difficile stabilire la temperatura dato i moti convettivi del calore, ma si stima intorno ai 500°C a confine con la crosta terrestre e 3000°C a confine con il nucleo. Infine abbiamo la **crosta terrestre**, che partendo dalla superficie, arriva fino a 70 km di profondità. Insieme, il *mantello superiore* e la *crosta terrestre* formano la **litosfera**. La litosfera è suddivisa in una decina di placche tettoniche principali e altre numerose placche di minori dimensioni (figura 1.2). Queste placche "galleggiano" sullo strato immediatamente sottostante del mantello superiore.[1]

Esse, data la forte pressione e le alte temperature, subiscono sforzi di enormi dimensioni che formano i **terremoti**. I terremoti sono vibrazioni della crosta terrestre, provocate dallo spostamento di una o più placche nella litosfera. Le placche si muovono flettendosi lentamente e poi rilasciando (raggiunto il *punto di rottura*) in maniera elastica tutta l'energia accumulata. Il punto in cui viene generata questa energia è detto **ipocentro** (2), zona in cui è presente una frattura chiamata *faglia* (3), mentre il punto in superficie posto sulla verticale dell'ipocentro è chiamato **epicentro** (1).

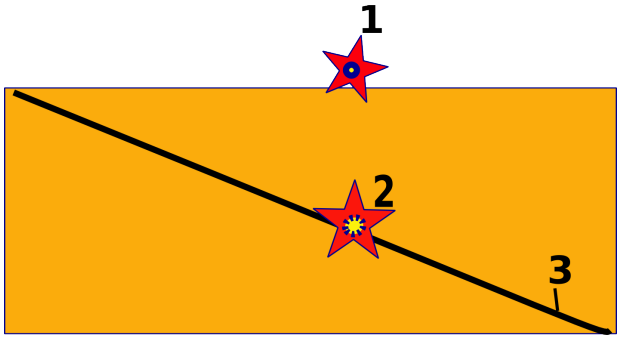
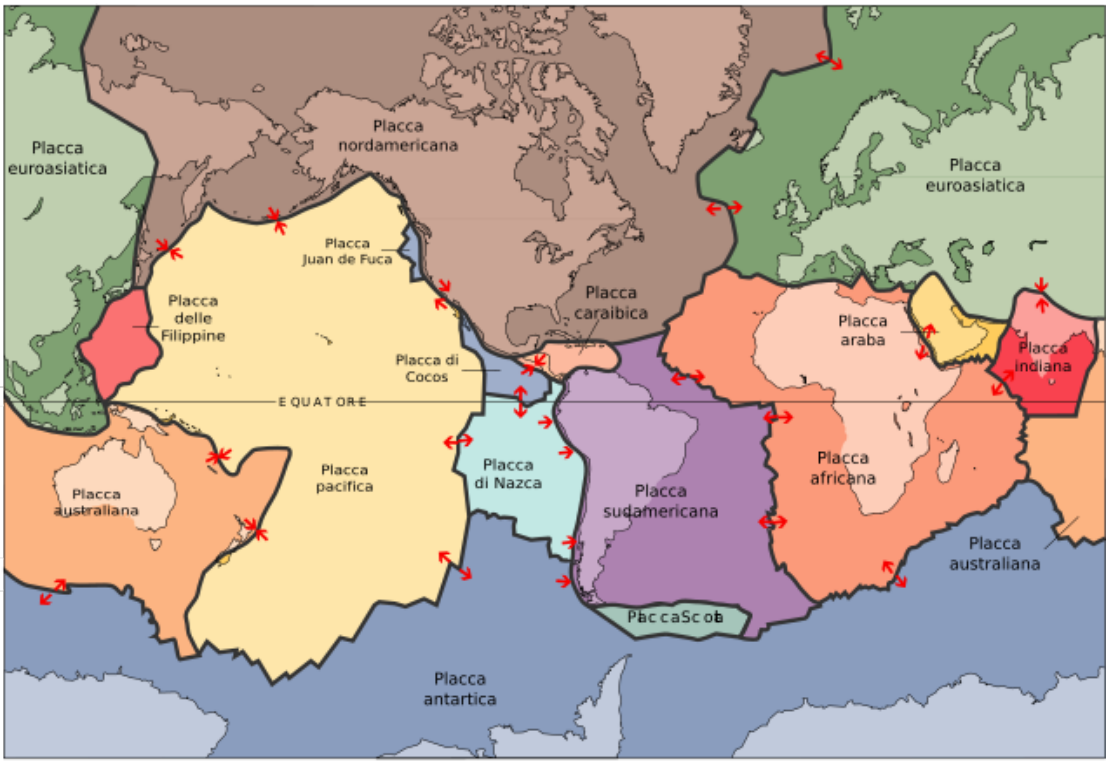


Figura 1.2: Placche tettoniche



Esistono tre tipi di faglie: faglie *trascorrenti*, faglie *dirette* e *inverse*. Esistono in egual numero differenti tipi di onde sismiche.
Le **onde di compressione o longitudinali** fanno comprimere e dilatare la materia nella stessa direzione con cui si propaga l'onda. Sono anche dette *primarie*, perché sono le onde che viaggiano a velocità più elevate.
Le **onde di taglio o trasversali** fanno compiere alla materia oscillazioni in modo perpendicolare alla loro direzione di propagazione. Hanno effetto solo nei solidi, non possono propagarsi attraverso corpi liquidi o gassosi. Vengono chiamate anche *onde secondarie*, essendo meno veloci delle precedenti.
Le **onde superficiali**, anche se il nome può essere mal interpretato, non si manifestano in superficie. Questo tipo di onde sono la combinazione delle due precedenti, perciò sono molto complesse e sono le più pericolose.

1.2 Misurazione dei terremoti

Tipologie di terremoti

Esistono 4 differenti tipologie di terremoti: *tettonici*, *vulcanici*, *da crollo* e *artificiali*.
I terremoti **tettonici**, come dice il nome, sono provocati dal movimento delle placche tettoniche e hanno origine lungo le faglie. Sono i più pericolosi ed i più frequenti.
I terremoti **vulcanici** sono originati dall'attività vulcanica nel sottosuolo. Sono meno pericolosi dei precedenti data la minor energia rilasciata e l'estensione limitata.
I terremoti **da crollo** si originano durante il crollo di montagne, grotte o frane. Hanno una bassa pericolosità e frequenza.
Infine, i terremoti **artificiali** vengono originati da attività umane. Ad esempio, una grande esplosione. In generale, hanno una potenza molto limitata.

Metodologie di misurazione

Esistono due tipologie principali di misurazione di un terremoto: la scala **Mercalli** e la scala **Richter**. La scala Mercalli misura l'*intensità* di un terremoto osservando i danni causati da esso. Per questo, dato che tiene in conto solo degli effetti che la scossa produce, può essere applicata anche ai terremoti avvenuti nel passato. Assegna dei numeri crescenti per intensità, si va dall'uno (impercettebile) sino a dodici (apocalittica). La scala (o meglio *indice*) Richter misura invece l'energia sprigionata dalla scossa, ossia la *magnitudo*. Viene chiamata scala, ma è più corretto dire indice dato che non ha un range di valori finito. La magnitudo è descritta da questa formula:

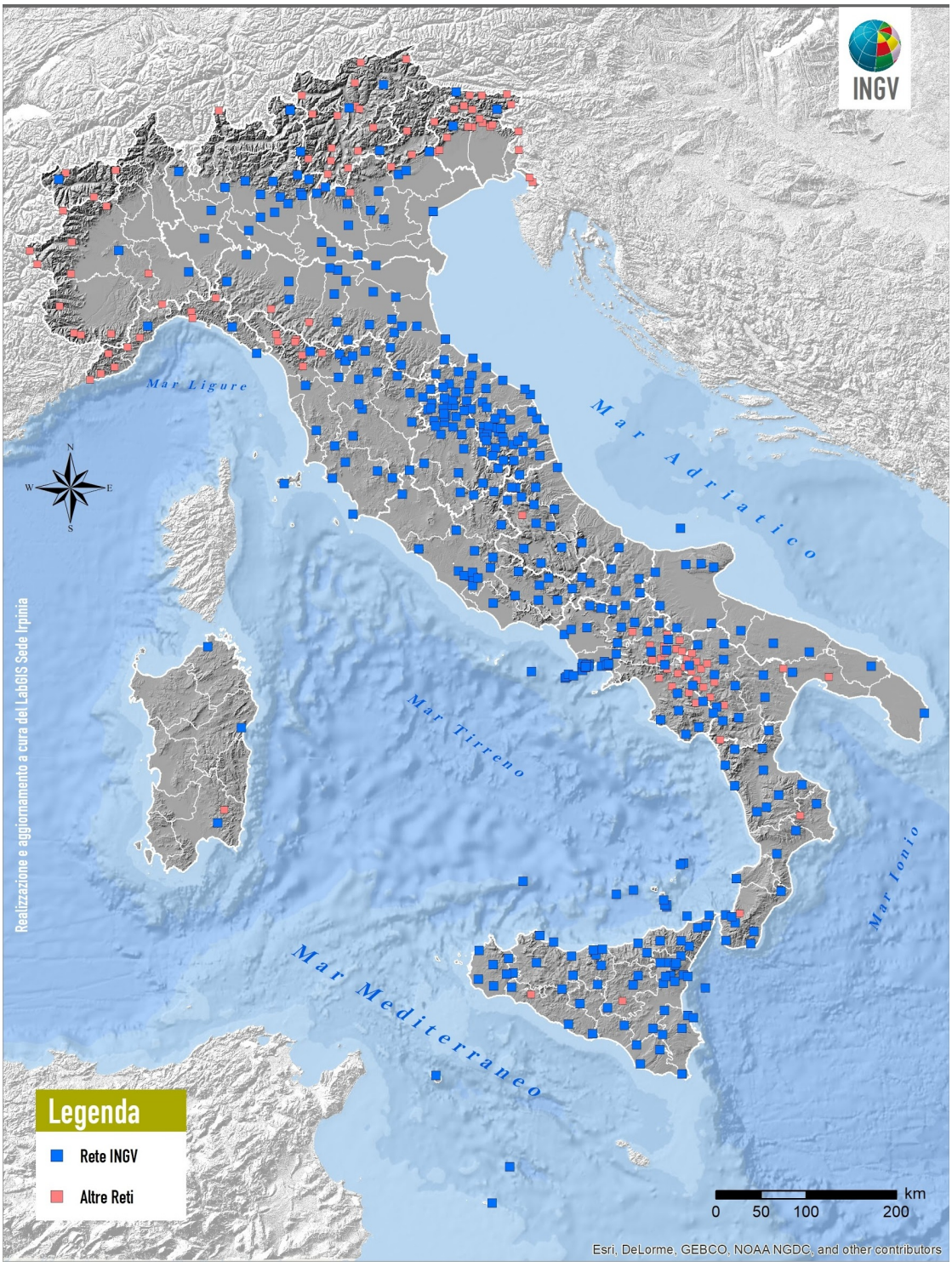
$$M_w = \frac{2}{3}\log_{10}(M_0) - 10,7$$

dove M_0 è il momento sismico all'ipocentro da esprimere in Newton per metro—N·m.[2] Ad oggi il massimo valore magnitudo registrato è 9,5.

Strumenti di misurazione

Gli strumenti di misurazione dei terremoti sono principalmente due: il **sismografo** ed il **sismometro**. Entrambi misurano accelerazione e velocità dei movimenti del suolo, con la differenza che il sismometro misura solamente, ma non può registrare i dati; il sismografo invece oltre alla misura produce anche un grafico temporale, chiamato appunto *sismogramma*. Uno strumento non può misurare ampie porzioni di territorio, per questo si creano delle reti che monitorano degli ampi spazi di suolo. In Italia questa rete è la **Rete Sismica Nazionale**. Con le tecnologie odierne, si utilizzano sismometri digitali insieme ad altri strumenti. Ad esempio, nella rete italiana, le stazioni sono composte da un sismometro, un accelerometro ed una antenna e ricevitore GPS. È una rete di stazioni sismiche (circa 300) disposte su tutto il territorio italiano e appena fuori dai confini. Per la maggior parte sono stazioni dell'INGV (Istituto Nazionale di Geologia e Vulcanologia), ma ne fanno parte anche altre piccole reti. Questa rete monitora sette giorni su sette, 24 ore su 24 i movimenti del suolo e li registra, inviandoli poi ai centri di elaborazione dati di Roma, Grottaminarda e Catania. La concentrazione maggiore di terremoti in Italia è nell'appennino e nel sud Italia, per questo la maggior parte delle stazioni è posizionata in questi punti, come si può notare nella Figura 1.3.

Figura 1.3: Rete Sismica Nazionale [3]



1.3 Piattaforma SeismoCloud

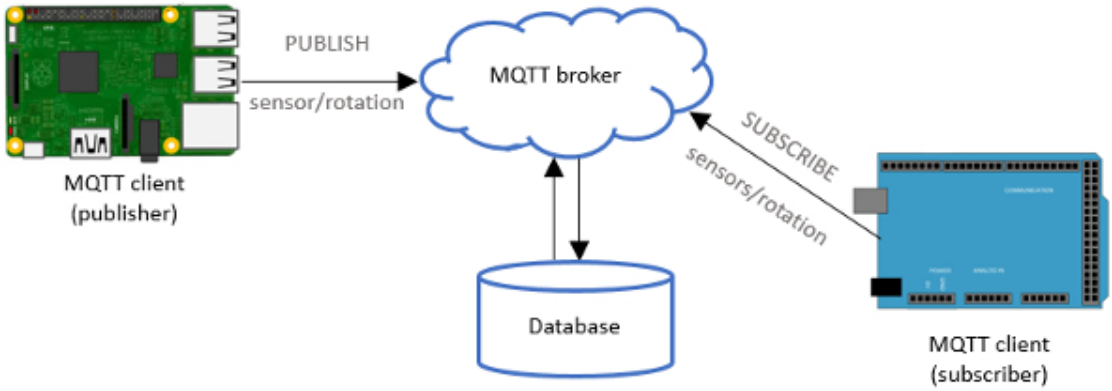
Introduzione al progetto

SeismoCloud è una rete di dispositivi IoT (Internet of Things) a basso costo connessi fra loro distribuiti in tutta Italia. Ha come scopo l'**Earthquake Early Warning**, ossia monitorare l'attività sismica e, nel caso viene rilevato un terremoto, notificare le persone interessate. L'obiettivo specifico dell'EEW è l'avviso **in tempo reale** di un possibile sisma. I sismometri possono essere costruiti personalmente se si ha capacità di elettronica di base, altrimenti viene utilizzato il sensore interno di uno smartphone. Come si può notare dalla Figura 1.4 la **Rete Sismica SeismoCloud** si estende su tutto il territorio nazionale. Si contano circa 100 sismometri attivi. Ad oggi il sistema EEW non è attivo perché i sismometri presenti non bastano a compiere delle rilevazioni sufficienti a stabilire l'effettiva presenza di un sisma. I nodi della rete sono ben distribuiti nel nord e centro Italia, male nel Sud. Questo è un problema perché l'attività sismica principale italiana si sviluppa lungo la catena appenninica fino in Sicilia. Grande menzione meritano anche i vulcani attivi (Etna, Stromboli, Vesuvio, Vulcano). È un progetto nato dall'Università La Sapienza e l'INGV. Viene presentato anche nelle scuole per sensibilizzare gli studenti sul tema terremoti ed utilizzare questo progetto come strumento didattico. Il codice di alcune porzioni di sistema è disponibile su GitHub sotto l'organizzazione SapienzaApps.

Figura 1.4: Rete Sismica SeismoCloud

Architettura di sistema

I sismometri sono di due tipologie: *fissi* e *mobili*. I **sismometri fissi** sono dei moduli formati da componenti elettronici a basso costo. Viene usato un chip con un modulo Wi-Fi integrato per la connessione ad Internet, nello specifico il NodeMCU/ESP8266 e un modulo con giroscopio e accelerometro, l'MPU6050. Sono pensati per essere fissati a muri portanti per una migliore rilevazione. I **sismometri mobili** invece utilizzano l'applicazione per smartphone, abilitando l'accelerometro interno del telefono per rilevare vibrazioni. Devono essere appoggiati su un piano orizzontale, per esempio un tavolo, per avere una rilevazione precisa. Durante la registrazione di un nuovo sismometro fisso esso viene localizzato per conoscere la sua posizione (fondamentale al rilevamento corretto dei sismi), mentre i sismometri mobili aggiornano di continuano la loro posizione. Nonostante ciò la privacy può essere conservata mostrando il sismometro nella mappa pubblica con una finta posizione casuale nel raggio di 2km. I sismometri utilizzano il protocollo ISO standard **MQTT** (Message Queue Telemetry Transport) per inviare e ricevere messaggi di controllo e dati.



Viene utilizzato questo protocollo perché è stato progettato per i dispositivi IoT, dato che questi ultimi hanno, solitamente, una batteria con piccola capienza ed in generale risorse limitate. Rispetto al precedente protocollo HTTP, esso invia messaggi ad alta affidabilità, ha una bassa latenza ed un basso overhead[4]. In MQTT, il `topic` è una stringa UTF-8 che un broker utilizza per filtrare i messaggi. Si può considerare come una etichetta che viene affissa a dei valori che si riferiscono tutti allo stesso argomento. Un broker è il responsabile della comunicazione tra i vari client connessi. I topic hanno una gerarchia basata a livelli ed ogni livello è separato da uno slash. Ad esempio il topic che si riferisce alla temperatura del sismometro con identificativo 34 potrebbe essere `sensor/34/temperature`. Dato che MQTT è un protocollo di tipo *publish/subscribe*, i client possono pubblicare dati relativi ad un topic, i quali verranno ricevuti dai client che sono iscritti a quel determinato topic. I client sono liberi di scegliere le azioni da compiere: possono solamente iscriversi ad alcuni topic (o a tutti), oppure solamente pubblicare dati su topic senza iscriversi, oppure possono eseguire entrambe le operazioni. I dati inviati vengono poi elaborati in un server centrale che li gestisce, compie dei calcoli su di essi (allo scopo di identificare una possibile scossa di terremoto e attivare una notifica di Earthquake Early Warning) e li archivia in un database.

Quando gli utenti richiedono i dati (da app o interfaccia web) non viene utilizzato il protocollo MQTT, ma il protocollo HTTP (Hyper Text Transfer Protocol). Per i messaggi inviati tramite HTTP viene utilizzato lo stile architetturale **REST** (REpresentational State Transfer). Questo schema è definito da dei principi [6]:

- `Client-Server`: Separare l'interfaccia dalla logica dei dati aumenta la portabilità del sistema.
- `Stateless`: Ogni richiesta dal client al server non deve basarsi su altri dati se non quelli interni alla richiesta.
- `Cacheable`: I dati, se possibile, devono essere salvati nella cache del client per utilizzi futuri.
- `Interfacce uniformi`: Prediligere identificazione delle risorse; manipolazione delle risorse tramite rappresentazioni; messaggi auto-esplicativi; hypermedia come motore dello stato dell'applicazione.
- `Sistema a livelli`: un sistema composto da più livelli indipendenti rende il sistema modulare.
- `Codice su richiesta (opzionale)`: È permesso scaricare ed eseguire codice (applet o script) per snellire il client.

Il nucleo dello stile REST sono le *risorse*. Una risorsa è qualsiasi informazione: un documento, una immagine, un video, un numero, una collezione di risorse. REST utilizza identificatori di risorse per aiutare la comunicazione tra le parti. Un esempio di risorsa in stile REST è: `example.domain/sensor/sensor-id/quake`, dove `sensor-id` è l'identificatore unico per un sensore. Tramite l'app mobile (disponibile per Android e iOS) si registrano i nuovi sismometri (sia fissi che mobili). Il sismometro interno viene registrato automaticamente quando viene scaricata l'applicazione per la prima volta. Il sismometro fisso invece crea una connessione Wi-Fi chiamata *SeismoCloud*. L'utente ci si collega ed inserisce i dati per connettersi alla rete Wi-Fi locale. Successivamente in automatico il dispositivo fisso si collega alla rete Wi-Fi e l'app cerca di trovare nella rete un sismometro. In entrambi i casi viene richiesto di consentire all'acquisizione della posizione GPS ed un nome per il nuovo sismometro. Attivando un sismometro viene prodotta una mole consistente di dati.

Esempio di dati ricevuti e/o inviati dal dispositivo

```
timereq = 1597913685108
timesync = 1597913685108;1597913683476;15979136857668
publicip = 129.65.2.45
localip = 192.168.0.12
essid = wifi-locale
rssi = -58.744674356
bssid = aa:bb:cc:dd:ee:ff
treshold = 27200000.0000
temperature = 34.100000
quake = 1597913685178;0.644563;0.353425;0.683452
location = 40.857634274356;15.24525254252
```

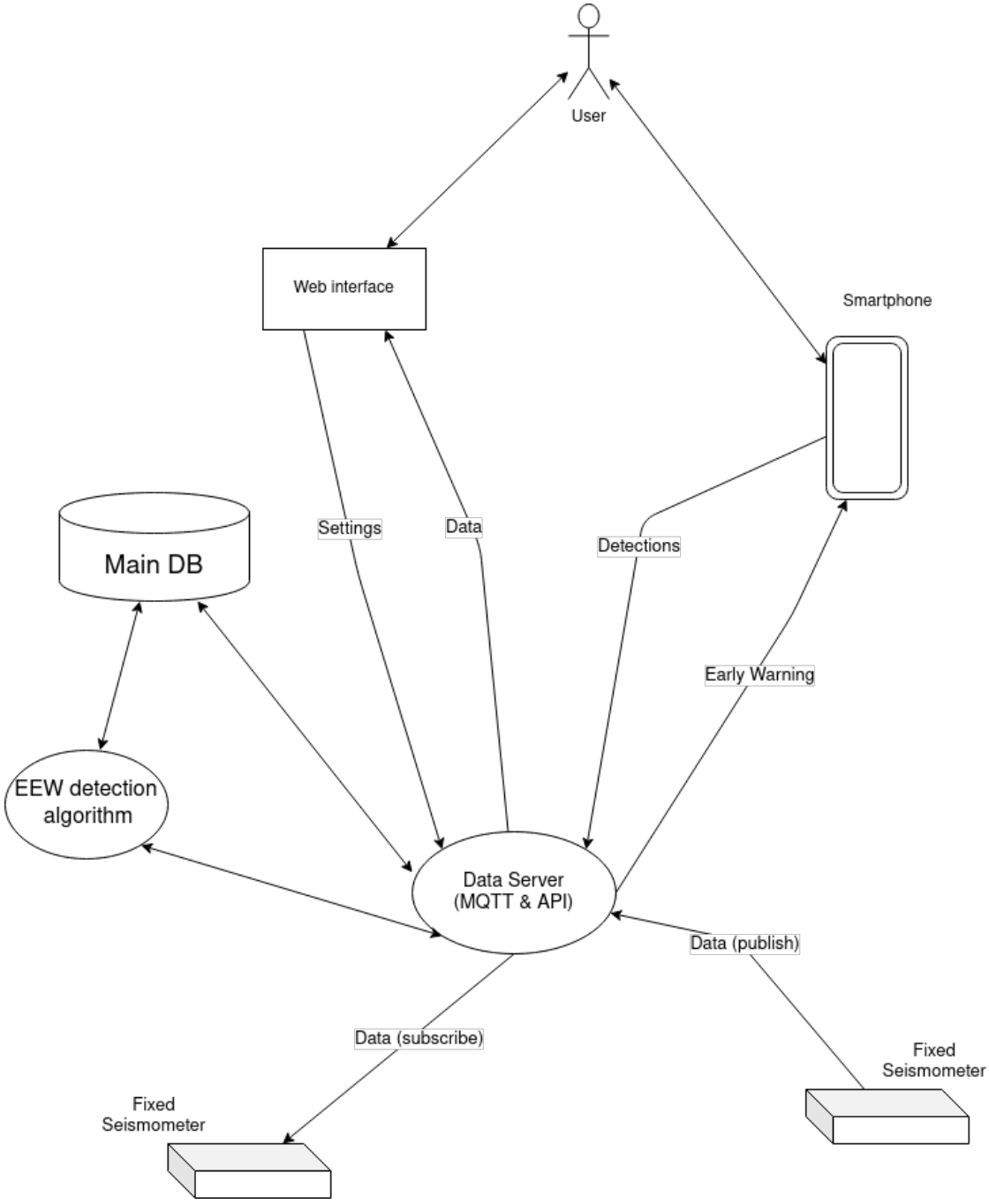
Con una cadenza periodica (qualche minuto, dipende dal carico del sistema) vengono pubblicati: indirizzo IP locale e pubblico, indirizzo MAC e qualità del segnale della rete Wi-Fi, localizzazione e temperatura del sismometro, soglia attuale (valore utile alla rilevazione di una vibrazione). Quando il sismometro rileva una vibrazione viene inviata l'intensità della vibrazione. Un topic MQTT è dedicato al comando *reboot*, ossia la possibilità di riavviare il sismometro pubblicando un messaggio.
Inoltre è possibile visualizzare la lista dei propri sismometri, una mappa aggiornata in tempo reale con i sismometri attivi e i terremoti recenti, una lista di terremoti con relativa magnitudo, posizione e profondità dell'ipocentro (ordinati cronologicamente) e infine alcune informazioni utili. È disponibile anche un'interfaccia web dove è possibile visualizzare tutte le informazioni prima citate ed in più una dashboard personalizzata. In quest'ultima è possibile visualizzare due grafici che mostrano il tempo di utilizzo assoluto e settimanale di ogni sismometro attivo. Dopo l'effettiva autenticazione (tramite password o QRCode) si entra nell'area privata, ossia relativa ad un determinato *gruppo* (insiemi di sismometri raggruppati per locazione). Qui, oltre alle funzionalità citate, è presente un sistema di **EUD** (End User Development). Dalla pagina Wikipedia [7]:

> *"Lo sviluppo per l'utente finale (EUD) o la programmazione per l'utente finale (EUP) si riferisce ad attività e strumenti che consentono agli utenti finali, persone che non sono sviluppatori di software professionisti, di programmare i computer. Le persone che non sono sviluppatori professionisti possono utilizzare gli strumenti EUD per creare o modificare artefatti software (descrizioni di comportamenti automatizzati) e oggetti di dati complessi senza una conoscenza significativa di un linguaggio di programmazione."*

Quindi, queste tipologie di sistemi permettono a utenti finali (diretti utilizzatori del sistema) di collegare dati e servizi in un modo logico e sviluppare funzionalità ad hoc precedentemente non disponibili. Ad esempio, si può ricevere un messaggio automatico tramite Telegram ogni volta che il mio sismometro vibra. Oppure, ricevere una notifica push sul proprio cellulare quando n sismometri rilevano una vibrazione nello stesso momento, o meglio ancora, allertare tutti gli utenti in un gruppo. È facile intuire che queste tipologie di automazione sono molto utili per un utente non esperto, soprattutto nell'ambito SeismoCloud, dove la community è un fattore cardine dell'applicativo.

Questo sistema viene spiegato meglio nel sottocapitolo 3.1 "Sistema Legacy".

Figura 1.5: Architettura di SeismoCloud



Questo lavoro di tirocinio è stato riassunto in un paper creato insieme al Professore Emanuele Panizzi ed il Dottorando Enrico Bassetti, il quale è stato sottomesso, accettato e presentato al Workshop EMPATHY 2020 (Empowering People in Dealing with Internet of Things) [9].

Simplify Node-RED For End User Development in SeismoCloud

Enrico Bassetti[a], Edoardo Ottavianelli[a] and Emanuele Panizzi[a]

[a]*Computer Science department, Sapienza University of Rome, Piazzale Aldo Moro 5, 00185 Roma, Italy*

Abstract

Networks of IoT devices often require configuration and definition of behavior by the final user. Node-RED is a flow-based programming platform commonly used for End User Development, but it requires networking and protocols skills in order to be efficiently used. We add a level of abstraction to Node-RED nodes in order to allow non-skilled users to configure and control networks of IoT devices and online services. We applied such abstractions to the SeismoCloud application for earthquake monitoring.

Keywords

End User Development, EUD, Human Computer Interaction, HCI, Internet of Things, IoT, Node-RED

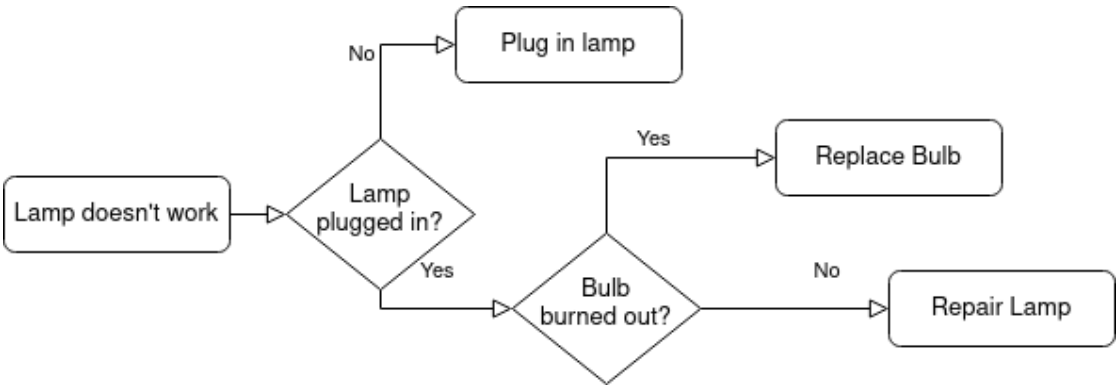
2. Progettazione dell'architettura del sistema EUD

2.1 Scelta dello strumento di sviluppo

Candidati iniziali

Piuttosto che progettare e costruire uno strumento di End User Development da zero, si è cercato uno scheletro già pronto e maturo. La ricerca è stata piuttosto semplice, un po' meno la valutazione dei possibili candidati. Il primo candidato è **IFTTT** (https://ifttt.com). È lo strumento più famoso, utilizzato da molte persone, con più di 5 milioni di download su Play Store. Il vantaggio di questa scelta è sicuramente la grande maturità della piattaforma. Gli svantaggi sono la difficoltà nell'integrare i nostri dispositivi con questo sistema, ma soprattutto la somiglianza con il sistema EUD da rimpiazzare, troppo semplice e con condizioni e azioni predefinite. Una seconda scelta è **draw.io** (https://draw.io). Strumento completamente differente al precedente, è un creatore di diagrammi di flusso. Questo è un tipo di programmazione di azioni che lascia molta libertà all'utente, rimanendo semplice ma efficace.

Figura 2.1: Esempio di flusso esportato da draw.io



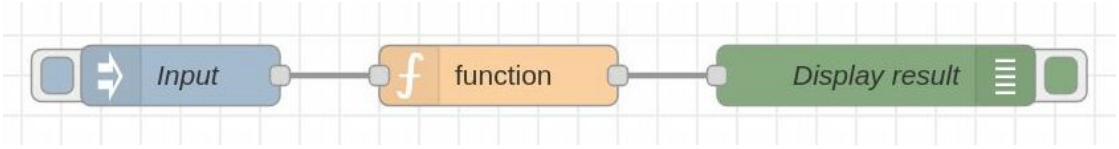
Il codice sorgente di draw.io è disponibile su GitHub. È ben documentato, quindi questo aiuta l'integrazione con SeismoCloud. Purtroppo si tratta di uno strumento front-end, quindi manca tutta la logica back-end; i dati inseriti dall'utente vengono esportati in file di tipo xml che devono poi essere tradotti in azioni registrate dal sistema.

La piattaforma Node-RED

Lo strumento che risolve i problemi precedentemente elencati è **Node-RED** (https://nodered.org). Come draw.io è un creatore di diagrammi di flusso ed è anche open-source con repository su GitHub sotto l'organizzazione node-red. È un servizio orientato all'*Internet of Things*, non è uno standard ufficiale, ma è molto utilizzato nel campo. A differenza del precedente esiste uno scheletro back-end, dato che è costruito utilizzando NodeJS. Vedremo più avanti in dettaglio come è stato progettato. La community, oltre ad essere pronta a risolvere dubbi sul forum ufficiale

(discourse.nodered.org), contribuisce attivamente al progetto implementando delle funzionalità che sono messe a disposizione di tutti tramite il famoso package manager NPM. Nel nome il 'node' è dato sia dal runtime NodeJS, sia dal nome che hanno gli elementi utilizzati per creare i flussi, chiamati appunto *nodi*.

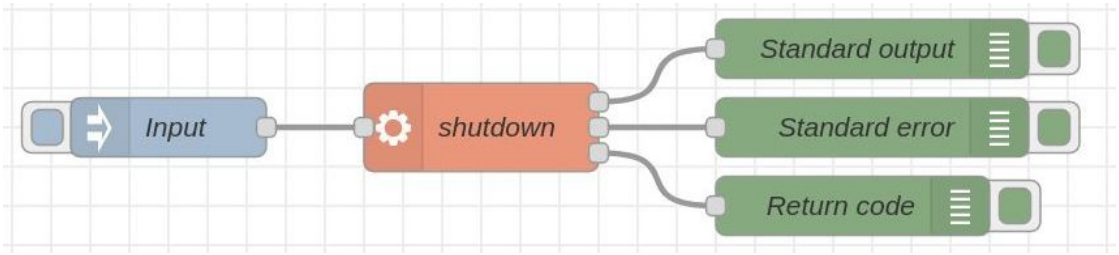
Figura 2.2: Esempio flusso di nodi in Node-RED



Come si nota in figura 2.2, i nodi sono dei rettangoli che hanno funzioni particolari:

- **Nodi input**: Questi nodi emettono valori e non prendono nulla in ingresso, quindi sono utilizzati per produrre dati o iniziare dei flussi. Possono avere da 1 a n valori in uscita.
- **Nodi intermedi**: Sono nodi che hanno sia input che output. Sono utilizzati per applicare delle funzioni ai dati in ingresso, oppure il dato in ingresso viene solamente utilizzato per far iniziare una procedura intermedia. Prendono in input un solo valore e possono avere da 1 a n valori in uscita.
- **Nodi output**: Sono gli opposti dei nodi input. Non hanno valori in uscita ma solo in ingresso. Sono utilizzati per terminare un flusso, solitamente sono quindi le azioni da effettuare nel caso in cui si verificano le condizioni stabilite. Possono avere un solo valore in ingresso.

Figura 2.3: Esecuzione del comando shutdown



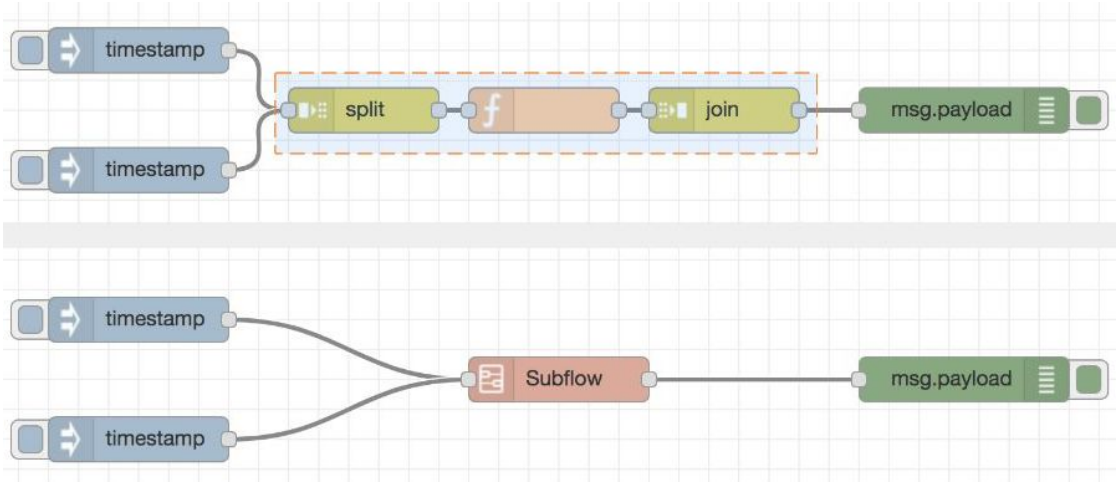
Nella figura 2.3 abbiamo un altro esempio. Qui quando viene cliccato il pulsante a sinistra del nodo *input*, il nodo invia in uscita un valore (in questo caso qualsiasi, ad esempio 3 o la stringa 'input') che fa eseguire il secondo nodo (di tipo *intermedio*). Il nodo *shutdown* esegue il comando 'shutdown' e stampa nella console di debug lo stdout, stderr ed il return code. Data la facilità d'uso, il grande supporto (repository con 10+K star su GitHub, community molto attiva e forum ufficiale), la maturità del progetto (versione 0.2 nel 2013, ad oggi (2020) 1.1.3), la facile estendibilità ed integrazione con dispositivi IoT (come appunto i nostri sismometri), decisamente Node-RED è lo strumento più adatto per risolvere i bisogni della community SeismoCloud.

2.2 Architettura adattata a SeismoCloud

Come funziona Node-RED

Node-RED può essere utilizzato in vari modi, ad esempio: su un computer in locale, tramite Docker, su un dispositivo Raspberry, oppure su alcuni sistemi Cloud (IBM Cloud, Amazon Web Services, Microsoft Azure). L'interazione con l'utente avviene completamente tramite un browser. Viene fornita una pagina divisa in tre sezioni: un insieme di funzionalità (nodi) disponibili, un foglio di lavoro dove poter creare i vari flussi ed un modulo di aiuto, documentazione e debug. Un **nodo** è il blocco costruttivo di base di un flusso. Le funzionalità dei nodi sono scatenate da eventi esterni (richieste HTTP o altro protocollo oppure un timer) o da input ricevuti da altri nodi. Processano il messaggio ricevuto oppure ne creano uno nuovo e possono inviare messaggi ai nodi successivi nel flusso. Un **nodo di configurazione** è un tipo speciale di nodo che setta delle configurazioni utili agli altri nodi, ma non fa parte di nessun flusso. Ad esempio, i nodi *MQTT-in* e *MQTT-out* utilizzano il nodo di configurazione *MQTT-broker* che rappresenta una connessione condivisa con un broker MQTT. Un **flusso** è rappresentato da un tab o foglio di lavoro ed è il metodo principale per organizzare il lavoro. Informalmente un flusso rappresenta un insieme di nodi connessi fra loro. Un **sottoflusso** è una collezione di nodi che vengono collassati graficamente in un unico nodo. Possono essere utilizzati per ridurre la complessità grafica del flusso, oppure per riutilizzare un gruppo di nodi più volte in un flusso.

Figura 2.4: Esempio di sottoflusso

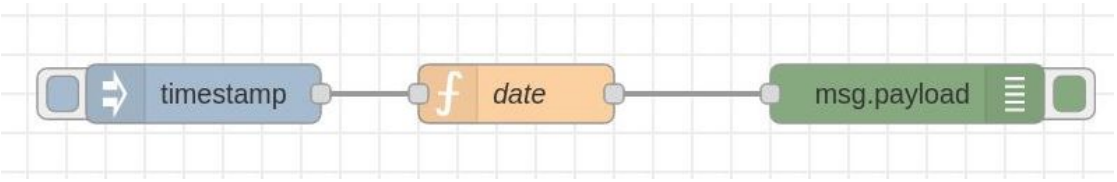


Il **contesto** è il modo principale di condividere informazioni tra i nodi senza essere connessi fra loro. Ne esistono di tre tipi:

- Node: Visibile solamente dal nodo che setta il valore
- Flow: Visibile a tutti i nodi che appartengono allo stesso flusso (o tab)
- Global: Visibile a tutti i nodi

Un **messaggio** è un oggetto JSON che viene passato attraverso i nodi. Di default si chiama *msg* ed ha due campi: *payload* e *id*. La **palette** è posizionata sulla sinistra dell'interfaccia e lista tutti i nodi presenti nel sistema. Dei nodi extra possono essere aggiunti direttamente dall'editor

attraverso la sezione impostazioni, si apre una finestra con una barra di ricerca, una volta inserito l'input vengono suggeriti dei nodi che hanno affinità con l'input. I nodi che vengono consigliati sono tutti sul repository ufficiale di Node-RED. Il **workspace** è lo spazio principale, dove tutti i nodi vengono posizionati in modo da creare degli automatismi. La **sidebar** è uno spazio posizionato a destra del workspace in cui ci sono degli strumenti utili, ad esempio la console di debug, l'interfaccia della configurazione dei nodi e la finestra descrittiva per ogni nodo. È molto orientato al riuso ed alla portabilità, infatti tutti i nodi o collezioni di nodi sono moduli Javascript che possono essere caricati e utilizzati poi da altre persone tramite NPM. Inoltre, ogni flusso viene poi tradotto in codice JSON per poter essere esportato ed importato in altri ambienti. Viene fornita anche una 'vetrina' (flows.nodered.org) dove possono essere esplorati nodi e flussi creati e pubblicati per la community. Esempio di un semplice flusso Node-RED:



Il nodo *date* non è altro che un nodo function che restituisce la data:

```
// Create a Date object from the payload
var date = new Date(msg.payload);
// Change the payload to be a formatted Date string
msg.payload = date.toString();
// Return the message so it can be sent on
return msg;
```

Viene tradotto così in JSON:

```
[
    {
        "id":"58ffae9d.a7005",
        "type":"debug",
        "name":"",
        "active":true,
        "complete":false,
        "x":640,"y":200,
        "wires":[]
    },
    {
        "id":"17626462.e89d9c",
        "type":"inject",
        "name":"timestamp",
        "topic":"",
        "payload":"",
        "repeat":"",
        "once":false,
        "x":240,
        "y":200,
        "wires":[["2921667d.d6de9a"]]
    },
    {
        "id":"2921667d.d6de9a",
        "type":"function",
        "name":"date",
```

```
27          "func":"// Create a Date object from the payload\nvar date = new Date(msg.
        payload);\n// Change the payload to be a formatted Date string\nmsg.payload =
        date.toString();\n// Return the message so it can be sent on\nreturn msg;",
28          "outputs":1,
29          "x":440,
30          "y":200,
31          "wires":[["58ffae9d.a7005"]]
32      }
33  ]
```

Dato che Node-RED ha tutti i dati in-memory, per salvare i flussi creati utilizza (di default) un file chiamato *flows.json*. Questo rende molto facile condividere i progetti.

Requisiti di sistema

Una caratteristica del precedente sistema che viene riportata nel nuovo è la divisione netta dei dati dei vari gruppi e la condivisione di essi nel gruppo di appartenenza. Questa scelta consente agli utenti di avere a disposizione più dati rispetto a quelli che avrebbero utilizzando solo i propri dispositivi. In media un utente ha un solo dispositivo proprietario, mentre un gruppo ne ha tra i dieci ed i quindici. Questo consente agli utenti di avere **ampia libertà nella scelta delle condizioni**. Quindi un requisito fondamentale è la *scissione e la replica di un sistema in più "copie" con dati differenti* (i gruppi in questo caso). I requisiti non funzionali che devono essere rispettati sono:

- Funzionalità: Devono essere garantite le funzionalità presenti nel precedente sistema ed in più ampliare dove possibile con le richieste (valutate dal team di sviluppo) degli utenti.
- Usabilità: Il sistema deve risultare semplice e utilizzabile ad un pubblico più vasto possibile. Verrà testata l'usabilità con varie tipologie di test.
- Affidabilità: Deve essere garantita la sicurezza dei dati e la privacy degli utenti. Il sistema deve essere costruito seguendo *privacy by design*.
- Prestazioni: Il sistema non deve sprecare risorse (sia di memoria, sia computazionali) e deve essere progettato con molta attenzione alle performance.

L'usabilità verrà discussa nel capitolo 3 e nel capitolo 5, l'affidabilità nel capitolo 4.

Vantaggi nell'utilizzo di Docker

Lo strumento che più soddisfa i requisiti imposti è **Docker**. È uno dei sistemi più noti per il deployment di applicazioni software. Dalla pagina Wikipedia che descrive Docker [11]:

> *"Docker è un progetto open-source che automatizza il deployment (consegna o rilascio al cliente, con relativa installazione e messa in funzione o esercizio, di un'applicazione o di un sistema software tipicamente all'interno di un sistema informatico aziendale) di applicazioni all'interno di contenitori software, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. Docker utilizza le funzionalità di isolamento delle risorse del kernel Linux come ad esempio cgroup e namespace per consentire a "container" indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale."*

Il concetto di container è già disponibile in Linux per eseguire un processo su cui possono essere limitate e gestite le risorse a lui allocate. Il problema dei container LXC (disponibili in Linux utilizzando cgroup e namespace) è che se un container viene spostato su una macchina con configurazione differente è molto probabile che non funziona, dato che ogni container LXC dipende strettamente dall'ambiente in cui è costruito (dipendenze esterne, configurazione di rete, memoria e impostazioni aggiuntive). Docker aggiunge **portabilità** ai container, cioè utilizzando un motore Docker i container diventano indipendenti dalla macchina in cui è stato creato. Altra funzione aggiunta è il **versioning**; si può tenere traccia delle varie versioni di un container, dando la possibilità di un upgrade o downgrade. Inoltre è orientato al **riuso**, dato che ogni container può essere utilizzato come base per uno più specifico [12]. Docker definisce un formato unico per creare container, più nello specifico *immagini*. Una immagine è il risultato dell'esecuzione di un file chiamato **Dockerfile**. Viene fornito il DockerHub, un portale dove sono presenti migliaia di immagini già pronte che possono essere utilizzate come base di partenza.

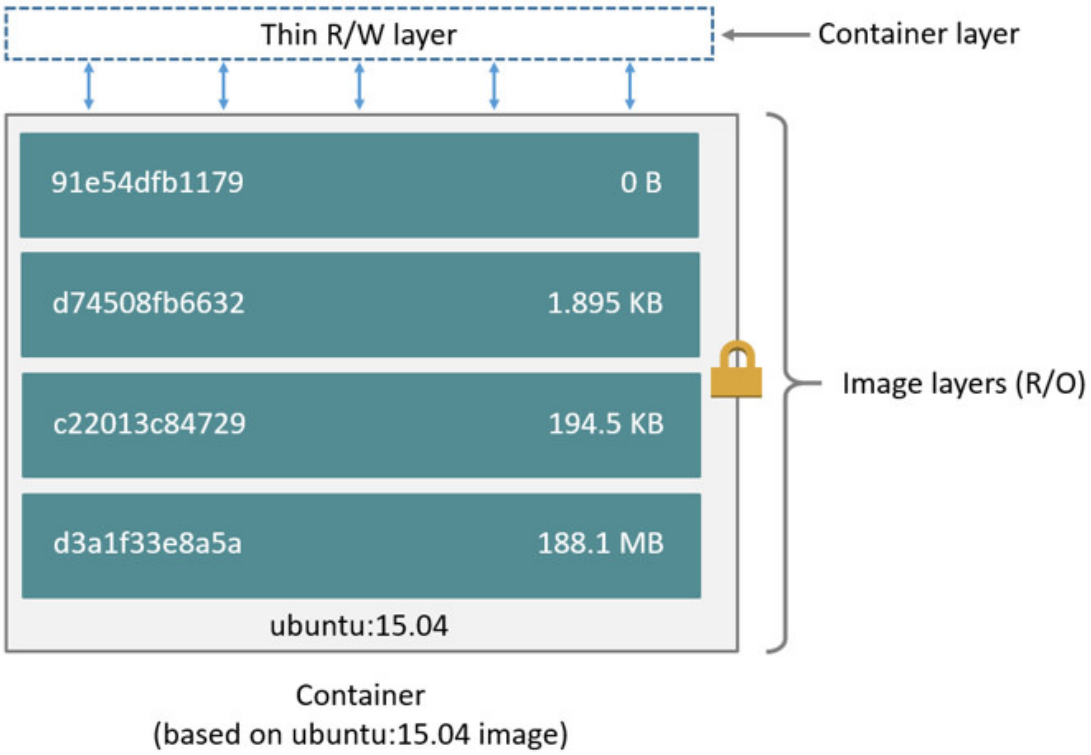
Esempio di Dockerfile:

```
1 FROM ubuntu:18.04
2 COPY . /app
3 RUN make /app
4 CMD python /app/app.py
```

Ogni istruzione crea un livello:

- FROM crea un livello dall'immagine Docker di Ubuntu versione 18.04
- COPY aggiunge i file dalla cartella corrente del Docker client
- RUN costruisce l'applicazione con make
- CMD specifica quale comando eseguire nel container

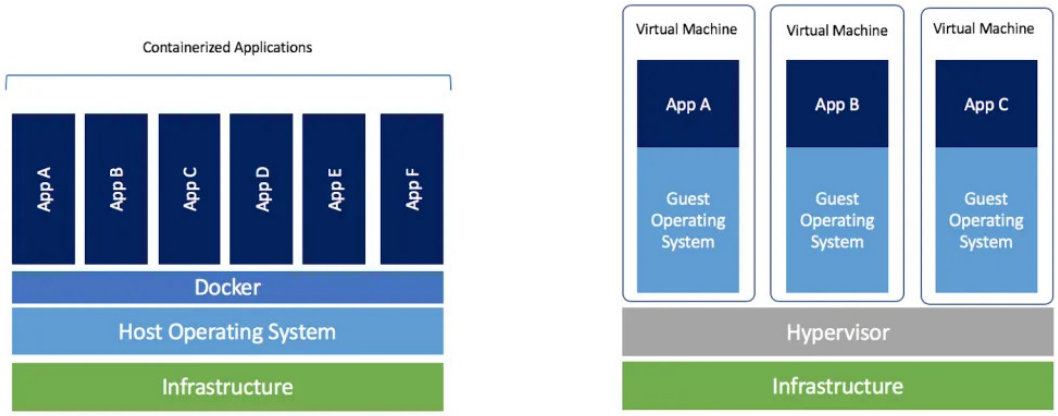
Una immagine Docker consiste in livelli read-only i quali rappresentano il risultato di istruzioni Docker. Una istruzione aggiunge qualcosa al livello precedente, in particolare vengono aggiunte solo le differenze dal livello che lo precede. Quando viene eseguita un'immagine e generato un container, si aggiunge un ulteriore livello sopra ai precedenti. Tutte le modifiche poi fatte al container in esecuzione sono eseguite su questo nuovo livello che quindi non è read-only,



ma l'unico writable ed è comunemente chiamato *container layer*. Questo tipo di architettura favorisce la condivisione di risorse, infatti se sono presenti più container creati partendo dalla stessa immagine, le modifiche verranno effettuate solo nell'ultimo livello dei rispettivi container condividendo tutti i livelli sottostanti[13]. Dato che tutti i dati scritti dal container vengono salvati nell'ultimo livello modificabile, quando il container viene distrutto vengono persi anche i dati scritti dal container. La persistenza dei dati può essere garantita da due metodi differenti. Un **volume** è una parte del filesystem amministrata da Docker in cui solo i processi Docker possono avere accesso. Solitamente risiedono in `/var/lib/docker/volumes/`. Un **bind mount** differisce dal volume in due punti: ogni processo può avere accesso ai dati e può risiedere ovunque nel sistema. Questi due metodi garantiscono che se il container va in uno stato di errore o la macchina dove viene deployato il container si spegne, i dati vengono comunque salvati in un disco fisso.
Differisce dalla **virtualizzazione** [14] in:

- Deployment semplificato: i container possono essere creati, distrutti e replicati in qualsiasi ambiente in modo semplice e veloce.
- Ampia portabilità: permette di impacchettare l'applicativo in un singolo elemento facilmente distribuibile.
- Prestazioni: Sono più leggeri e veloci delle macchine virtuali perché non richiedono una virtualizzazione del sistema operativo e condividono le risorse tra i vari container.
- Isolamento: Per quanto un container Docker isoli i vari applicativi, una Virtual Machine garantisce un maggiore isolamento.

Figura 2.5: Container vs Virtualizzazione



Questa divisione netta fra i container e la condivisione delle risorse per aumentare le prestazioni si prestano molto al caso in esame. Ogni container avrà in esecuzione una instanza di node-red configurata con i dati di un determinato gruppo. Come si può notare dai grafici nelle prossime pagine, tratti dallo studio di *Potdar et al.*[15], ci sono molti vantaggi prestazionali nell'utilizzo di container Docker invece di macchine virtuali.

La comparazione nei test delle prestazioni della CPU è stato fatta utilizzando gli strumenti sysbench, Phoronix e Apache benchmark. È stato utilizzato il metodo del **massimo numero primo**. Viene dato un tempo massimo, in questo caso 60 secondi e deve essere dato in risultato il numero primo massimo minore di 50000 con 4 thread operativi. Per il test di compressione dati è stato utilizzato 7-Zip con un file di 10GB. Le prestazioni della memoria RAM (utilizzando RAM speed/SMP) sono state misurate con gli strumenti INTmark e FLOATmark che misurano le prestazioni massime possibili di cache e memoria mentre leggono e scrivono blocchi individuali di dati. In un ulteriore test è stato utilizzato un caso particolare, cioè Eigth Queen, una simulazione di un problema del gioco degli scacchi. Come possiamo notare dai grafici, le prestazioni dei

Table 1. Virtual Machine versus Docker Container

	Virtual Machines	**Docker Containers**
Isolation Process Level	Hardware	Operating System
Operating System	Separated	Shared
Boot up time	Long	Short
Resources usage	More	Less
Pre-built Images	Hard to find and manage	Already available for home server
Customised preconfigured images	Hard to build	Easy to build
Size	Bigger because they contain whole OS underneath	Smaller with only docker engines over the host OS
Mobility	Easy to move to a new host OS	Destroyed and recreated instead of moving.
Creation time	Longer	Within seconds

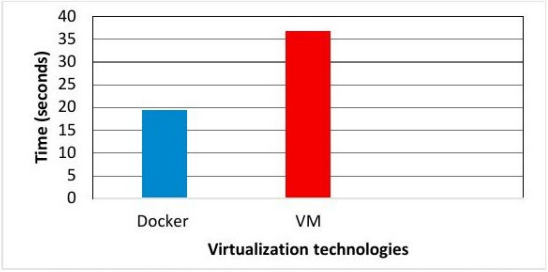


Fig. 4. CPU Comparison of Docker and Virtual Machine

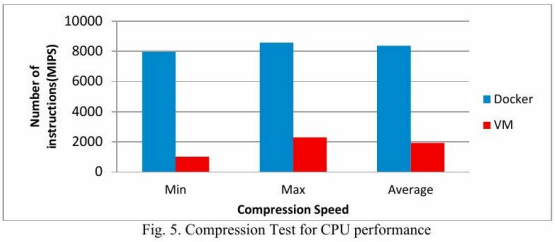


Fig. 5. Compression Test for CPU performance

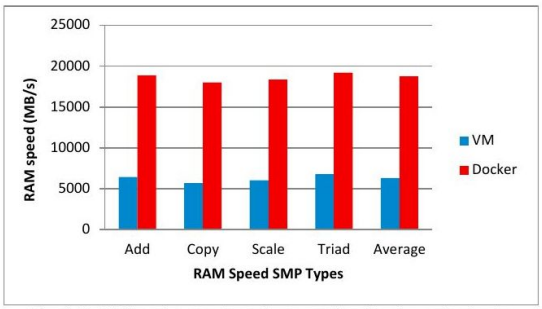


Fig. 6. RAM Speed comparisons between virtualization technologies

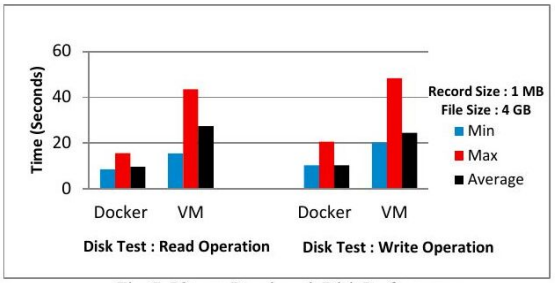


Fig. 7. IOzone Benchmark Disk Performance

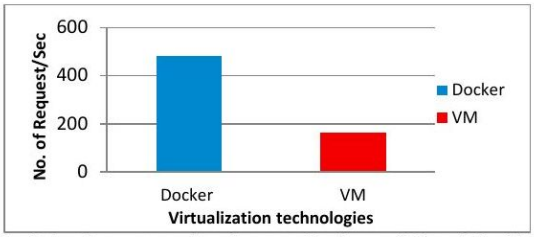


Fig. 8. Load test comparison between Docker and Virtual Machine

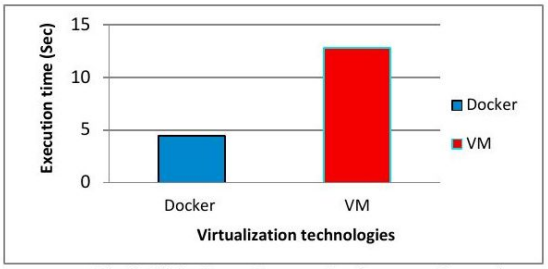


Fig. 9. Eight Queen Program Performance Comparison

container Docker sono di gran lunga superiori rispetto alle macchine virtuali. Dato che l'immagine del nostro sistema è identica per tutti i container si ha un netto miglioramento delle performance utilizzando container perché tutti condividono la stessa immagine di base.

Esempio di immagine utilizzata in SeismoCloud partendo dalla immagine di Node-RED su DockerHub come base(https://hub.docker.com/r/nodered/node-red/).

```
1  # from nodered/node-red:1.0.6-12
2  FROM nodered/node-red@sha256:82
       f367ab41d19e3ebd5e9b1cc62e4f615d4b4be4fa8bc83f1bf5441ba8c0d32a
3
4  # Qui vengono settate tutte le variabili d'ambiente
5  ENV GROUP_ID="ID_35762634"
6
7  # Vengono indicati i volumi per la persistenza dei dati
8  VOLUME /volume
9
10 # Tutte le cartelle vengono copiate nell'immagine
11 COPY data data
```

```
12
13 RUN chown -R node-red data
14 USER node-red
15
16 RUN npm install ... #Tutti i nodi custom necessari
17
18 CMD ["npm", "start"]
```

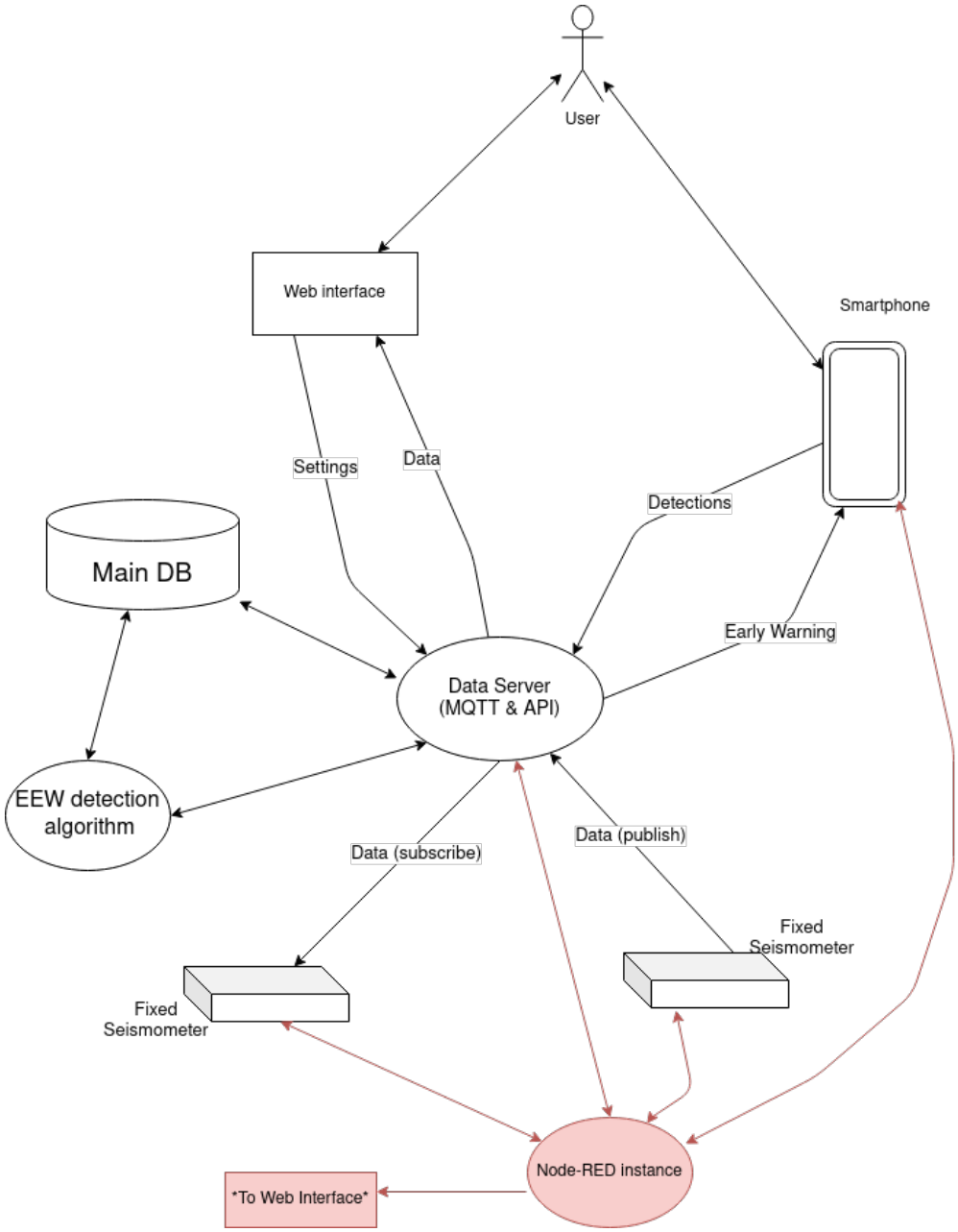
La seconda riga differisce dall'immagine definita nel Dockerfile di esempio per un codice dopo la definizione dell'immagine di partenza. Questo codice è formato da due campi: `algoritmo:valore-esadecimale`. In questo caso l'algoritmo utilizzato è `sha256`. Viene calcolato lo stesso algoritmo su tutti i livelli che compongono l'immagine, successivamente viene creato un file `Manifest`, a cui poi viene calcolato lo stesso algoritmo di hash che corrisponde al digest dell'immagine. Ovviamente questo hash serve a verificare che effettivamente quello che è stato scaricato corrisponde all'immagine desiderata. Non utilizzando un hash, ma come scritto prima un tag (ubuntu:`18:04`) viene scaricata un'immagine ma non viene verificato il contenuto che appunto potrebbe essere manomesso.

Il deploy di tanti servizi può essere semplificato utilizzando **docker-compose**.

```
1  version: '3'
2  services:
3
4    seismocloud-nodered-1:
5      restart: always
6      image: seismocloud-nodered:latest
7      environment:
8        - NUM_GROUP = 2222345435
9      ports:
10       - 1880:1880
11     volumes:
12       - ./volume:/volume
13
14   seismocloud-nodered-2:
15     restart: always
16     image: seismocloud-nodered:latest
17     environment:
18       - NUM_GROUP = 2222345436
19     ports:
20       - 1880:1880
21     volumes:
22       - ./volume:/volume
```

Compose è uno strumento per definire ed eseguire applicazioni Docker multi-container. Si usa un file di tipo YAML per definire tutti i componenti ed impostare il servizio. Vengono creati i file Dockerfile relativi a tutte le immagini che devono essere costruite ed un file `docker-compose.yml`. In questo caso appunto, vengono eseguiti due servizi node-red con immagine customizzata (seismocloud-nodered) per due differenti gruppi. Con un singolo comando si possono eseguire tutti i servizi configurati in precedenza (*docker-compose up*), oppure stopparli (*docker-compose stop*) e con un altro farli ripartire (*docker-compose restart*). Se, mentre tutti i servizi sono in esecuzione, si vuole modificare lo stato di uno solo è possibile farlo con i comandi di Docker.

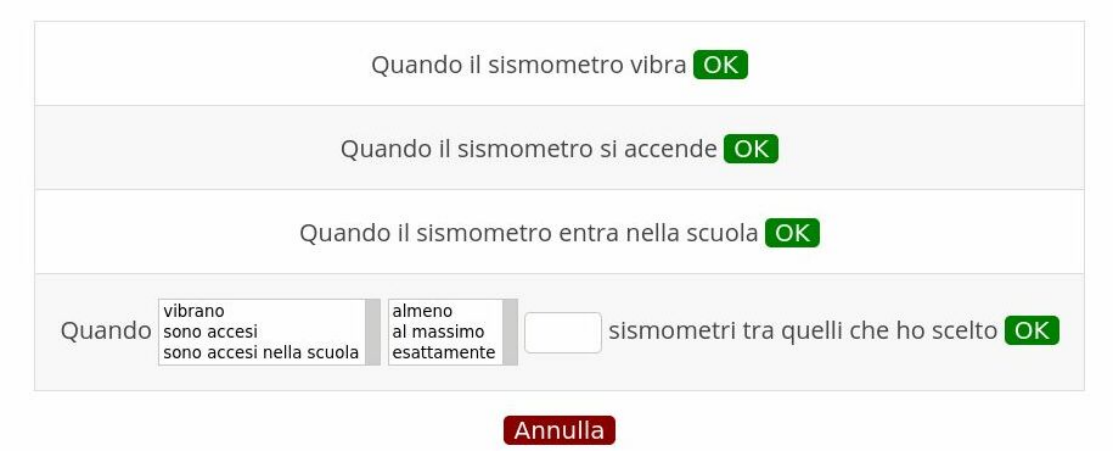
Qui con **node-RED instance** si intende un container per un gruppo specifico. Il container interagisce con l'interfaccia web per garantire l' autenticazione di un utente; interagisce con i sismometri attraverso i vari protocolli per lo scambio di dati (HyperText Transfer Protocol, Message Queue Telemetry Transport). Interagisce con gli smartphone nel caso in cui gli utenti decidono di ricevere notifiche push o altri tipi di alert (messaggi Telegram o altra applicazione). Con il data server scambia informazioni attraverso il broker mqtt e riceve informazioni sul gruppo (statistiche e dati dei sismometri) attraverso le API.



3. Implementazione delle funzionalità

3.1 Sistema Legacy

Come è stato accennato nella descrizione dell'architettura del sistema SeismoCloud (Cap 1.3), è presente un sistema di End User Development. In particolare è possibile programmare della azioni quando si verificano determinate condizioni pur non avendo competenze in programmazione software. Le condizioni disponibili sono tre: quando n (o un solo) sismometri vibrano/sono accesi, quando un sismometro entra in una scuola, oppure la notifica di un Earthquake Early Warning. Le possibilità per il metodo di notifica sono quattro: un messaggio tramite Telegram, una notifica push ai membri del gruppo, un widget e la pubblicazione di una API. Visto che i dati

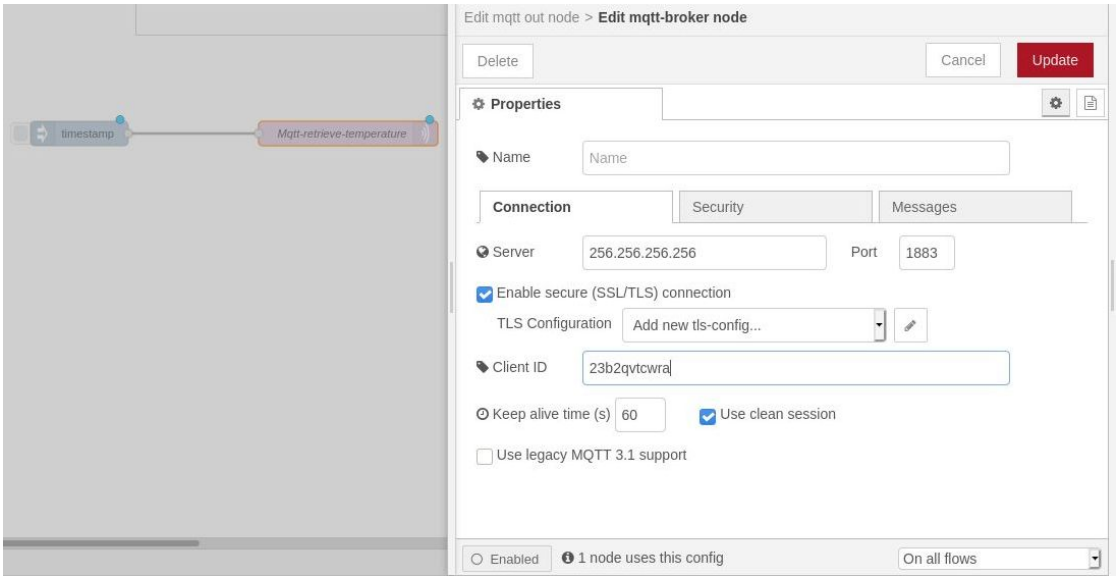


prodotti sono tanti, sarebbe molto utile ampliare il range di condizioni ad un più vasto campo, tenendo in conto anche di tutti i dati che non vengono considerati in questo sistema, ma che vengono comunque collezionati dai sismometri (temperatura, indirizzi IP, soglia di Quake, rssi, reboot..). Sarebbe ancora più utile lasciare completa **libertà all'utente** (sotto alcune restrizioni di sicurezza) integrando il sismometro con l'ambiente locale potenziando così le funzionalità e l'utilità di un sismometro. Quindi, un primo problema del sistema legacy di EUD è sicuramente la rigidità nelle azioni messe a disposizione per l'utente. Con Node-RED, possono essere costruite tantissime azioni automatizzate dato l'elevato numero di funzionalità offerte. Si potrebbe quindi decidere di riavviare un sismometro ogni volta che la temperatura supera i 50°C. Oppure mandare una mail, pubblicare un tweet e scaricare un file quando vengono rilevate tre vibrazioni nello stesso minuto. Un ulteriore vantaggio è l'integrazione con i dispositivi smart/IoT. Node-RED è stato ideato e viene mantenuto come strumento per la programmazione flow-based per ambienti IoT/industriali. Ad esempio se si possiede qualche dispositivo per una smart home (telecamere, smart TV, frigorifero etc...) possono essere collegati al sismometro ed automatizzati basandosi anche su eventi sismici. Il tutto non deve risultare complesso, deve essere possibile l'utilizzo per utenti non esperti in informatica o elettronica, ma allo stesso tempo i più competenti devono poter sfruttare le loro abilità. La soluzione a questo bisogno dovrebbe essere: *facilmente utilizzabile, sicura, affidabile e performante*. Senza ombra di dubbio un punto di forza del precedente sistema è l'elevata **usabilità**. Infatti, essendo disponibili poche funzionalità e quindi pochi pulsanti

è molto facile da comprendere. Questa facilità d'uso deve essere mantenuta anche nel nuovo sistema per non escludere utenti non tecnicamente competenti. Il vecchio sistema di EUD rende possibile, come già introdotto, la programmazione di azioni condivisa con il proprio gruppo di appartenenza. Questa impostazione viene ripresa nel nuovo sistema, quindi ogni membro di un gruppo deve poter accedere alla propria *istanza di gruppo di Node-RED* per programmare le azioni desiderate con tutti e soli i sismometri appartenenti al gruppo. Le azioni si possono basare su dati provenienti da tutta la rete. La ragione principale di questa scelta è che in questo modo si hanno molti più dati disponibili.

3.2 Difficoltà e limiti nell'uso di Node-RED

L'immagine di partenza di Node-RED fornisce alcune funzionalità di base. Nella palette di sinistra sono presenti sei sezioni (common, function, network, sequence, parser, storage) che raggruppano i vari nodi in base all'argomento di appartenenza. È stato già anticipato che questo strumento viene utilizzato nell'industria e ha un grande potenziale, possono essere programmate un vasto range di azioni utilizzando molte tecnologie e protocolli differenti. Sono presenti anche azioni che non possono essere programmate ad *alto livello*, cioè si deve scendere nei tecnicismi della tecnologia utilizzata. Un esempio di ciò è l'**invio di una richiesta di sottoscrizione ad un topic verso il broker mqtt di SeismoCloud**.

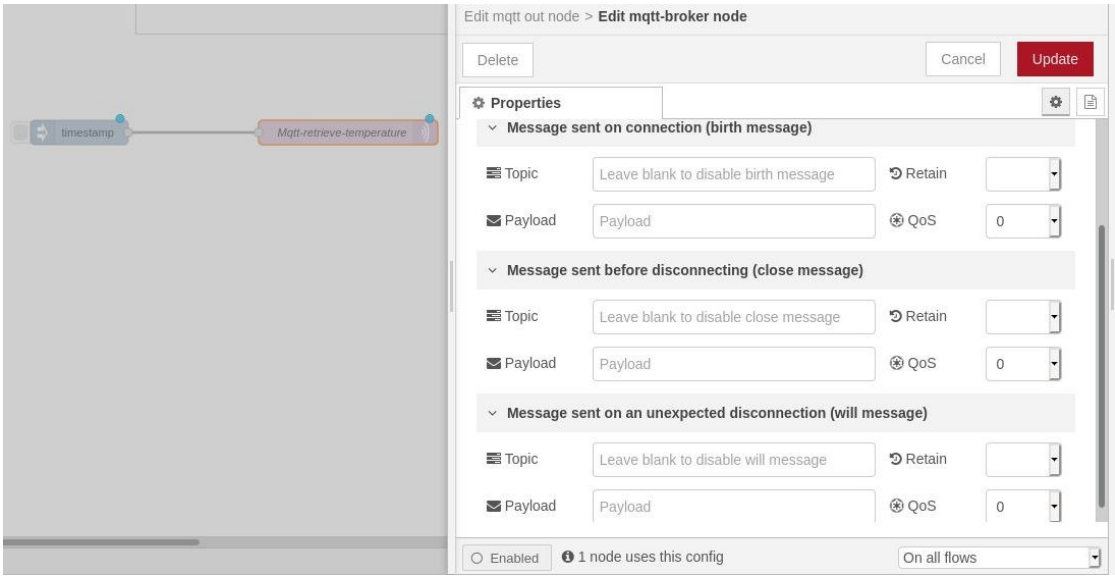


Il problema che si nota qui è l'elevato numero di impostazioni e settaggi che l'utente deve inserire per configurare correttamente un client MQTT per ricevere un solo tipo di dato. Vengono richieste competenze in questa specifica tecnologia e non solo (anche nozioni generali di reti). Ad esempio, si deve inserire l'indirizzo IP del server (in questo caso scelto appositamente non esistente come esempio) `256.256.256.256`, la porta di riferimento `1883`, un client ID ed un keep alive time. Se si vuole anche configurare una connessione sicura tramite *SSL/TSL* devono essere configurati anche i relativi parametri per la cifratura dei dati. Se il server è protetto da username e password (come nel caso SeismoCloud), c'è un tab anche per quello. Quindi qui c'è anche un problema di sicurezza: per fornire dati agli utenti devono essere fornite anche le informazioni di autenticazione ed accesso al server. Inoltre, ovviamente, l'utente deve essere a conoscenza di come sono formati tutti i topic di cui ha bisogno. Se inoltre si vogliono pubblicare messaggi e non solamente riceverli si hanno a disposizione più tipologie di messaggi per comunicare con il server. Il protocollo MQTT inoltre ha il particolare *Quality of Service* che funziona in questo modo: Si può settare un valore numerico (0 , 1 o 2) che corrisponde al grado di accuratezza ed affidabilità nella consegna dei messaggi.

- QoS 0: Con questa impostazione si ha un servizio *orientato alle prestazioni con il minor impiego di energia*. Il client non riceve feedback dell'avvenuta consegna del messaggio.

- QoS 1: Quando si riceve un messaggio con QoS uguale ad 1, si deve inviare un messaggio di acknowledgment che certifica che la ricezione è correttamente avvenuta. Specularmente, se si invia un messaggio con QoS 1 allora si deve aspettare il feedback. Nel caso di un problema di rete (ad esempio congestione) e il mittente non riceve in un determinato lasso di tempo l'acknowledgment rinvierà di nuovo il messaggio. *Questo può portare ad avere messaggi duplicati.*
- QoS 2: È la modalità più costosa in termini di sforzo (in modo relativo, il protocollo è progettato per una buona gestione delle risorse). Vengono inviati più messaggi per confermare la ricezione ed evitare eventuali duplicati.

Ovviamente non esiste un valore QoS migliore in assoluto, ma dipende dal messaggio e dallo scopo di esso. Inoltre questo fa comprendere quante competenze sono richieste per l'invio di un messaggio con protocollo MQTT. Ne esistono differenti di protocolli disponibili, quindi la conoscenza tecnica richiesta è elevata pur non richiedendo esperienza in programmazione attraversi dei linguaggi.



Quindi bisogna semplificare tutto questo processo minimizzando il numero di interazioni con l'utente e dando per scontati tutti i dati che sono già noti al sistema SeismoCloud. Questo servirà a semplificare e rendere più usabile il sistema, considerando che il sistema precedente nascondeva tutte queste configurazioni aggiuntive.

3.3 Implementazione delle funzionalità tramite nodi

Come è stato introdotto, tutte le funzionalità aggiuntive personalizzabili sono implementate tramite *nodi*. **L'obiettivo principale è cercare di costruire nodi che hanno la minima configurazione necessaria con il massimo delle funzionalità offerte**.

Come è costruito un nodo

I nodi sono moduli NPM ed hanno bisogno di un file Javascript per le funzionalità programmate, un file HTML per il layout ed un file JSON (chiamato *package.json*) per pacchettizzarlo in un modulo. Il file package.json può essere prodotto in modo automatico con il comando fornito dal package manager `npm init`.

```
1 {
2     "name" : "node-red-contrib-example-lower-case",
3     ...
4     "node-red" : {
5         "nodes": {
6             "lower-case": "lower-case.js"
7         }
8     }
9 }
```

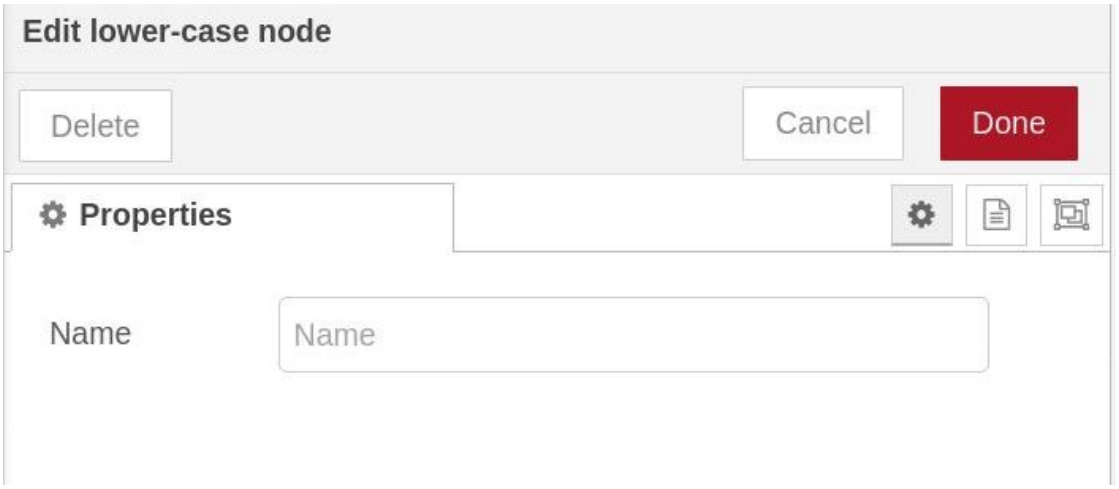
```
1  module.exports = function(RED) {
2      function LowerCaseNode(config) {
3          RED.nodes.createNode(this,config);
4          var node = this;
5          node.on('input', function(msg) {
6              msg.payload = msg.payload.toLowerCase();
7              node.send(msg);
8          });
9      }
10     RED.nodes.registerType("lower-case",LowerCaseNode);
11 }
```

Qui abbiamo un esempio di file Javascript. Nella riga `1` il modulo esporta una funzione che viene eseguita quando il runtime Node-RED viene eseguito. La funzione definisce un solo input `RED`, il quale è un oggetto che garantisce l'accesso alle API del runtime RED. Dalla riga `2` in poi abbiamo la definizione del nodo, viene creato un nuovo nodo con le configurazioni di base (riga `3`). Dalla riga `5` alla riga `8` viene definito il comportamento quando il nodo riceve un input, ossia viene ricevuto l'oggetto `msg`, viene estratto il `msg.payload` e viene trasformato in minuscolo. Successivamente viene restituito in output l'oggetto `msg`. Nella riga `10` viene poi aggiunta al runtime api la funzione appena sopra definita tramite il nome "lower-case".

```
1  <script type="text/javascript">
2      RED.nodes.registerType('lower-case',{
3          category: 'function',
4          color: '#a6bbcf',
5          defaults: {
6              name: {value:"lower-case"}
7          },
8          inputs:1,
9          outputs:1,
10         icon: "file.png",
```

```
11          label: function() {
12              return this.name||"lower-case";
13          }
14      });
15  </script>
16
17  <script type="text/html" data-template-name="lower-case">
18      <div class="form-row">
19          <label for="node-input-name"><i class="fa fa-tag"></i> Name</label>
20          <input type="text" id="node-input-name" placeholder="Name">
21      </div>
22  </script>
23
24  <script type="text/html" data-help-name="lower-case">
25      <p>A simple node that converts the message payloads into all lower-case
        characters</p>
26  </script>
```

Infine abbiamo il file HTML che definisce il layout del nodo. Nella riga 2 viene associata tutta la definizione delle impostazioni che segue con il nodo chiamato "lower-case". Viene associata una categoria al nodo alla riga 3. La categoria è uno strumento utile per l'usabilità, infatti permette di raggruppare nodi simili sotto lo stesso insieme. In questo caso viene inserito all'interno di *function* essendo una funzione Javascript che riceve e modifica l'input. Viene poi assegnato un colore al nodo in dicitura esadecimale uguale per tutti i nodi della stessa categoria, dei parametri di default (in questo caso solamente il nome del nodo), il numero di input e output accettati, un file PNG per associare al nodo un logo che ricorda l'azione che la funzione svolge.



Configurazione Node-RED

Quando Node-RED viene eseguito deve essere presente il file `settings.js` dove vengono settate tutte le configurazioni[18]. Sono disponibili cinque sezioni di configurazione: configurazione del runtime, dell'editor e dei suoi temi, della dashboard e dei nodi. Questo file offre molte possibilità di configurazione, ma qui vengono esposte solo quelle utilizzate nel sistema EUD di SeismoCloud. Come è stato spiegato nel paragrafo 2.2, i flussi creati vengono salvati tramite un file JSON che può essere settato tramite il valore `flowFile: <flowsfilename>.json`. I nodi vengono cercati dal runtime all'interno della cartella definita da `nodesDir` (default `$HOME/.node-red/nodes`). È possibile definire una porta dedicata al servizio, di default è 1880. Sono disponibili vari livelli di logging:

- fatal: Solo gli errori che rendono l'applicativo inutilizzabile sono registrati
- error: Registra gli errori che sono ritenuti fatali per una particolare richiesta + fatal errors
- warn: Registra problemi che non sono fatali + errors + fatal errors
- info: Registra informazioni riguardo l'esecuzione generale dell'applicativo + warn + error + fatal errors
- debug: Registra informazioni più verbose rispetto ad info + info + warn + error + fatal errors
- trace: Registra log dettagliati + debug + info + warn + error + fatal errors

Come si può facilmente notare `fatal` corrisponde al meno verboso e gradualmente si aumenta fino ad arrivare a loggare ogni singola informazione. Di default il livello è settato ad `info`; viene lasciato quest'ultimo perché essendo un sistema appena creato e non maturo è fondamentale una attività intensa dei log, utili a fornire feedback e segnalare eventuali errori; non viene utilizzato il `debug` perché troppo verboso e non adatto.

Per quanto riguarda la configurazione dell'editor, qui si ha la possibilità di settare le categorie per i nodi. Di default sono presenti: `['common', 'function', 'network', 'sequence', 'parser', 'storage']`; nel sistema invece sono state rimosse le categorie `storage` e `network` (nel capitolo 4 vengono analizzate in dettaglio le motivazioni) e sono state aggiunte le categorie `SeismometerData` e `apiSeismocloud`. I nodi che rientrano nelle categorie `network` e `storage` sono eliminati con questo comando:

```
1 //EXCLUDING NODES FROM PALETTE:
2   nodesExcludes: ['32-udp.js', '31-tcpin.js', '22-websocket.js', '21-httprequest.
      js', '21-httpin.js',
3     '10-mqtt.js', '06-httpproxy.js', '05-tls.js', '90-exec.js', 'node-red-node-
      tail', '10-file.js',
4     '23-watch.js'],
```

Queste sono le configurazioni disponibili per il tema dell'editor, utili soprattutto per una alta comprensibilità dell'interfaccia.

```
1 editorTheme: {
2     page: {
3         title: "SeismoCloud EUD",
4         favicon: "/absolute/path/to/theme/icon",
5         css: "/absolute/path/to/custom/css/file",
6         scripts: [ "/absolute/path/to/custom/script/file", "/another/script/file"]
```

```
    },
    header: {
        title: "SeismoCloud EUD",
        image: "/absolute/path/to/header/image", // or null to remove image
        url: "https://www.seismocloud.com/" // optional url to make the header
    text/image a link to this url
    },
    deployButton: {
        type:"simple",
        label:"Save",
        icon: "/absolute/path/to/deploy/button/image" // or null to remove image
    },
    menu: { // Hide unwanted menu items by id. see packages/node_modules/@node-red
    /editor-client/src/js/red.js:loadEditor for complete list
        "menu-item-import-library": false,
        "menu-item-export-library": false,
        "menu-item-keyboard-shortcuts": false,
        //"menu-item-help": {
            //label: "Alternative Help Link Text",
            //url: "http://example.com"
        //}
    },
    userMenu: false, // Hide the user-menu even if adminAuth is enabled
    palette: {
        editable: false, // Enable/disable the Palette Manager
        //theme: [ // Override node colours - rules test against category/type by
    RegExp.
        //    { category: ".*", type: ".*", color: "#f0f" }
        //]
    },
    projects: {
        enabled: false // Enable the projects feature
    }
},
```

È possibile aggiungere un titolo personalizzato alla pagina Web, una icona, del codice css e degli script personalizzati. Nell'interfaccia c'è un header che ospita una immagine che può funzionare da link, nel nostro caso rimanda al sito SeismoCloud. In alto a destra nell'interfaccia, il tasto Deploy (che appunto effettua il deploy dell'applicativo) è rinominato in `Save` perché più comprensibile ad un pubblico non esperto. Nel menu è possibile far comparire o non comparire alcune funzionalità, nel caso SeismoCloud sono utili e perciò sono lasciate attive (ad esempio la possibilità di esportare ed importare flussi pronti tramite il file in formato json. Vari gruppi possono condividere un flusso complesso cambiando pochi dettagli). È stato disabilitato il palette manager perché consentiva di aggiungere nodi esterni alla palette tramite Internet, potenzialmente un problema di sicurezza (analizzato nel capitolo 4). Anche la funzionalità projects per ora è disabilitata perché non necessaria, almeno per le prime release del sistema (Projects permette di avere un supporto per `Version Control System` come GitHub etc..).

Queste sono invece le impostazioni di configurazione dei nodi: in millisecondi sono definiti i tempi di riconnessione e timeout per i vari protocolli e relative richieste.

```
  // Retry time in milliseconds for MQTT connections
  mqttReconnectTime: 15000,

  // Retry time in milliseconds for Serial port connections
  serialReconnectTime: 15000,

  // Retry time in milliseconds for TCP socket connections
```

```
8   socketReconnectTime: 10000,
9
10  // Timeout in milliseconds for TCP server socket connections
11  //  defaults to no timeout
12  socketTimeout: 10000,
13
14  // Maximum number of messages to wait in queue while attempting to connect to
      TCP socket
15  //  defaults to 1000
16  tcpMsgQueueSize: 1000,
17
18  // Timeout in milliseconds for HTTP request connections
19  //  defaults to 120 seconds
20  httpRequestTimeout: 10000,
21
22  // The maximum length, in characters, of any message sent to the debug sidebar
      tab
23  debugMaxLength: 1000,
```

Implementazione di nodi che utilizzano MQTT per i dati relativi ai sismometri

Come spiegato nel paragrafo 3.2 “Difficoltà e limiti nell’uso di Node-RED”, un utente che non ha competenze tecniche informatiche potrebbe avere grandi difficoltà nel configurare un’azione piuttosto semplice. L’obiettivo del lavoro svolto è di semplificare il più possibile la creazione di flussi Node-RED riducendo al minimo le competenze richieste e la configurazione da parte dell’utente. Molti valori che vengono inseriti (e poi ripetuti nel caso di più nodi che utilizzano il protocollo MQTT) sono già a conoscenza del sistema SeismoCloud, perciò si è optato per la creazione di nodi customizzati che hanno già all’interno tutte le informazioni note.

Qui è presente la lista dei vari nodi creati per il sistema EUD di SeismoCloud; sono tutti moduli NPM che rispettano la definizione di modulo definita precedentemente in questo paragrafo.

- node-red-contrib-rssi (Input: 0, Output: 1): Questo nodo produce in output l’indicatore di potenza del segnale ricevuto (RSSI), una misura stimata di quanto bene un dispositivo può ricevere segnali dal punto di accesso a cui è collegato.
- node-red-contrib-bssid (Input: 0, Output: 1): Questo nodo produce in output il BSSID, che identifica l’insieme di servizi di base del dispositivo. Il più delle volte è associato all’indirizzo MAC del punto di accesso a cui è collegato.
- node-red-contrib-essid (Input: 0, Output: 1): Questo nodo restituisce il nome del punto di accesso alla rete a cui è collegato il dispositivo.
- node-red-contrib-localip (Input: 0, Output: 1): Questo nodo restituisce l’indirizzo Ip nella rete locale del dispositivo.
- node-red-contrib-publicip (Input: 0, Output: 1): Questo nodo produce in output l’indirizzo Ip pubblico su Internet del dispositivo.
- node-red-contrib-threshold (Input: 0, Output: 1): Questo nodo restituisce la soglia utile al calcolo di un Earthquake Early Warning.

- node-red-contrib-alive (Input: 0, Output: 1): Questo nodo segnala che il dispositivo è attivo al momento.
- node-red-contrib-timesync (Input: 0, Output: 1): Questo nodo sincronizza il timer del dispositivo.
- node-red-contrib-timereq (Input: 0, Output: 1): Questo nodo aiuta il precedente nella sincronizzazione.
- node-red-contrib-quake (Input: 0, Output: 1): Questo nodo produce in output i valori di una vibrazione del dispositivo sopra la soglia.
- node-red-contrib-disconnect (Input: 0, Output: 1): Questo nodo restituisce il segnale inviato quando un dispositivo viene disconnesso.
- node-red-contrib-reboot (Input: 1, Output: 1): Questo nodo segnala il riavvio del dispositivo.
- node-red-contrib-temperature (Input: 0, Output: 1): Questo nodo restituisce il valore che indica la temperatura del dispositivo in °C.
- node-red-contrib-probespeed (Input: 0, Output: 1): Questo nodo setta la velocità della sonda in Hz.
- node-red-contrib-sigma (Input: 0, Output: 1): Questo nodo restituisce il valore `sigma` (utile al rilevamento di possibili scosse).
- node-red-contrib-stream (Input: 0, Output: 1): Questo nodo restituisce (`on` / `off`) l'impostazione di ricezione dati dal server.
- node-red-contrib-streamdata (Input: 0, Output: 1): Questo nodo restituisce in output i valori di tutte le vibrazioni rilevate (sia sopra che sotto la soglia) in formato `x;y;z;timestamp` dove x, y e z sono i rispettivi valori nei tre assi.

Tutti questi dati vengono inviati e ricevuti tramite il protocollo MQTT dal server e dal client. La maggior parte delle informazioni di configurazione non cambia, quindi sono stati creati per la maggiore parte di essi (tranne reboot e alive, spiegata in seguito la motivazione) nodi che funzionano allo stesso modo. Dato che Node-RED utilizza NodeJS come runtime, il linguaggio utilizzato è Javascript e la libreria che aggiunge funzionalità riguardanti MQTT è **MQTT.js**. La scelta è motivata dal codice open-source, dalla attiva community di utenti e sviluppatori e dalla maturità del progetto [19]. Per la seguente spiegazione viene preso come esempio **node-red-contrib-rssi**. Il nucleo delle funzionalità è all'interno del file Javascript, in questo caso `rssi.js`. Dalla riga 1 alla riga 5 viene creato il nodo con le configurazioni di base (config) e vengono istanziate alcune variabili per facilità di lettura.

```
1 module.exports = function (RED) {
2   function RssiNode(config) {
3     RED.nodes.createNode(this, config);
4     var node = this;
5     var seismometer = config.seismometer
```

Qui dalla riga 39 alla 52 viene creato effettivamente il client MQTT e settato il topic. In questo caso il valore `seismometer` è il codice identificativo del sismometro. Viene poi creata una funzione callback in risposta ad un evento di connessione: quando il client MQTT si connette si deve iscrivere al topic selezionato, se il tutto va a buon fine viene loggata l'informazione, altrimenti stampato l'errore.

```
39      //===== Connect the client to the server and  define the topic =====
40      var client = mqtt.connect(mqtt.IClientOptions);
41      var topic = 'sensor/' + seismometer + '/rssi';
42
43      //====== on connect event, subscribe to the topic =====
44      client.on('connect', function () {
45          client.subscribe(topic, function (err) {
46              if (!err) {
47            node.log("Client succesfully subscribed to " + topic + " !");
48          } else {
49            console.log(err)
50          }
51        })
52      });
```

Dalla riga 60 in poi vengono definite tutte le altre funzioni callback per gli altri eventi: quando viene ricevuto un messaggio, quando avviene un errore di connessione dati MQTT o un errore della configurazione del nodi e quando il nodo viene distrutto dall'utente. Infine viene aggiunto alle API runtime di Node-RED il nodo `RssiNode` con il nome di *rssi*.

```
60          //===== on message event, log on console and send msg =====
61          client.on('message', function (topic, message) {
62            node.log("Message received from " + topic + ":" + message.toString());
63            var msg = { payload: message.toString() };
64            node.send(msg);
65          });
66
67          //======= on error event, log on console ========
68          client.stream.on('error', function () {
69            node.log("MQTT ERROR on " + topic);
70          });
71
72          //== on error event, log error on console and display error status ==
73          node.on("error", function (err) {
74            node.status({ fill: "red", shape: "ring", text: "node-red:common.status.
    error" });
75            node.error(err.toString());
76          });
77
78          //======== on close event, log on console and destroy client ========
79          node.on('close', function () {
80            client.end()
81            node.log("Client succesfully unsubscribed by " + topic + " !");
82          });
83        });
84      });
85    }
86    RED.nodes.registerType("rssi", RssiNode);
87  }
```

Il codice mostrato sopra definisce le funzionalità del nodo, per impostare il layout abbiamo bisogno di un file HTML, in questo caso **rssi.html**. In questo snippet di codice viene definito il menù a tendina che permette di scegliere tra i vari sismometri del gruppo il sismometro selezionato.

```
45 <script type="text/html" data-template-name="rssi">
46     <div class="form-row">
47         <label for="node-input-name"><i class="icon-tag"></i> Name</label>
48           <input type="text" id="node-input-name" placeholder="Name">
49     </div>
50     <div class="form-row">
51     <label for="node-input-seismometer"><i class="icon-tag"></i> Seismometer</
       label>
52         <select id="node-input-seismometer">
53         </select>
54     </div>
55 </script>
```

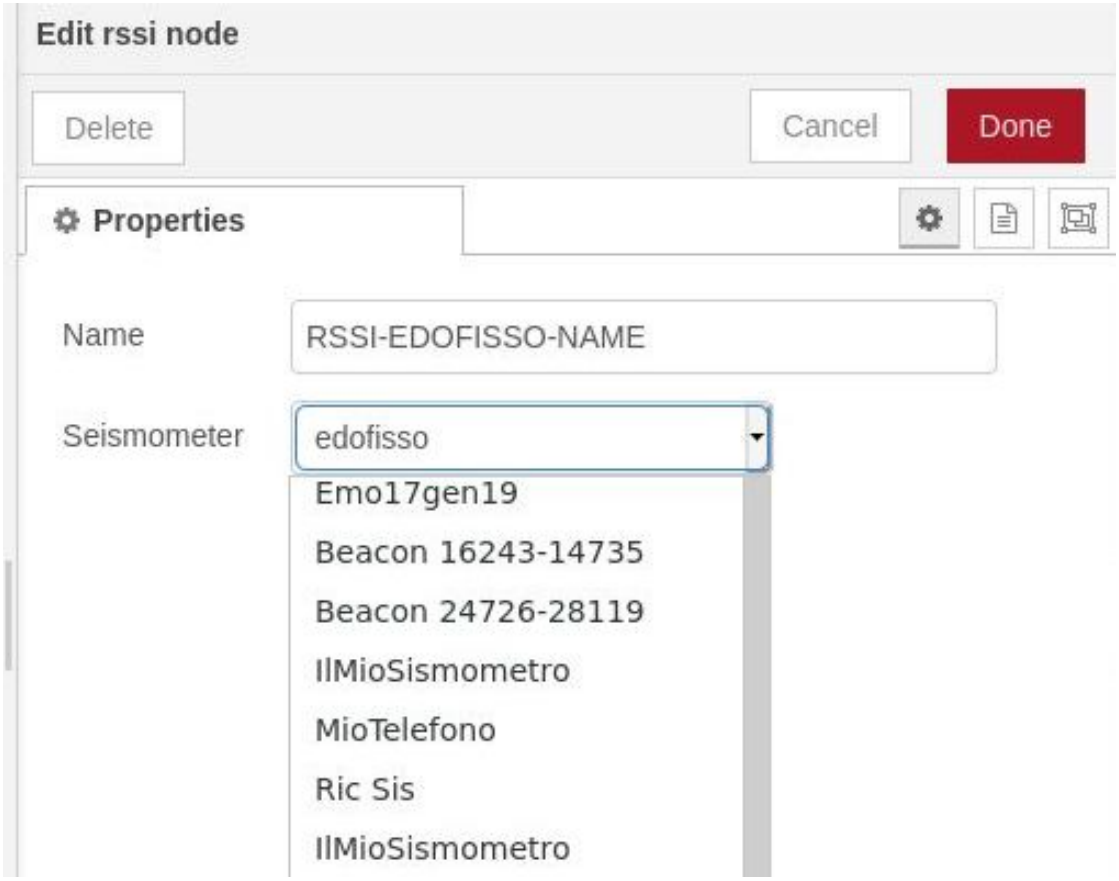
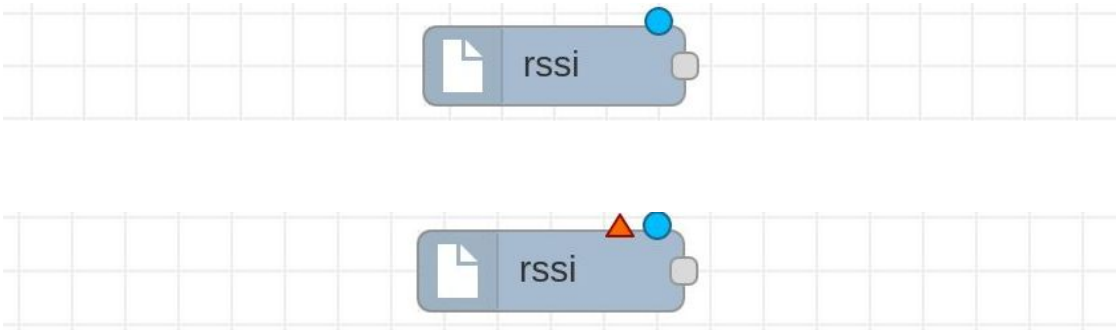


Figura 3.1: Dati necessari per la configurazione di un nodo SeismoCloud

Tramite il file `package.json` invece vengono definite impostazioni di base per il modulo e dipendenze esterne.

```
{
  "name": "red-contrib-rssi",
  "version": "1.0.0",
  "description": "This node returns periodically information about Received signal
     strength indication. INPUT: 0 | OUTPUT: 1 (string)",
  "main": "rssi.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {
    "mqtt": "3.0.0",
    "url-parse": "1.4.7"
  },
  "node-red": {
    "nodes": {
      "rssi": "rssi.js"
    }
  },
  "author": "edoardottt",
  "license": "ISC"
}
```



Le figure mostrate qui sopra raffigurano due nodi rssi inseriti nel workspace. Entrambi hanno un cerchio azzurro in alto a destra, mentre solo il secondo ha anche un triangolo arancione. Il cerchio blu sta ad indicare che ancora non sono state salavate le modifiche effettuate al flusso, mentre il triangolo arancione indica la non corretta (o mancante) configurazione del nodo. Per configurare il nodo bisogna definire il sismometro a cui il nodo si riferisce, è necessario un doppio click sinistro sul nodo, comparirà il menu laterale (figura 3.1) che consente di selezionare uno tra i dispositivi nel gruppo. Il cerchio blu è inserito di default nell'interfaccia da Node-RED, l'altro segnale è stato aggiunto successivamente per aiutare l'utente nel processo di creazione dei flussi.

I nodi che hanno un compito speciale sono **Alive** e **Reboot**. Il nodo Alive quando viene caricato dal runtime esegue una chiamata alle API di SeismoCloud per ottenere tutti i dati dei sismometri che sono registrati nel gruppo selezionato. Quest'ultimi (nome del dispositivo, identificativo alfanumerico, latitudine e longitudine, timestamp di ultima attività, tipo e versione) verranno poi messi a disposizione ed utilizzati dagli altri nodi e dal runtime stesso per eseguire dei controlli e per settare tutte le informazioni che non devono essere inserite dall'utente proprio perché il sistema ne è già a conoscenza. Il nodo reboot invece quando riceve in input qualcosa

(non ci sono controlli, può essere una stringa, un numero, in generale un valore qualsiasi) pubblica un messaggio (in questo caso un semplice "y") sul topic dedicato al sismometro selezionato per effettuare il riavvio dello stesso.

```
80      //== on input event, publish a message on this topic ==
81      node.on('input', function (msg) {
82          var message = "y"
83          client.publish(topic, message, function (err) {
84              if (!err) {
85              node.log("Client succesfully published to " + topic + " !");
86              var msg = { payload: "reboot" };
87              node.send(msg);
88            }
89          })
90      });
```

Implementazione di nodi che utilizzano API SeismoCloud per statistiche e informazioni sul sistema

- node-red-contrib-get-seismometer-info (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce le informazioni del dispositivo selezionato.
- node-red-contrib-list-earthquakes (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce la lista dei terremoti (di default avvenuti nelle ultime quattro settimane).
- node-red-contrib-get-all-seismometers-in-group (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce le informazioni di tutti i sismometri registrati nel gruppo di cui l'utente fa parte.
- node-red-contrib-get-weekly-ranking-for-all-devices-in-group (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce una classifica dei dispositivi più attivi nell'ultima settimana.
- node-red-contrib-general-statistics-for-groups (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce le statistiche generali per il gruppo di cui l'utente fa parte.
- node-red-contrib-weekly-statistics-for-groups (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce le statistiche per i gruppi nell'ultima settimana.
- node-red-contrib-get-activity-intervals-for-seismometers (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce i 100 intervalli di attività più recenti per il sismometro selezionato.
- node-red-contrib-get-statistics-for-active-seismometers-in-the-latest-minutes (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce le statistiche dei sismometri attivi negli ultimi cinque minuti.
- node-red-contrib-get-all-perceptible-earthquakes (Input: 1, Output: 1): Questo nodo quando riceve un input restituisce tutti i terremoti che potrebbero essere stati percepiti da uno o più sismometri all'interno del gruppo di cui l'utente fa parte.

Per questi nodi la parte di configurazione nel file `package.json` e nel file HTML non cambia molto, ma cambia completamente il file Javascript che definisce le funzionalità. In questo caso (`list-earthquakes.js`) ogni volta che il nodo riceve un input viene effettuata una richiesta HTTPS al server SeismoCloud e successivamente viene restituita in output la risposta.

```
20    node.on('input', function (msg) {
21      const https = require('https');
22      const URL = "/earthquakes/"
23      var apiServerUrl = "API-SERVER-URL-HERE";
24      var access_token = "ACCESS-TOKEN-HERE";
25
26      //HEADERS
27      const options = {
28        headers: {
29          'Host': apiServerUrl,
30          'Connection': 'keep-alive',
31          'X-Access-Token': access_token,
32        }
33      }
34      //EFFECTIVE GET REQUEST
35      https.get("https://" + apiServerUrl + URL, options, (resp) => {
36        var data = '';
37        // A chunk of data has been received.
38        resp.on('data', (chunk) => {
39          data += chunk;
40        });
41        // The whole response has been received. Send as msg.payload.
42        resp.on('end', () => {
43          node.log("Retrieved list-earthquakes");
44          var msg = { payload: data };
45          node.send(msg);
46        });
47      });
48    });
```

Figura 3.2: Nodi custom SeismoCloud per informazioni generali della rete

4. Sicurezza e protezione dei dati

4.1 Sicurezza nella piattaforma Node-RED

Come viene scritto chiaramente nella documentazione di Node-RED [20], non è sicuro di default. Vengono espressi e chiariti dei punti su come risolvere alcuni problemi di accesso. Viene anche detto però che questo è sufficiente solo se viene effettuato il deploy su un rete fidata, non il nostro caso quindi, ma comunque sono punti necessari. Quando il server viene avviato ovviamente viene esposto il servizio pubblicamente su Internet e quindi chiunque può accedere all'interfaccia. Innanzitutto Node-RED utilizza il protocollo HTTP, il quale invia i dati in chiaro che possono essere letti da chiunque sta sniffando il traffico. La documentazione suggerisce di utilizzare il protocollo **HTTPS**, il quale utilizza il protocollo TLS che cifra i dati sui canali di comunicazione. Per attivare quest'ultimo ci si deve munire di una chiave privata e di una catena di certificati in formato PEM. Successivamente dobbiamo inserire all'interno del file `settings.js`, questo fa si che i messaggi inviati tramite il protocollo HTTP vengano ora cifrati e spediti in modo sicuro.

```
1 https: {
2     key: require("fs").readFileSync('privkey.pem'),
3     cert: require("fs").readFileSync('cert.pem')
4 },
```

Successivamente la documentazione consiglia di inserire un metodo di autenticazione basato su username e password per gestire i vari privilegi degli amministratori, degli utenti e di chi non ha accesso al sistema. Devono essere create delle credenziali associate ad una classe di privilegi che vengono poi salvate nel file settings.js. Dato che l'interfaccia della dashboard personale di SeismoCloud è in fase di aggiornamento, la pagina che gestisce l'accesso alla parte privata di Node-RED è lasciata ad un lavoro futuro (Sottocapitolo 5.3). Quindi, da qui in poi si assume che la parte di autenticazione è sicura e dentro al sistema sono presenti solamente utenti SeismoCloud e non utenti non riconoscibili rispettando la buona pratica del **Principio del privilegio minimo**. Ovviamente è una forte assunzione, ma il sistema Node-RED in questo caso viene utilizzato in un modo non previsto, perciò nasconde molti errori di sicurezza che verranno esaminati meglio ed in dettaglio nei prossimi sottocapitoli.

4.2 Ricerca ed analisi di vulnerabilità

L'immagine Docker di Node-RED di default ha sei insiemi di nodi inseriti nella palette: common, function, network, sequence, parser, storage. A queste si sono aggiunte due categorie particolari per SeismoCloud: apiSeismoCloud e SeismometerData. Alcuni nodi già installati all'interno dell'intefaccia sono stati considerati pericolosi. Come già anticipato, gli utenti devono superare una fase di autenticazione, ma questi nodi vengono comunque considerati pericolosi per l'uso che ne potrebbe anche un utente di SeismoCloud. L'autenticazione certifica che l'utente appartiene effettivamente ad un gruppo, ma questo non esclude cattive intenzioni dello stesso. Ad esempio un nodo che è stato considerato pericoloso è il nodo **exec**. Questo nodo ha un parametro in input e restituisce tre output (Standard output, standard error e return code). Come si evince dal nome esegue comandi di sistema. In figura è riportato un esempio di uso cattivo del nodo exec. In realtà però, ci sono molti scenari più pericolosi, come ad esempio l'utilizzo della macchina su

cui viene hostato il server come mittente dell'attacco e non come destinatario. Questo nodo fa parte della categoria function, l'unico considerato pericoloso di questa categoria. Gli altri sono: udp, tcpin, websocket, httprequest, httpin, mqtt, httpproxy, tls, tail, file, watch. Si nota bene che molti sono protocolli di rete e sono stati rimossi per lo stesso motivo sopra. Possono essere utilizzati in modo maligno per fare richieste non consentite o effettuare qualche azione malevola con la rete attraverso il server. Gli altri (tail, file e watch) sono nodi della categoria storage e si occupano di leggere, modificare e salvare file. Anche questi nodi possono essere utilizzati da malintenzionati per leggere file sul server o creare codice da utilizzare in seguito.

Abbiamo visto che ci sono dei nodi che possono essere pericolosi se lasciati a completa disposizione dell'utente; togliendo questi nodi dalla palette, è sicura l'interfaccia? No.

Infatti, Node-RED mette a disposizione nella sezione di impostazioni una funzionalità molto utile (ma pericolosa per il nostro scopo) che consente all'utente di installare nodi esterni che sono disponibili nel contenitore di repository pubblico (flows.nodered.org). In questo modo si può ad esempio installare nodi come quello mostrato in figura che consente di eseguire codice Javascript. Un altro problema potrebbe essere il **typo-squatting**. L'utente potrebbe voler scaricare ed installare un nodo utile, ad esempio *node-red-contrib-alexa*, ma sbaglia a digitare premendo una c al posto di una x, quindi il sistema gli suggerisce il nodo *node-red-contrib-aleca*, che invece potrebbe essere un nodo malevolo (chiunque può caricare nodi sullo store ufficiale).

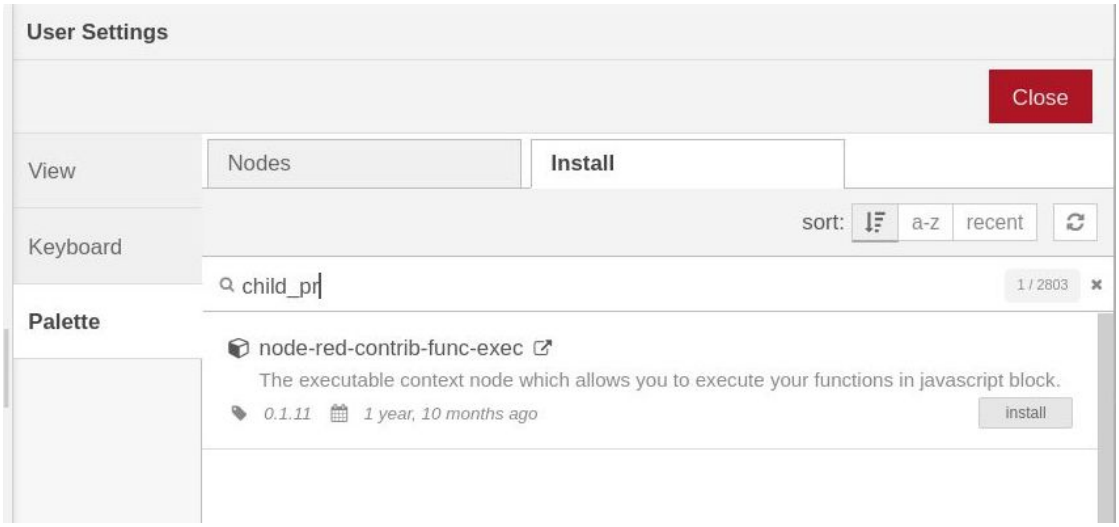


Figura 4.1: Installazione di un nodo esterno

Per quanto riguarda il nodo *function*, che appunto consente lo stesso di eseguire codice Javascript, è statto attentamente analizzato e viene ritenuto meno pericoloso perché viene utilizzato

il modulo *vm* (di NodeJS) per creare una sandbox ed eseguire il codice in maniera sicura [21]. Questa funzionalità nasconde comunque ancora altre criticità che verranno mostrate in maniera dettagliata in seguito.

Come abbiamo visto nel sottocapitolo 3.3, la procedura per comunicare con i sismometri ed il sistema SeismoCloud è cambiata radicalmente favorendo l'usabilità e l'accessibilità a persone non tecnicamente competenti. I nodi che sono stati creati per comunicare attraverso il protocollo MQTT con SeismoCloud richiedono solamente il nome del nodo ed il nome del sismometro. Durante la ricerca degli errori si è scoperto un errore banale ma "fastidioso". Era possibile infatti inserire un nome di un sismometro che non esisteva nel gruppo, così facendo veniva comunque creato un client MQTT e veniva messo in ascolto su un topic non esistente. Ovviamente la soluzione è di seguire sempre il mantra **mai fidarsi dell'input dell'utente**. La risoluzione viene dettagliata nel prossimo sottocapitolo.

Per una veloce configurazione e deploy dei servizi si era scelto di utilizzare le variabili d'ambiente dei container Docker. In questo modo con poche parole chiave era possibile settare i container con i giusti parametri per i differenti gruppi. Però alcune delle variabili settate sono **dati sensibili**, ad esempio hostname, username e password del server MQTT. Queste informazioni possono essere lette dall'interfaccia utente attraverso l'utilizzo del nodo Inject, oppure inserendo qualche riga di codice Javascript nel nodo Function.

```
1  msg.payload = "";
2  msg.payload = msg.payload + "API SERVER URL: " + env.get("API_SERVER_URL") + "
   \n";
3  msg.payload = msg.payload + "MQTT SERVER: " + env.get("MQTT_SERVER") + "\n";
4  msg.payload = msg.payload + "MQTT USER: " + env.get("MQTT_USER") + "\n";
5  msg.payload = msg.payload + "MQTT PASS: " + env.get("MQTT_PASS") + "\n";
6  msg.payload = msg.payload + "MQTT PINNING CERT: " + env.get("MQTT_PINNING_CERT
   ");
7  return msg
```

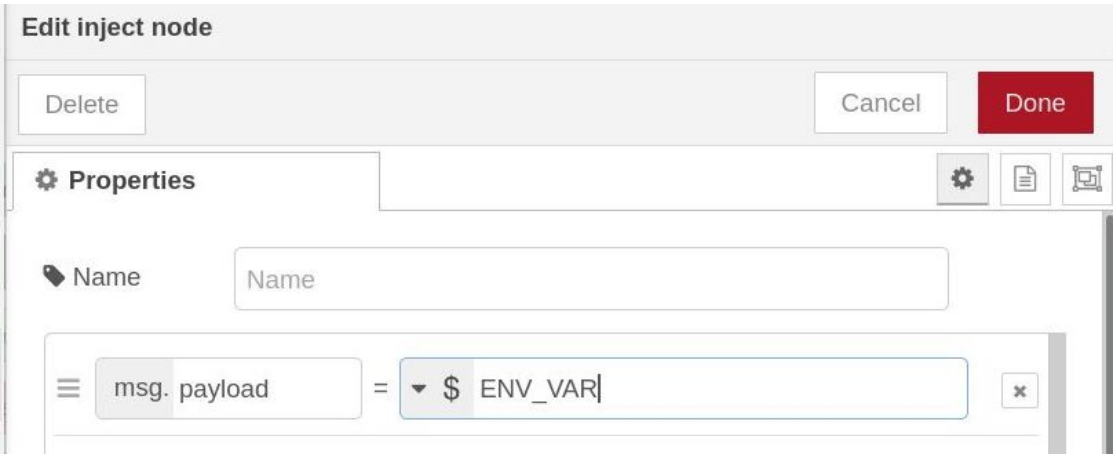


Figura 4.2: Il nodo Inject può restituire in output i valori delle variabili d'ambiente

Come è stato accennato nel capitolo 3, sono disponibili alcuni modi per inviare messaggi tra i vari nodi, uno di questi è la condivisione di variabili con scope differente. Ci sono diversi scope: node (visibili solo al nodo), flow (visibili solo ai nodi nello stesso flusso) e global (disponibili a tutto il sistema). Come si legge dallo studio *Securing IoT apps in Node-RED* di Olsson

[20], la quasi totalità dei nodi non utilizza variabili con scope globale o di flusso. Questo è un bene perché l'utente si aspetta che i dati fluiscano in correlazione ai flussi visuali creati e non vengano scambiati tra i nodi in modo arbitario dalle azioni create. Comunque, l'interfaccia da la possibilità di leggere tutte le variabili d'ambiente disponibili (incluse quelle inserite per l'uso di SeismoCloud). Tra queste variabili sono presenti alcune che forniscono molte informazioni sul sistema in uso (ad esempio le varie versioni di Docker, Node-RED, npm, NodeJS...), sulla distribuzione dei file tra le varie cartelle e alcuni valori critici per la sicurezza (come ad esempio utente, password e hostname del server MQTT).

```
1  npm_config_user_agent: "npm/6.XX.X node/v12.XX.X linux x64"
2  NODE_VERSION: "12.XX.X"
3  npm_config_bin_links: "true"
4  HOSTNAME: "1cs453a053a1"
5  ACCESS_TOKEN: "ACCESS-TOKEN"
6  YARN_VERSION: "1.22.4"
7  npm_node_execpath: "/usr/local/bin/node"
8  NODE_RED_VERSION: "v1.X.X"
9  SHLVL: "1"
10 HOME: "/usr/src/node-red"
11 npm_config_editor: "vi"
12 npm_config_rollback: "true"
13 npm_config_tag_version_prefix: "v"
14 npm_config_cache_max: "Infinity"
15 npm_config_userconfig: "/usr/src/node-red/.npmrc"
16 npm_config_tmp: "/tmp"
17 npm_config_package_lock: "true"
18 GROUP_ID: "ID_46355245"
19 MQTT_PASS: "MQTT-PASS"
20 npm_config_sso_type: "oauth"
21 npm_config_shell: "bash"
22 npm_config_format_package_lock: "true"
23 npm_config_prefix: "/usr/local"
24 API_SERVER_URL: "API-SERVER"
25 MQTT_SERVER: "MQTT-SERVER"
26 npm_config_cache: "/usr/src/node-red/.npm"
27 npm_package_scripts_start: "node $NODE_OPTIONS node_modules/node-red/red.js $FLOWS
       "--userDir" "/data" "npm" "start""
28 npm_config_viewer: "man" "/usr/local/lib/node_modules/npm/node_modules/node-gyp/
       bin/node-gyp.js"
29 PATH: "/usr/local/lib/node_modules/npm/node_modules/npm-lifecycle/node-gyp-bin:/
       usr/src/node-red/node_modules/.bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
       usr/bin:/sbin:/bin"
30 npm_config_audit_level: "low"
31 NODE: "/usr/local/bin/node"
32 MQTT_USER: "MQTT-USER"
33 npm_config_maxsockets: "50"
34 npm_config_sso_poll_frequency: "500"
35 npm_package_main: "node_modules/node-red/red/red.js"
36 npm_config_loglevel: "notice"
37 npm_lifecycle_script: "node $NODE_OPTIONS node_modules/node-red/red.js $FLOWS "--
       userDir" "/data" "npm" "start""
38 NODE_PATH: "/usr/src/node-red/node_modules:/data/node_modules"
39 npm_lifecycle_event: "start"
40 npm_config_argv: "{"remain":["--userDir","/data","npm","start"],"cooked":["start",
       "--","--userDir","/data","npm","start"],"original":["start","--","--userDir","
       /data","npm","start"]}"
41 npm_config_searchlimit: "20"
42 npm_config_unsafe_perm: "true"
43 npm_config_update_notifier: "true"
44 npm_config_auth_type: "legacy"
```

```
npm_config_node_version: "12.XX.X"
npm_config_git_tag_version: "true"
npm_config_commit_hooks: "true"
npm_config_shrinkwrap: "true"
npm_package_license: "Apache-2.0"
npm_config_fetch_retry_factor: "10"
npm_config_strict_ssl: "true"
npm_config_globalconfig: "/usr/local/etc/npmrc"
npm_config_init_module: "/usr/src/node-red/.npm-init.js"
FLOWS: "/storage/flows.json"
npm_config_globalignorefile: "/usr/local/etc/npmignore"
npm_execpath: "/usr/local/lib/node_modules/npm/bin/npm-cli.js"
PWD: "/usr/src/node-red"
affinity:container: "=2e20e0f42d676f2f311aa1b07d65yw1r696f174ec9bf8e0"
INIT_CWD: "/usr/src/node-red"
NODE_RED_HOME: "/usr/src/node-red/node_modules/node-red"
```

Un ulteriore problema di sicurezza sono i canali di comunicazione. Abbiamo visto che Node-RED consiglia di utilizzare un certificato e una chiave per utilizzare HTTPS. Però anche se la connessione è cifrata, questo non vuol dire che nessun malintenzionato può comunicare con il server attraverso le API. Quindi si necessita di un metodo di autenticazione anche per le richieste verso il server API.

Inoltre, è stato argomentato come utilizzare HTTPS per cifrare i dati senza usare il meno sicuro HTTP, ma il sistema utilizza MQTT che invia dati in chiaro sui canali di comunicazione. Per una connessione più sicura è necessario un metodo di cifratura dei messaggi anche sul protocollo MQTT.

4.3 Risoluzione dei problemi

Come è stato descritto nel precedente sottocapitolo, un problema di sicurezza sono i nodi evidenziati come possibili vettori di attacco (nodi appartenenti alla categoria storage, che si occupano di leggere, modificare e creare file, network, che contiene nodi che utilizzano vari protocolli per comunicare sulla rete ed il nodo exec, unico della categoria function). La soluzione in questo caso è stata la rimozione degli stessi dall'interfaccia utilizzando le impostazioni nel file `settings.js`.

```
//EXCLUDING NODES FROM PALETTE:
  nodesExcludes: ['32-udp.js', '31-tcpin.js', '22-websocket.js', '21-httprequest.
    js', '21-httpin.js', '10-mqtt.js', '06-httpproxy.js', '05-tls.js', '90-exec.js
    ', 'node-red-node-tail', '10-file.js', '23-watch.js'],
```

Ovviamente è un grande compromesso tra la sicurezza del sistema e le funzionalità offerte. Questi nodi consentivano una grande (sproporzionata) libertà all'utente di comunicare con la rete. Potevano essere effettuate richieste verso la maggior parte dei servizi web avendo a disposizione molti protocolli (mqtt per i dispositivi IoT, http per i servizi Web), comunque necessitano una elevata conoscenza dei protocolli e delle reti, quindi poco utilizzabili per gli utenti SeismoCloud non esperti. Durante i prossimi sviluppi potrebbero essere utili dei rilasci di copie di questi nodi, che quindi imitano le loro funzionalità, ma adattati per eseguire richieste in sicurezza e per essere utilizzati senza una eccessiva configurazione.

Purtroppo però è possibile, anche eliminando questi nodi dall'interfaccia, emulare le loro funzionalità attraverso il download di nodi esterni. Ovviamente questa è una funzionalità molto utile in Node-RED per la maggior parte dei campi dove viene utilizzato (ad esempio nell'industria,

automatizzando processi. Invece di scrivere nuovo codice posso ottenere le stesse funzionalità già prodotte e collaudate dalla comunità Node-RED), mentre per il caso SeismoCloud è una vulnerabilità che consente di ottenere grandi privilegi sulla piattaforma. Ad esempio, si può scaricare un nodo che effettua una richiesta http, oppure un nodo che attraverso la libreria *child_process*, esegue un fork di un processo per eseguire codice Shell sulla macchina ospitante. Per bloccare ciò è stato inserito questo codice. Così facendo viene disabilitata questa funzionalità.

```
93      palette: {
94        editable: false // Enable/disable the Palette Manager
95      },
```

Riguardo l'accesso ai vari tipi di context (node, flow e global) nessuno dei nodi introdotti da SeismoCloud utilizzano questo metodo per scambiare informazioni. Come è stato scritto nella sezione precedente, neanche i nodi che troviamo in SeismoCloud di default, quindi nessun nodo presente nella palette ne fa uso. Tuttavia, alcuni dati che vengono utilizzati dal backend NodeJS, dal package manager NPM e da Docker vengono comunque caricati nel global context per facilità d'uso. Questo può portare, come si è precedentemente visto, ad esporre tante informazioni sul sistema e sulle variabili d'ambiente. È possibile nascondere questo dettaglio inserendo nel file di impostazioni queste modifiche: si elimina (o commenta come in questo caso) la riga che setta ed esporta la variabile `env` che da accesso alle variabili d'ambiente e si setta a falso l'impostazione che gestisce l'esportazione delle chiavi del global context.

```
56  functionGlobalContext: {
57      // env: process.env
58    },
59  exportGlobalContextKeys: false,
```

Con questa modifica viene disabilitata la funzione `global.keys()` che permetteva di elencare tutte le variabili d'ambiente in un solo messaggio. Ovviamente, conoscendo il nome delle variabili d'ambiente che utilizza Node-RED (ma anche npm, NodeJS e Docker) era facile anche senza l'utilizzo di questa funzione.

Comunque, rimane un problema fondamentale: il nodo Inject può comunque stampare i valori delle variabili di sistema (che sappiamo contenere dati sensibili). Per non rinunciare alla comodità d'uso e facilità di deploy dei servizi, le variabili di sistema vengono gestite in maniera differente dal solito. Questo modulo (chiamato appunto *readEnv*) viene caricato dai nodi che ne hanno bisogno, legge il valore della variabile d'ambiente desiderata e: a) se il primo carattere del valore è un @, allora il contenuto rimanente del valore indica la locazione nel sistema dove risiede il valore (e.g. /secret-values/password.txt), ne legge il contenuto e se non ci sono errori restituisce il valore, altrimenti blocca l'esecuzione del programma e stampa nei log un errore. b) se il primo carattere del valore della variabile non è un @, allora legge l'intero contenuto della variabile e lo restituisce. In questo modo, anche se un utente riesce a stampare nell'interfaccia i valori delle variabili di sistema riuscirà a leggere i veri valori per tutte le variabili non pericolose, mentre leggerà il percorso del file in cui risiede il vero valore nell'altro caso. Se questi file vengono posizionati in un'unica cartella all'interno della stessa di Node-RED, questo fa si che l'utente non potrà avere nessuna informazione sulla reale struttura dei file nel backend.

```
1  // Read the environment variable or the associated file.
2  // ========
3
4  module.exports = {
```

```
  get: function (key) {
    var envVar = process.env[key];
    var fs = require("fs");
    // external file
    if (envVar[0] == "@") {
      try {
        var value = fs.readFileSync(envVar.substr(1));
      } catch (err) {
        console.error('ERROR: ', err)
        process.exit(1)
      }
      return value
    }
    // otherwise direct value
    return envVar
  }
};
```

Per il controllo dell'input da parte dell'utente è stato semplicemente inserito un controllo che verifica che l'input inserito corrisponda ad un dato effettivamente presente nel sistema. Viene effettuata una chiamata alle API per ottenere i dati sugli utenti aggiornati in tempo reale. La risposta della chiamata API contiene l'identificativo alfanumerico del dispositivo, latitudine, longitudine, nome, timestamp di ultima attività, tipo, modello e alivetime. Per un utente non esperto è difficile modificare il menu a tendina precompilato con tutti i sismometri disponibili (non si può modificare con il semplice click), per un utente competente in materia, basta aprire la console di debug di un qualsiasi Browser e modificare il codice della pagina Web. Se l'utente non inserisce un sismometro valido, quindi molto probabilmente ha cambiato i valori tramite console, viene stampato nei log un segnale di warning e ovviamente l'azione non viene effettuata dal server.

```
    // The whole response has been received. Print out the result.
      resp.on('end', () => {
        var obj = JSON.parse(data)
        for (i in obj) {
          devices.push(obj[i])
        }
        var deviceIdOk = false // this indicates if the entered input is in the
    group seismometers
        for (i in devices) {
          if (devices[i]['ID_Device'] == seismometer) {
            deviceIdOk = true
            break
          }
        }

        // Only if the entered input is in the known seismometers' id
        if (deviceIdOk) {
```

Nel sottocapitolo 3.3 è stato definito il comportamento e le funzionalità dei nodi che utilizzano le API di SeismoCloud per accedere a dati della rete di sismometri. Per l'applicazione Android e iOS viene utilizzato un altro metodo di sicurezza, qui si è scelto di aggiungere alle richieste HTTP un token di accesso per verificare che la richiesta provenga effettivamente da un client autorizzato a ricevere ed utilizzare quei dati.

Un problema che non è stato descritto nella ricerca di vulnerabilità è la possibile esposizione della posizione geografica degli utenti nel sistema. In realtà il problema non sussiste perché di

default il sistema SeismoCloud ha l'opzione di conservazione della privacy attiva, quindi la posizione che in realtà viene inviata è una posizione effimera, cioè randomizzata in una raggio di due kilometri avente il centro nella reale posizione. Questo fa si che la reale posizione dei sismometri non viene esposta, ma comunque l'algoritmo di Eartquake Early Warning non ne risente dato che i terremoti interessano vaste porzioni di territorio e pochi kilometri di randomizzazione non hanno effetto sul risultato prodotto.

I dati che vengono inviati e ricevuti tramite il protocollo MQTT sono scambiati senza nessun protocollo di cifratura. Il client MQTT prende i dati, li incapsula in un pacchetto TCP/IP e lo invia. I dati inviati in questo modo possono essere intercettati da tutti i componenti dell'infrastruttura Internet in cui passa il pacchetto, oppure possono essere letti/modificati da un intruso nella rete, eseguendo appunto un attacco "Man in the middle". TLS (Transport Security Layer) è un protocollo che cifra i dati e li invia in maniera sicura, lo stesso che viene utilizzato per cifrare i dati da HTTP in HTTPS. Così facendo chiunque prova a ottenere i pacchetti inviati non leggerà il reale contenuto di essi, ma dei caratteri che non hanno senso insieme, non potendo così ottenere nessuna informazione. È altamente consigliato utilizzare MQTT con l'aiuto di TLS, infatti la porta 8883 è lo standard per questo tipo di connessione. IANA (Internet Assigned Numbers Authority) lo ha chiamato "secure-mqtt".

Dato che il sismometro non ha una memoria cosi grande e che questa azione comporterebbe uno sforzo non indifferente, viene utilizzato il certificate pinning anziché una catena di certificati come di solito accade. Viene passato come parametro al container Docker un certificato da considerare valido (o meglio, ovviamente viene passato in input un valore del tipo "@percorso-dove-trovare-il-certificato" per le motivazioni di sicurezza sopra esposte) ed un modulo esterno ha il compito di leggerlo e , nel caso non ci siano errori, configurare le opzioni del client MQTT per instaurare una connessione MQTT over TLS.

```
module.exports = {
  setProtocol: function () {

    var mqtt = require('mqtt');
    var fs = require("fs");
    var parse = require("url-parse");
    var readEnv = require('./read-env');

    var certificate = readEnv.get("MQTT_PINNING_CERT");
    var mqtt_server = readEnv.get("MQTT_SERVER");
    var mqtt_user = readEnv.get("MQTT_USER");
    var mqtt_pass = readEnv.get("MQTT_PASS");
    var url = parse(mqtt_server, true);

    mqtt.IClientOptions = {
      host: url.hostname,
      username: mqtt_user,
      password: mqtt_pass,
      rejectUnauthorized: true,
      servername: url.hostname
    };
    // MQTT OVER TLS
    if (url.protocol == "tls") {
      // === no certificate ===
      if (certificate == "" || typeof certificate == "undefined") {
        //=== client configuration ===
        mqtt.IClientOptions.port = url.port
        mqtt.IClientOptions.protocol = url.protocol
      }
      // === certificate ===
      else {
```

```
32          //=== read certificate ===
33          try {
34            var CERT = fs.readFileSync(certificate);
35          } catch (err) {
36            console.error('ERROR: ', err)
37            process.exit(1) //mandatory (Node docs)
38          }
39          //=== client configuration ===
40          mqtt.IClientOptions.port = url.port
41          mqtt.IClientOptions.protocol = url.protocol
42          mqtt.IClientOptions.ca = [CERT]
43        }
44      }
45      // MQTT
46      else {
47        //=== client configuration ===
48        mqtt.IClientOptions.port = '1883'
49        mqtt.IClientOptions.protocol = 'mqtt'
50      }
51      return mqtt.IClientOptions
52    }
53  };
```

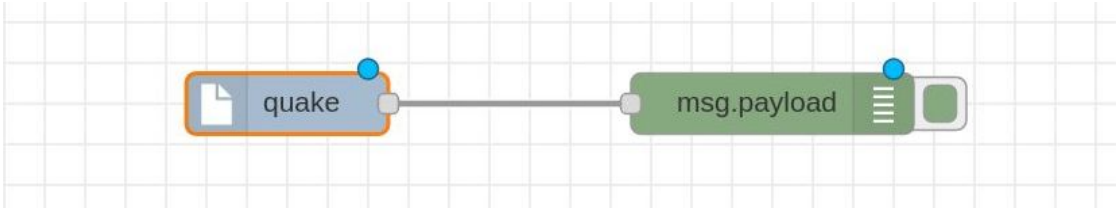
5. Test, conclusioni e sviluppi futuri

5.1 Iterazioni dei Test

Per una efficiente usabilità del sistema è assolutamente necessaria una parte accurata di test. In questo caso sono stati coinvolti 7 utenti non esperti ed un esperto. Questa scelta non è casuale, infatti tanti utenti non esperti servono a verificare che il sistema è adatto a persone che non hanno competenze informatiche elevate, mentre la valutazione dell'esperto serve ad assicurare che le buone pratiche di usabilità ed esperienza utente vengano rispettate. Per quanto riguarda invece i compiti che richiedono un background tecnico, l'applicativo è stato testato internamente nel team di sviluppo per più di qualche mese. Le modalità di test effettuati sono due: in *remoto* e *dal vivo*. Questa scelta è motivata da vantaggi e svantaggi per una o per l'altra scelta: da remoto si può controllare meglio tutta l'attività "digitale" dell'utente, ad esempio i movimenti del cursore, quindi banalmente dove si aspetterebbe di trovare alcune sezioni; con un test dal vivo invece si ha un maggior coinvolgimento da entrambe le parti, facendo così sentire l'utente più a suo agio e riuscendo a captare meglio le sue reazioni durante il test. Sono state utilizzate diverse tecniche: **think aloud** (all'utente viene chiesto di pensare a voce alta ed esprimere tutte le emozioni, sensazioni e pensieri che ha durante tutta la durata del test), **cooperative evaluation** (utente e somministratore del test si pongono domande a vicenda ed esprimono i loro giudizi sull'esperienza d'uso), **post-task walkthrough** (finito il test, indipendentemente dal tipo effettuato, vengono poste delle domande per comprendere i risultati. Si hanno due opzioni: 1) appena finito il test, dove l'utente ha ben chiare le sue opinioni, ma il somministratore non ha tempo di pensare alle domande, 2) il contrario del precedente) e **expert based** (un esperto revisiona l'interfaccia e da i suoi pareri su di essa) [17].

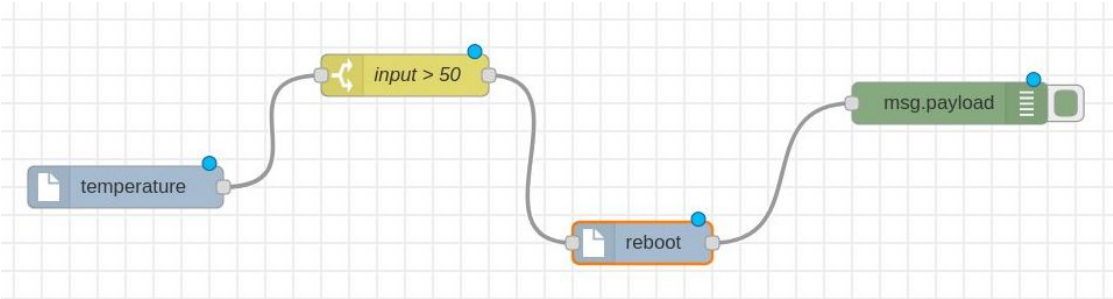
Sono stati effettuate due sessioni di test con una revisione e correzione degli errori trovati appena effettuata la prima delle due. I task scelti da svolgere per gli utenti erano quattro (con alcune variazioni durante i vari test per capire come reagivano gli utenti) di varia difficoltà per misurare fino a che punto il sistema risultasse utilizzabile per un utente non esperto:

- Difficoltà Livello 1: Ottenere il valore Quake del sismometro 'edotest' e stamparlo nella console di debug.

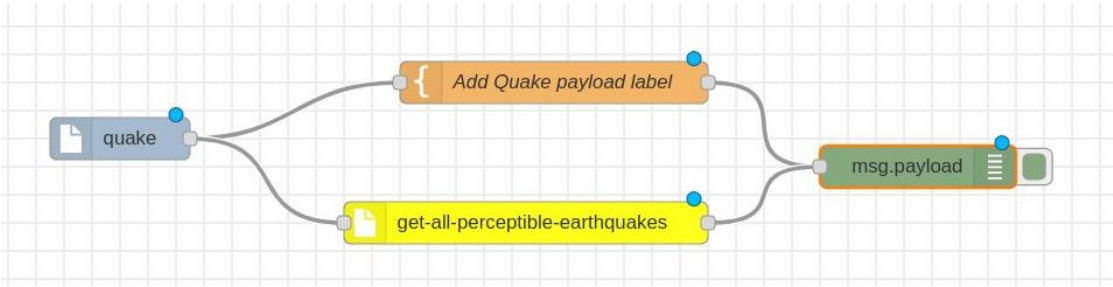


- Difficoltà Livello 2: Stampare tutte le informazioni del sismometro 'edotest' nella console di debug ogni volta che si riceve un segnale Quake.

- Difficoltà Livello 3: Effettuare il reboot del sismometro 'edotest' ogni volta che la temperatura è maggiore di 50°C.



- Difficoltà livello 4: Prendere il valore Quake del sismometro 'edotest', applicare ad esso un template (ad esempio "`Quake: <msg.payload>`") e stamparlo nella console di debug con le informazioni relative agli ultimi terremoti possibilmente rilevabili dal dispositivo selezionato.



Come detto appena sopra questi test sono solo dei formati utili per catalogarli in gruppi di varia difficoltà, le modifiche effettuate all'interno dei test sono ad esempio: rssi o altro nodo al posto del nodo Quake, interruzione del test per chiedere all'utente di spiegare in dettaglio la funzione di un nodo, oppure aprire il pannello di impostazioni per cambiare qualche configurazione.

Effettuata la prima serie di test, con circa 15 test individuali sono stati evidenziati i seguenti problemi:

- tasto di visualizzazione della console di debug non messo in risalto
- Prima clic destro su un nodo per la configurazione, solo dopo viene fatto un doppio clic
- Descrizione poco dettagliata dei valori del sismometro

- SeismometerAttributes è vago per indicare i dati relativi ai sismometri, meglio SeismometerData

Il primo problema nasce dal fatto che alcune funzionalità del sistema di End User Development per SeismoCloud non sono ancora state sviluppate. Perciò la disponibiltà di azioni è limitata a poche scelte. La console di debug una volta sviluppate o integrate (alcune funzionalità "finali", quindi di notifica utente o nel senso di azione finale, sono già fornite dai vari produttori di servizi Web o da alcuni volenterosi sviluppatori Open Source: Telegram, Twitter, Email, integrazione con oggetti per smart-home) le funzionalità finali sarà solo uno strumento di utilità per controllare appunto se ci sono errori nei flussi progettati. Per questo motivo non è stato ritenuto un errore da risolvere.

Il secondo problema è stato rilevato maggiormente durante i test da remoto, dove il controllo del cursore dell'utente è più intenso. Quando un utente doveva selezionare un nodo per configurare le impostazioni di base (nome del nodo, sismometro selezionato) 4 dei 7 utenti hanno in prima battuta provato un clic destro sul nodo, che non ha nessun effetto, e poi in seconda battuta un doppio clic sinistro, che è appunto l'azione corretta.

La terza criticità era la descrizione di alcuni nodi, per esempio, uno di questi era il nodo **BSSID**. Questo termine indica l'indirizzo MAC di un punto di accesso alla rete dove il sismometro si connette. Il problema è stato rilevato chiedendo (durante la tipologia di test *cooperative*

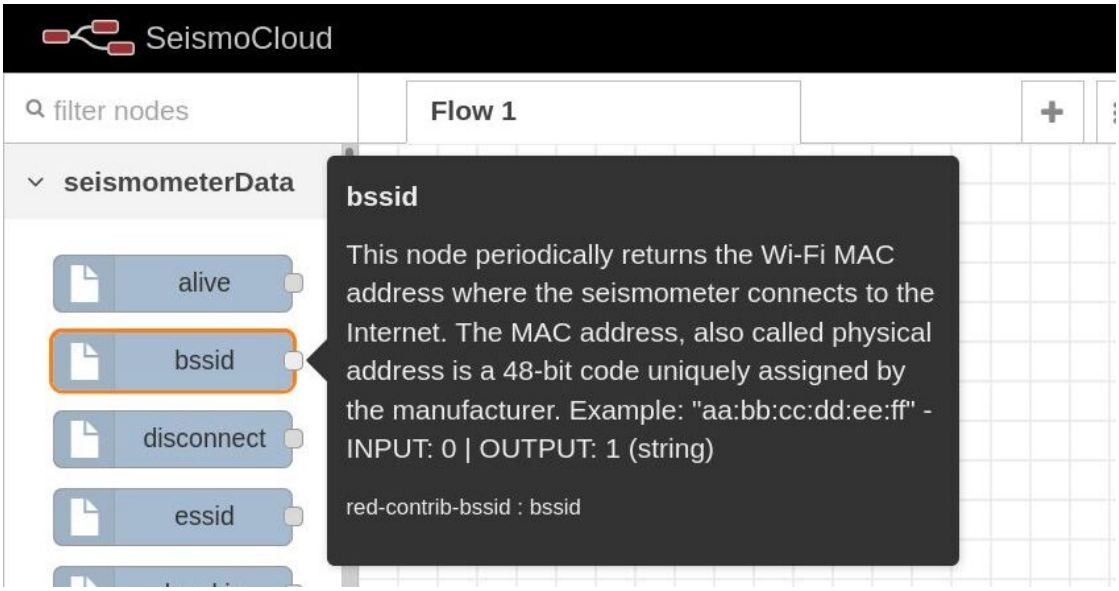


Figura 5.1: nodo BSSID corretto dopo la prima sessione di test

evaluation) agli utenti se, una volta configurato il nodo, avessero capito che tipologia di dati stavano usando, o quanto meno sapessero dare una spiegazione sommaria della funzione del nodo. Tutti gli utenti non competenti hanno risposto che il nodo restituisce un valore in output, ma che non avevano capito minimamente la tipologia di dato. La verifica si è estesa ai nodi meno comprensibili e che richiedono un background tecnico più forte, come ad esempio RSSI, StreamData oppure è stato richiesto di spiegare la differenza tra Local IP e Public IP. La risoluzione del problema è stato aggiungere dettagli alle descrizioni dei nodi che non hanno avuto esito positivo in questo test.

L'ultimo problema è stato rilevato dal somministratore dei test, dato che il titolo "SeismometerAttributes" non era adatto a descrivere in modo comprensibile le funzionalità coinvolte e soprattutto non entrava nel box ad esso dedicato.

La seconda sessione di test è stata effettuata dopo aver parzialmente risolto i problemi della prima. È stata aggiunta una descrizione molto più dettagliata per tutti i nodi di cui risultava ambiguo il significato ed è stato modificato il nome "SeismometerAttributes" con "SeismometerData". Le modifiche sono state sottoposte ai test che sono stati ripetuti sia cambiando gli utenti coinvolti, sia con gli stessi che avevano effettuato i test precedentemente. Le modifiche hanno avuto esito positivo, quindi i test non si sono concentrati su questo fattore, quanto più sulla creazione di flussi.

Tutti gli utenti hanno compreso come i nodi devono essere collegati per formare un flusso, prendendo e rilasciando i nodi dalla palette allo spazio di lavoro, molto utile in questo caso la texture del foglio di lavoro che imita un foglio a quadretti. Data la differenziazione in categorie per argomenti differenti (e colorati in modo differente), cercare un singolo nodo sulla palette è risultato un facile compito. Tutti gli utenti hanno completato i compiti di basso e medio livello senza problemi significativi, anche se ci è voluto più tempo del previsto per i compiti di alta difficoltà.

5.2 Conclusioni

Dai risultati che sono stati ottenuti si sta procedendo nella giusta direzione. È stato progettato ed implementato un sistema di End User Development **sicuro, efficiente e facilmente usabile**. Sono stati trovati i problemi più evidenti in sicurezza ed in usabilità; è stata applicata una soluzione efficace per alcuni, per altri è definito il metodo di risoluzione. Come si nota dalle implementazioni tecniche apportate descritte nel capitolo 3 e dai test svolti definiti in questo capitolo, l'interfaccia risulta usabile e richiede la minima configurazione dell'utente per programmare anche azioni complesse senza avere una conoscenza tecnica né di programmazione, né di protocolli di rete. L'applicativo è stabile, è stato testato più volte con gli utenti ed internamente nel team di sviluppo. Come si è visto nel capitolo 4 è anche sicuro e i dati della rete di sismometri e le informazioni personali degli utenti SeismoCloud rimangono protetti sia nel sistema, sia nella comunicazione tra le varie parti. È stata creata una piattaforma che sarà utilizzata come base con le funzionalità necessarie ad interfacciarsi con il sistema ed i vari dispositivi connessi. Viene definita "base" perché non sono stati sviluppati tanti nodi che consentono azioni che potrebbero essere utili per un utente. Il lavoro svolto fino ad ora ha prodotto un servizio pronto ad interfacciarsi con i vari componenti del sistema (sismometri, utenti e API SeismoCloud), mancano comunque delle funzionalità dette "finali" che consentano all'utente di venir notificato o performare altre azioni automatiche. Nel prossimo paragrafo quindi vengono espressi alcuni percorsi da seguire per aggiungere interessanti funzionalità e continuare ad elevare l'utilità di questo sistema. Si continuerà ad ascoltare gli utenti SeismoCloud ricevendo feedback utili e proposte di servizi cercando di far rimanere il sistema comprensibile, sicuro ed usabile.

5.3 Sviluppi futuri

Integrazione social network e dispositivi IoT per smart home

L'introduzione di azioni che utilizzano le API di social network o funzioni di notifica è molto utile. In particolare, esiste già un modulo NPM pronto e maturo (Versione 8.6.4) per Telegram [23], per Twitter [24] e per Facebook [25]. Sono molto utili per condividere dati, ma soprattutto per notificare ad un bacino molto ampio di utenza facile da raggiungere Earthquake Early Warning. In più, per quanto riguarda Twitter e Facebook, non è nemmeno necessario conoscere dati dei destinatari del messaggio, basta condividere un messaggio automatico con le informazioni utili. Ancora più utile è la possibilità di localizzare il proprio Tweet (o post Facebook) tramite la localizzazione GPS per una più rapida ed efficace diffusione dell stesso in aree interessate all'evento sismico. L'INGV ha avviato un servizio simile, questo un Tweet di esempio(https://twitter.com/ingvterremoti/status/1259681114920230912).

Come è stato introdotto nel capitolo 2.1, una funzionalità interessante sarebbe l'integrazione con dispositivi per una smart-home. Anche qui dato l'alto contributo della community esistono moduli pronti all'uso (necessaria configurazione). Ad esempio integrazione con Alexa [26] e Google Home [27]. Oltre ai soliti dispositivi smart (luci, elettrodomestici, TV...) l'utente ha completa libertà (privilegiando l'usabilità e la sicurezza) di creare da sé programmi software che si interfacciano con dispositivi hardware creati personalmente. Ad esempio, programmare un GPIO (General Purpose Input/Output) ad emettere un segnale quando avviene una certa condizione. Quest'ultimo poi potrà essere utilizzato per interfacciarsi con dispositivi che hanno bisogno di essere spenti o accesi in determinate condizioni di sicurezza (generatore elettrico, valvola del gas), dato che il GPIO ha solo due stati.

Ottimizzazione delle prestazioni tramite la condivisione di un solo client MQTT

Un ristretto limite prestazionale è dato dall'utilizzo di tanti client MQTT quanti sono i nodi presenti nel workspace. La libreria MQTT utilizzata offre la disponibilità di assegnare un identificativo univoco per ogni client. L'attuale implementazione istanzia un client MQTT (con un identificativo randomico con il formato

```
clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8))
```

ogni volta che viene creato un nodo, e viene distrutto quando il nodo viene cancellato dal workspace. In generale, data la brevità dei messaggi trasferiti e l'implementazione orientata all'efficienza del protocollo MQTT, l'applicativo non ne risente molto; ma con il crescere dei gruppi, degli utenti e quindi dei dispositivi coinvolti si potrebbero aver problemi di scalabilità, per questo un unico client condiviso con i nodi del gruppo (o flusso, dipende dalle connessioni simultanee possibili) potrebbe aiutare l'efficienza del sistema. Inoltre, con le variabili a differente scope (Node, Flow e Global definite nel sottocapitolo 2.2) è possibile condividere in modo facile lo stesso client per azioni differenti.

Metodo di accesso sicuro ed affidabile a SeismoCloud Node-RED

Il sistema Node-RED è stato inizialmente progettato per essere inserito in una rete affidabile e sicura, tipicamente all'interno di una rete locale protetta da sistemi di firewall. Nel corso del tempo il progetto è maturato e ha riscosso successo per l'elevate potenzialità dello stesso, quindi sono state sviluppate alcune tecniche per renderlo adatto in una rete pubblicamente accessibile. In questa sezione Wiki di Node-RED [28] vengono espressi alcuni suggerimenti per il deploy sicuro sulla rete pubblica. Viene consigliato di utilizzare sempre HTTPS per cifrare i messaggi che vengono scambiati sui canali di comunicazione, di non utilizzare una porta minore di 1024 per motivi di sicurezza (richiedono privilegi elevati, non utili ma dannosi), utilizzare un reverse proxy per mascherare il server sulla rete. In ogni caso, il sistema SeismoCloud ha tra gli sviluppi in corso un ammodernamento dell'interfaccia Web privata per un gruppo e tutta la parte di accesso relativo. La parte di autenticazione ora viene gestita tramite una password o un codice QR che può essere scansionato tramite l'app SeismoCloud. L'accesso al sistema di End User Development è gestito da questa dashboard, perciò è assolutamente necessario uno studio sulla sicurezza effettiva dell'autenticazione per evitare usi scorretti dei dati che il servizio fornisce.

6. Ringraziamenti

Sono tante le persone che ho incontrato in questo fantastico percorso, a partire dai professori fino ai miei colleghi di corso, collaboratori, amici. Credo che ogni persona ed interazione che io ho avuto con queste mi abbia in qualche modo aiutato o per lo meno cambiato qualche mia caratteristica o pensiero.
Ringrazio in primis la mia famiglia che mi ha sempre dato una mano e che ho trovato al mio fianco nei momenti più difficili. Ringrazio vivamente il Prof. Emanuele Panizzi che mi ha seguito constantemente in questo percorso facendomi crescere sia dal punto di vista tecnico, sia personale.
Non posso non ringraziare Enrico per tutte le volte che mi ha aiutato quando avevo difficoltà nel proseguire con il lavoro.
Ho avuto il piacere di condividere questi anni stupendi con dei colleghi meravigliosi, è anche grazie a loro se ora sto scrivendo queste frasi.

Questo è un grandissimo traguardo per me, ma piuttosto che un punto di arrivo, lo vedo come un ottimo punto di partenza verso altri obiettivi.
Spero vivamente che ogni persona possa avere la fortuna che ho avuto io nel vivere questi anni bellissimi pieni di esperienze formative condividendoli con persone fantastiche.

Estote semper Parati

7. Bibliografia

[1] **The inner structure of the Earth**.
http://sci.fgt.bme.hu/ volgyesi/gravity/ppfold.pdf
1982, L.Volgeyesi, M. Moser

[2] **Earthquake Magnitude, Energy Release, and Shaking Intensity**.
https://www.usgs.gov/natural-hazards/earthquake-hazards/science/earthquake-magnitude-energy-release-and-shaking-intensity

[3] **Istituto Nazionale di Geologia e Vulcanologia**. http://ingv.it

[4] **Ottimizzazione delle risorse nell'uso di servizi in background in SeismoCloud per Android** https://github.com/Enrico204/bachelor-degree-thesis
2017, Enrico Bassetti.

[5] **Seismo Berkeley Lab FAQ**. http://seismo.berkeley.edu/outreach/faq.html

[6] **REST API documentation**. https://restfulapi.net/

[7] **Wikipedia — End User Development**.
https://en.wikipedia.org/wiki/End-user_development

[8] **Publish MQTT messages and subscribe to message topics**.
https://www.mathworks.com/help/supportpkg/raspberrypi/ref/publish-and-subscribe-to-mqtt-messages.html

[9] **Simplify Node-RED for End User Development in SeismoCloud**
2020, EMPATHY Workshop; Enrico Bassetti, Edoardo Ottavianelli, Emanuele Panizzi.

[10] **Node-RED Wiki**. https://github.com/node-red/node-red/wiki

[11] **Wikipedia — Docker**. https://it.wikipedia.org/wiki/Docker

[12] **What does Docker technology add to just plain lxc**.
https://docs.docker.com/engine/faq/#what-does-docker-technology-add-to-just-plain-lxc

[13] **About storage drivers — Docker Documentation**.
https://docs.docker.com/storage/storagedriver/

[14] **How is Docker different from a Virtual Machine**
https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine

[15] **Performance Evaluation of Docker Container and Virtual Machine**
https://www.sciencedirect.com/science/article/pii/S1877050920311315,
2019, Amit M Potdar, Narayan D G, Shivaraj Kengond, Mohammed Moin Mulla.

[16] **Best practices for writing Dockerfiles — Docker Documentation**
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[17] **Human Computer Interaction Book — Evaluation techniques**
https://www.hcibook.com/e3-docs/slides/notes-pdf/e3-chap-09-6up.pdf

[18] **Node-RED configuration — Node-RED Documentation**
https://nodered.org/docs/user-guide/runtime/configuration

[19] **MQTT.js Documentation**
https://github.com/mqttjs/MQTT.js/wiki

[20] **Securing Node-RED — Node-RED Documentation**
https://nodered.org/docs/user-guide/runtime/securing-node-red

[21] **Function node — Node-RED GitHub**
https://github.com/node-red/node-red/blob/master/packages/node_modules/@node-red/nodes/core/function/10-function.js
2020, [Online, acceduto il 16/09/2020]

[22] **Securing IoT Apps in Node-RED — University of Gothenburg**
https://odr.chalmers.se/bitstream/20.500.12380/300759/1/CSE%2020-06%20Olsson%20ODR.pdf
2020, Lars Eric Olsson.

[23] **Node-RED Library — Telegram node**
https://flows.nodered.org/node/node-red-contrib-telegrambot.
2020, [Online, acceduto il 16/09/2020]

[24] **Node-RED Library — Twitter node**
https://flows.nodered.org/node/node-red-node-twitter.
2020, [Online, acceduto il 16/09/2020]

[25] **Node-RED Library — Facebook node**
https://flows.nodered.org/node/node-red-contrib-facebook.
2020, [Online, acceduto il 16/09/2020]

[26] **Node-RED Library — Alexa Assistant**
https://flows.nodered.org/node/node-red-contrib-alexa-smart-home.
2020, [Online, acceduto il 16/09/2020]

[27] **Node-RED Library — Google Home Assistant**
https://flows.nodered.org/node/node-red-contrib-googlehome.
2020, [Online, acceduto il 16/09/2020]

[28] **How to safely expose Node-RED to the Internet**
https://github.com/node-red/cookbook.nodered.org/wiki/How-to-safely-expose-Node-RED-to-the-Internet.