UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

"Sviluppo e test di una Web Application per Servizi di Localizzazione Indoor basata su Beacon Bluetooth"

"Development and test of a Web Application for Indoor Localization Services based on Bluetooth Beacon"

Relatore: Prof.ssa

Paola Pierleoni

Tesi di Laurea di:

Fabrizio Varelli

Sommario

1. Introduzione

La presente tesi ha lo scopo di implementare un sistema di Indoor Positioning (IPS) al livello cloud.

L'IPS è una rete di dispositivi che vengono utilizzati per localizzare persone o oggetti in luoghi chiusi nelle situazioni in cui il GPS e altre tecnologie satellitari vengono meno. La tecnologia scelta per lo sviluppo del progetto è il Bluetooth Low Energy (BLE), nato con il l'avvento del BT 4.0. Il Bluetooth di ultima generazione, non mantenendo una connessione costante con i dispositivi, invia i dati quando necessario, rendendosi perfetto per letture periodiche, con il vantaggio di consumare meno energia. Questa tecnologia è implementata nei Beacon Bluetooth, ossia piccoli dispositivi che incorporano microcontrollore, batteria e sensori/attuatori. Creando una rete di questi devices, attraverso il loro segnale di potenza ricevuto (RSSI) e la tecnica del Fingerprinting, è possibile implementare un sistema completo che fornisca il servizio della localizzazione indoor. La tecnica a RSSI si basa sulla misurazione della potenza del segnale trasmesso dai Beacon e ricevuto da un dispositivo. L'algoritmo di localizzazione associato è quello della trilaterazione, utile per trovare la posizione di un eventuale dispositivo nell'area di copertura dei Beacon. Lo svantaggio è la poca precisione ed è per questo che affianchiamo a questa tecnica il Fingerprinting, ovvero la registrazione, in una fase offline, dei valori di RSSI e di posizione di ogni Beacon in un database (DB). In questo modo, nella fase operativa di ricerca del dispositivo e quindi nella fase di lettura dei nuovi valori di RSSI, verrà effettuato un confronto con quelli registrate nel DB migliorando la precisione fino a 60 cm.

Tutto questo meccanismo verrà portato in cloud. Infatti, mentre tempo fa era necessario installare applicazioni e servizi direttamente su un sistema operativo rendendo sconveniente per un'azienda occuparsi dei costi di gestione di hardware e

di personale specializzato, con i container tutto il procedimento viene semplificato e reso versatile grazie alla loro capacità nel gestire lo spazio fisico e le risorse software autonomamente. Essi garantiscono anche una elevata stabilità, data dalla separazione delle applicazioni dal sistema operativo, evitando qualsiasi conflitto software. Inoltre, è possibile inserire all'interno di ogni container le risorse software utili al programmatore in modo che non debba perdere tempo nella ricerca e nella creazione di un environment adatto.

I software principalmente utilizzati in questo lavoro di tesi sono NEO4J, NGIN-X e Docker. Proprio quest'ultimo è la chiave del funzionamento del progetto, poiché si occupa della creazione e gestione dei container. NEO4J e NGIN-X sono i programmi che devono essere caricati all'interno di due containers separati, che verranno poi fatti interfacciare tra di loro.

Il primo container gestirà NEO4J, ovvero un gestore di database a grafo che permette una rapida e facile visualizzazione dei dati e delle loro relazioni. Questo lo rende visivamente rapido da scorrere e gestiremo con esso tutti i valori di RSSI e di posizione dei circa 80 Beacon posizionati lungo il piano universitario. In questo progetto viene utilizzato un DB contenente come nodi i Beacon e le posizioni assunte dal dispositivo ricevente durante la fase offline della tecnica del Fingerprinting. Questi nodi sono messi in relazione attraverso i valori di RSSI. Il sistema al momento è stato installato presso il Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche.

Il secondo container, invece, dirigerà NGIN-X, ovvero il programma che amministrerà il file HTML contenente l'algoritmo di localizzazione. Essendo un web server ad alte prestazioni ci permetterà di creare un sito sempre affidabile e raggiungibile ovunque ci troviamo.

In definitiva il file HTML plotterà la posizione di ogni Beacon e di ogni dispositivo presente nell'area indicando in questo modo il punto preciso in cui trovarlo e il modo migliore per raggiungerlo.

Lo scopo di questa tesi è quella di analizzare i dati ricevuti, catalogarli in un database e operare con questi dati da remoto in modo che non sia necessaria una macchina locale che svolga questo lavoro.

L'obiettivo finale di questo progetto è quello di replicare l'intero meccanismo all'interno dell'azienda **Fater S.p.A.** in modo che macchine e materiali siano sempre individuabili e facilmente raggiungibili dai dipendenti, senza perdita di tempo, riducendo così tempistiche e i costi.

Questa tesi è inserita nel progetto europeo:

Progetto dal titolo "SADAB-IT - Smart Awareness in Digital Automation and Business Intelligence with Integrated Tools" presentato a valere sul bando PON MISE "Fabbrica Intelligente, Agrifood e Scienze della vita"Asse I, Azione 1.1.3 PON Imprese e competitività 2014-2020. Partecipanti: Università Politecnica delle Marche; FATER S.p.A., Via Alessandro Volta 10, Pescara; Selda s.r.l., Via Porta Torricella 7, Ascoli Piceno.

1.1 Indoor Positioning

L'Indoor Positioning System è una rete di dispositivi utilizzati per localizzare persone o oggetti nelle situazioni in cui il GPS e altre tecnologie satellitari vengono meno, come all'interno di edifici a più piani, aeroporti, parcheggi sotterranei.

Sono molte le tecniche utilizzate e altrettanti i dispositivi sfruttati per fornire una stima della posizione in locali interni come smartphone, antenne WiFi e Bluetooth, smartwatch e smartband. Per la tecnologia IPS vengono sfruttati anche sensori luminosi, onde radio, campi magnetici e segnali acustici. Grazie a questi sensori, l'IPS può raggiungere una precisione nella stima della posizione di circa 2 cm, molto simile ai moderni GPS usati all'aperto.

Gli IPS utilizzano diverse tecniche, tra cui la misurazione della distanza tra i nodi[1], il posizionamento magnetico e la navigazione stimata. Tramite questi si possono individuare i dispositivi mobili fornendone la posizione ambientale. Purtroppo, la varietà di tecnologie usate per l'IPS ha portato alla frammentazione del design, con sistemi che utilizzano varie tecnologie ottiche, radio o persino acustiche rendendo spesso difficile far interfacciare nuovi dispositivi alla rete creata.

L'IPS viene sfruttato ampiamente nei settori: commerciale, militare per il monitoraggio dell'inventario e culturale per informazioni relative alle opere d'arte. Esistono diversi sistemi già in commercio, ma non esiste uno standard per la creazione di un sistema di indoor positioning, cosa che porta ad avere molta diversità nell'installazione dei sistemi. Da un lato si ha il vantaggio di adattare il sistema alle dimensioni spaziali, alle esigenze di precisione e ai vincoli di budget di chi lo richiede, dall'altro si ha il problema della compatibilità con i dispositivi utilizzati.

[1] Per nodi si intendono i dispositivi utilizzati nella rete IPS che sfruttano la connessione WiFi o Bluetooth per interfacciarsi tra loro, ad esempio punti di accesso WiFi, Beacon Bluetooth o BLE.

1.2 Tecniche Bluetooth per la localizzazione

Per la localizzazione indoor, le tecnologie WiFi e Bluetooth sono le più utilizzate e sfruttano spesso le stesse tipologie di algoritmi. Infatti, i nodi Bluetooth e/o WiFi vengono sfruttati come access point, in questo modo quando gli utenti vi si collegano con il loro smartphone è possibile stimare la loro posizione attraverso alcuni dei seguenti algoritmi appositamente creati.

1.2.1 Time of Arrival (ToA)

È possibile calcolare la distanza tramite la tecnica ToA poiché i segnali viaggiano con una velocità nota. A questo punto, sapendo il tempo di arrivo a due stazioni base si restringerà il campo a quello di intersezione dei due cerchi; aggiungendo i dati di una terza stazione base sarà possibile trovare la posizione di un singolo punto.

1.2.2 Time Difference of Arrival (TDoA)

Viene utilizzato quando non c'è sincronia tra i trasmettitori e ricevitori, sfruttando la differenza dei tempi si ottiene un risultato costante che sarà poi utilizzato similmente alla tecnica ToA.

1.2.3 Angle of Arrival (AoA)

Con l'avvento delle interfacce Wi-Fi MIMO[2], che utilizzano più antenne, è possibile stimare l'AoA dei segnali multipath ricevuti negli array di antenne nei punti di accesso e applicare la triangolazione per calcolare la posizione dei dispositivi client. Il calcolo tipico dell'AoA viene eseguito supponendo un array di antenne di *M*

[2] Acronimo di Multiple in Multiple out, la tecnologia MiMo sfrutta il multipath. Questa proprietà permette di aumentare la velocità di trasmissione senza che sia necessario aumentare la larghezza della banda: il segnale sarà inviato da diverse antenne di trasmissione e, grazie al rimbalzo dell'onda su muri, mobili e altri oggetti presenti nell'ambiente, raggiungerà l'antenna ricevente seguendo percorsi multipli in tempi leggermente diversi, creando, così diversi flussi dati simultanei in grado di trasportare più informazioni.

antenne equidistanti da una distanza di *d* e un segnale che arriva all'array di antenne attraverso *L* percorsi di propagazione. Una distanza aggiuntiva di $d \sin\theta$ viene percorsa dal segnale per raggiungere la seconda antenna dell'array. Attraverso vari calcoli è così possibile trovare gli angoli di arrivo alle diverse antenne per poi applicare la trilaterazione[3].

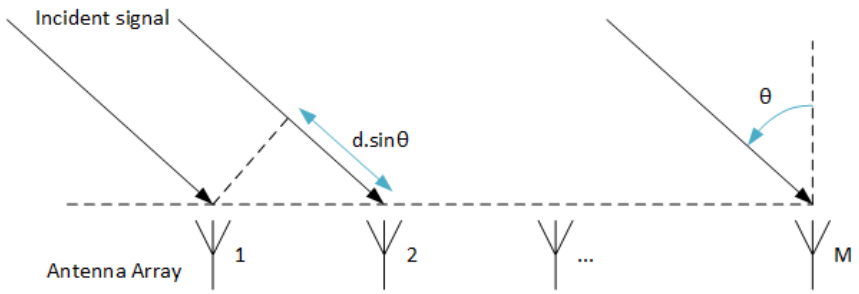


FIGURA 1: PATTERN ANTENNE MIMO

1.2.4 Received Signal Strength-Based Positioning Technology (RSSI)

Le tecniche di localizzazione RSSI si basano sulla misurazione della potenza del segnale da un dispositivo client a diversi punti di accesso e quindi sulla combinazione di queste informazioni con un modello di propagazione, per determinare la distanza tra il dispositivo client e i punti di accesso. Le tecniche di trilaterazione possono essere utilizzate per calcolare la posizione stimata del dispositivo client rispetto alla posizione nota dei punti di accesso.

Sebbene sia uno dei metodi più economici e più facili da implementare, il suo svantaggio è che non fornisce una precisione molto buona[4], perché le misurazioni RSSI tendono a fluttuare in base ai cambiamenti nell'ambiente o al fading

[3] La trilaterazione, come fa sottintendere il nome, sfrutta solo tre dispositivi client ma molto spesso se ne usano molti di più per questo si parla di multilaterazione.
[4] In letteratura studi dimostrano una media di precisione della localizzazione di 2-4 metri

multipath[5]. Per migliorare la tecnologia si è così pensato alla tecnica del Fingerprinting. Questo metodo si basa semplicemente sulla rilevazione della potenza del segnale da diversi punti di accesso presenti nell'area e sulla memorizzazione di queste informazioni in un database insieme alle coordinate note del dispositivo client in una fase offline. Durante la fase di tracciamento online, il vettore RSSI attuale di un dispositivo situato in una posizione sconosciuta viene confrontato con quelli memorizzati nel database e viene restituita la corrispondenza più vicina come posizione stimata dell'utente. Tali sistemi possono fornire una precisione media tra 0.6m e i 1.3m.

Il suo principale svantaggio è che qualsiasi modifica dell'ambiente come l'aggiunta o la rimozione di mobili o edifici può modificare il fingerprinting corrispondente a ciascuna posizione, richiedendo un aggiornamento del database. Tuttavia, l'integrazione di sensori sonori o fotocellule può risolvere il problema mappando l'area e restituendo con vari algoritmi i nuovi valori di RSSI.

[5] È una forma di distorsione di un segnale che giunge a destinazione sotto forma di un certo numero di repliche, sfasate nel tempo, originate dai vari percorsi (multipath) che il segnale stesso può aver seguito durante la sua propagazione e sommantesi tra loro in ricezione.

1.3 Bluetooth Low Energy e Beacon

Da sempre si è utilizzata la tecnologia Bluetooth (BT) per la connessione wireless di dispositivi per lo streaming continuo di dati o audio. Inventato formalmente nel 1999, si è evoluto velocemente grazie al suo basso costo e la sua facilità di impiego.

La tecnologia BT sincronizza le operazioni con un segnale di clock in tempo reale. Questo serve a sincronizzare gli scambi di pacchetti tra i dispositivi. Il clock del Bluetooth è realizzato con un contatore a 28 bit che viene posto a 0 all'accensione del dispositivo e subito dopo continua senza mai fermarsi, incrementandosi ogni 312,5 µs.

Ogni dispositivo, connettendosi ad una rete Bluetooth, viene identificato tramite un codice di 24 bit dagli altri dispositivi. I collegamenti che possono essere stabiliti tra i dispositivi sono di due tipi:

- Connectionless: è un servizio asincrono dove il trasmettitore può in qualunque momento iniziare ad inviare i propri pacchetti purché si conosca l'indirizzo di destinazione, senza che venga richiesta alcuna connessione prima di inviare i pacchetti
- Connection Oriented: un servizio sincrono dove viene richiesta una connessione tra i dispositivi prima di inviare i dati.

Una rete di soli due dispositivi connessi tra loro tramite BT viene chiamata piconet e sono due i ruoli che questi dispositivi possono ricoprire:

- Master: ovvero il dispositivo che si occupa di tutto ciò che concerne la sincronizzazione del clock deli altri dispositivi slave.
- Slave: ovvero le unità sincronizzate al clock del master ed al canale di frequenza.

Quindi possiamo individuare due stati in cui un dispositivo Bluetooth si può trovare:

- Quello di connessione se è connesso a un altro dispositivo ed è coinvolto con esso nelle normali attività;
- Quello di standby se il dispositivo non è connesso o non è coinvolto nelle attività della piconet. Questo stato è concepito come un modo per far risparmiare energia ai dispositivi, in quanto se uno di essi non è coinvolto attivamente all'interno di una connessione, non c'è motivo che assorba picchi di potenza pari a quelli dei dispositivi attivi. Quando un'unità si trova in standby ascolta il canale ogni 1,28 secondi per rilevare eventuali messaggi provenienti dal master.

Sono moltissime le versioni che si sono susseguite nel tempo. Con le versioni 1.1 e 1.2 dello standard Bluetooth vengono gestiti trasferimenti di dati con velocità fino a 723,1 kbit/s. Nella versione 2.0 è stata aggiunta una modalità con velocità più elevata, che consente trasferimenti fino a 3 Mbit/s. Tuttavia, questa modalità comporta un incremento notevole dei consumi di corrente nei dispositivi, che prevalentemente sono alimentati a batteria. Con la versione 4.0 dello standard la velocità di trasferimento è stata incrementata fino a 4 Mbit/s. Allo stesso tempo è stato introdotto un nuovo criterio, la riduzione della durata dei segnali trasmessi, che a parità di quantità di dati trasferiti, riesce a dimezzare la corrente richiesta rispetto al Bluetooth versione 1.2. Questo criterio dà vita al Bluetooth Low Energy che, come dice il nome, ha come obiettivo principale quello di aggregare dati provenienti da diversi sensori, come sensori di umidità, sensori per monitorare la frequenza cardiaca, termometri, ecc., tramite un'ottimizzazione della struttura di trama[6] e l'impiego di dispositivi più efficienti energeticamente, ma a discapito della

[6] Un insieme di bit generato al livello 2 del modello ISO/OSI e che costituisce l'entità minima che viene trasmessa su una particolare rete. Ogni pacchetto che viaggia sulla rete viene suddiviso in diverse trame, ciascuna delle quali riporta un insieme di bit che ne compongono l'intestazione (header) ed un insieme di bit che ne compongono la coda. Questi servono a distinguere una trama dall'altra e a verificarne il contenuto una volta che la trama giunge a destinazione.

velocità, che in questa modalità si attesta a 1 Mbit/s. In parole povere il Bluetooth Low Energy (BLE) è una nuova generazione di BT che consuma molto meno rispetto a quello tradizionale. Questo accade perché invece di mantenere una connessione costante con i dispositivi, il protocollo BLE invia i dati quando necessario. Questo significa che è perfetto per letture periodiche provenienti da sensori ma non può essere utilizzato per streaming audio e video. Per esempio, potremmo sfruttarlo per letture di sensori di temperatura o pressione di una piccola stazione metereologica oppure per ricevere informazioni di un'opera d'arte se il dispositivo BLE è situato nelle vicinanze.

Il campo di applicazioni che ci interessa di questa tecnologia è quella relativa ai Beacon Bluetooth. Questi sono piccoli trasmettitori che sfruttano la tecnologia del BLE, per trasmettere il loro ID ai dispositivi elettronici nelle vicinanze. La tecnologia consente a smartphone e altri dispositivi di eseguire azioni in prossimità di un Beacon. Questo perché oltre all'ID vengono inviati diversi byte che possono essere utilizzati per determinare la posizione fisica del dispositivo o attivare un'azione basata sulla posizione del dispositivo come un check-in sui social media o una notifica. Per esempio, un'applicazione potrebbe distribuire messaggi in un punto di interesse specifico, ad esempio un negozio, una fermata dell'autobus, una stanza o un luogo più specifico come un distributore automatico.

Il campo di utilizzo dei Beacon BT che mi interessa approfondire è quello relativo agli IPS, che aiuta gli smartphone a determinare la loro posizione in un determinato luogo. Infatti, grazie all'aiuto di un Beacon BT, un utente con uno smartphone può trovare approssimativamente la sua posizione relativa ad un Beacon BT nelle vicinanze all'interno di un supermercato.

Come si è capito dai vari esempi, i Beacon BT sono un dispositivo di trasmissione unidirezionale e richiede una applicazione specifica installata sul proprio dispositivo per far sì che si possano ricevere le informazioni. Ciò garantisce che solo l'app

installata possa tracciare gli utenti mentre camminano passivamente intorno ai trasmettitori, garantendone quindi la privacy.

Con più Beacon per stanza, è possibile stimare la posizione di un utente con una precisione di circa due metri, tramite la trilaterazione. I Beacon BT sono infatti in grado di trasmettere il valore RSSI oltre ad altri dati. Utilizzando la nota potenza del segnale di uscita del Beacon e la potenza del segnale osservata dal dispositivo ricevente, è possibile fare un'approssimazione della distanza tra il Beacon e il dispositivo.

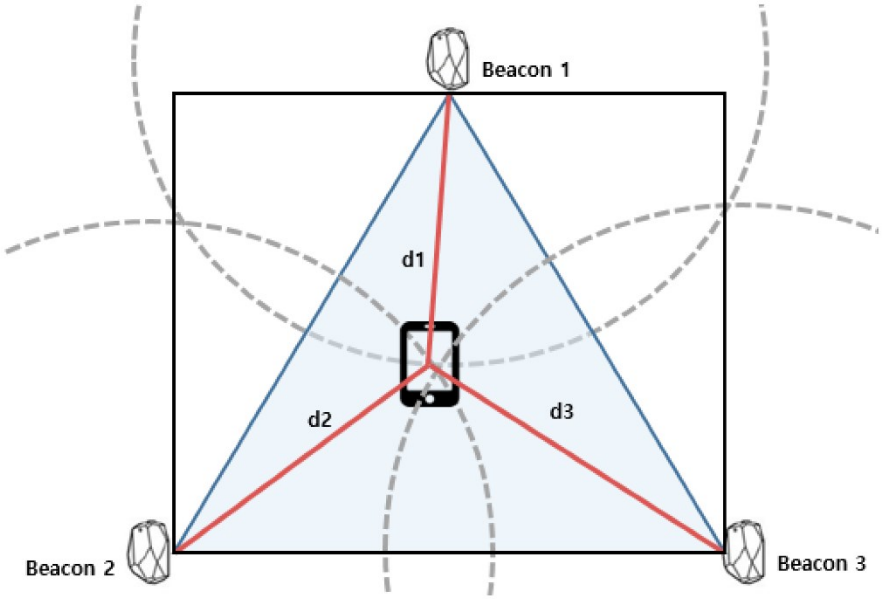


FIGURA 2: ESEMPIO DI TRILATERAZIONE

1.4 Vantaggi applicazioni cloud

Prima dell'avvento del cloud computing tutte le applicazioni e i servizi che potevano essere utili per un'azienda dovevano essere caricati in un server locale dell'azienda che potesse reggere per potenza e capacità tutti i processi da svolgere. Questo richiedeva degli enormi costi aggiuntivi che vanno dal semplice upgrade di hardware per l'implementazione di questi servizi, al costo delle varie figure professionali che dovevano recarsi costantemente in azienda per risolvere ogni tipo di problema.

Oggi invece con i servizi di cloud computing è possibile eliminare una buona parte di queste problematiche, risparmiando sui costi grazie alla loro velocità di utilizzo, ai loro tempi di operatività e alla facilità di utilizzo. Infatti, basterà collegarsi ad un browser, accedere al servizio, personalizzarlo ed usarlo.

Potremmo riassumere i vantaggi principali del cloud computing nei seguenti punti:

1. Assenza di costi relativi all'hardware

Il primo vantaggio del cloud computing è quello di consentire alle aziende di accedere alle infrastrutture hardware in modo nuovo, prescindendo dal concetto di possesso e di investimento, in quanto non ci sono più i costi relativi all'hardware.

2. Flessibilità

Dato il punto uno, il cloud computing risulta più flessibile proprio grazie ad una infrastruttura cloud. La potenza della macchina varierà in dipendenza delle necessità, senza che l'utente se ne accorga e permettendo in questo modo prestazioni migliori.

3. Collaborazione

Non essendo più limitati dall'HW, è possibile usare le applicazioni cloud ovunque e in qualsiasi momento, a prescindere dal dispositivo utilizzato. Inoltre, il team di

lavoro può vedere in tempo reale i progressi dei singoli membri, modificando i file in ogni momento e ovunque ci si trovi.

4. Protezione e sicurezza dei dati

Come ultimo punto vale la pena sottolineare che prima, per riparare danni e occuparsi della sicurezza, vi era un enorme difficoltà data sia dai costi, sia dagli addetti specializzati da assumere. Adesso invece non sono più necessari team enormi per la cyber security cosa che ha permesso di alleggerire ulteriormente le spese dell'azienda. Infatti, molto spesso i servizi cloud già hanno integrati servizi di questo tipo.

Inoltre, se la perdita di hardware aziendale può costituire un danno economico non indifferente, la perdita di dati o interruzione di servizi può essere un danno ancor peggiore. Il vantaggio del cloud è quello di dare sicurezza in casi del genere, perché è possibile avere accesso ai dati indipendentemente da quello che succede ai pc.

1.5 Possibilità dell'utilizzo dei containers

Tempo addietro era necessario installare le applicazioni direttamente sul sistema operativo, questo aveva il grosso svantaggio di avere l'applicazione e i suoi file eseguibili solo su quel particolare OS. Una possibile soluzione poteva essere quella di creare una virtual machine (VM). Ma questo portava con sé il grande svantaggio di appesantire la macchina su cui si lavorava e non c'era alcuna possibilità di poterla cambiare per provare il tutto altrove.

Con i containers si sviluppano delle applicazioni che sono virtualizzate a livello di sistema operativo invece che a livello hardware. In questo modo, diversi containers possono essere eseguiti direttamente sul kernel del sistema operativo, rendendoli molto più leggeri, veloci e quindi sfruttano meno risorse e memoria rispetto a quanto richiederebbe l'avvio di un intero sistema operativo.

In particolare, i containers isolano le applicazioni l'una dall'altra e, a meno che non le si connetta esplicitamente, non bisognerà preoccuparsi di conflitti di codice ad esempio.

Così tutto l'environment crea un ulteriore livello di sicurezza, dato che le applicazioni non vengono eseguite direttamente sul sistema operativo su cui si lavora.

Altro punto a favore è quello di poter includere nei containers le librerie software necessarie all'applicazione, facilitando lo sviluppatore nel creare un proprio ambiente di lavoro. Questo porta, a lungo termine, ad avere un incremento di produttività ed un abbassamento degli errori e bug commessi dallo sviluppatore stesso.

I containers sono in grado di funzionare ovunque: sia su sistemi operativi Linux, Windows, Mac, su macchine virtuali o fisiche, sulla macchina di uno sviluppatore o nel cloud pubblico.

Il classico software legato ai containers è Kubernetes perché rende più semplice il deployment e la gestione delle applicazioni. Kubernetes esegue automaticamente implementazioni e rollback[7], monitorando lo stato dei servizi per evitare errori di implementazione prima che sia troppo tardi. Esegue inoltre continuamente controlli di integrità dei servizi, riavviando i containers che non si sono avviati o che sono bloccati e annunciando ai client solo i servizi che si sono avviati correttamente. Kubernetes scala automaticamente i servizi in base all'utilizzo, consentendo di eseguire solo ciò di cui si ha bisogno. Come i containers, Kubernetes permette di gestire in maniera dichiarativa il cluster, consentendo una configurazione controllata a livello di versione e facilmente replicabile.

[7] Il rollback in informatica è un'operazione che permette di riportare la base di dati a una versione o stato precedente.

2. Software e Tecnologie Utilizzate

2.1 Kubernetes

Kubernetes è una piattaforma open–source eseguibile in infrastrutture on–premises[8], ibride o cloud ideata per automatizzare la messa in campo, lo scaling e la gestione di applicazioni in containers, raggruppandoli in unità logiche di più alto livello, che ne facilitano la gestione e la manutenzione. Kubernetes è un progetto scritto in Go, inizialmente sviluppato da Google, attualmente parte della Cloud Native Computing Function, costruito sulla base dei principi che permettono a Google di eseguire e gestire miliardi di containers ogni settimana, garantendo un'ottima scalabilità al crescere delle dimensioni del proprio team.

Kubernetes sfrutta un contenitore per distribuire ed eseguire in remoto le applicazioni, in modo da evitare interruzioni di servizi e di renderle accessibili a molti. È possibile montare un sistema di archiviazione composto da storage locale o da dischi forniti da cloud pubblico. In questo modo Kubernetes gestisce lo spazio di archiviazione come meglio crede per evitare i problemi e per mantenere sempre un servizio stabile. Se ci sono molte persone collegate può anche gestire quanto traffico dedicare agli utenti. Cercherà anche di gestire al meglio le risorse, in modo da evitare sovraccarichi di CPU e RAM. In pratica Kubernetes è un software in grado di utilizzare algoritmi per la gestione dello spazio e delle risorse per eseguire al meglio le applicazioni in modo da poterlo distribuire alla massa senza avere problemi di servizi. Nel nostro caso questo ci aiuta a gestire la moltitudine di richieste degli

[8] Per software on-premises si intende l'installazione ed esecuzione di software direttamente su macchina locale. Si contrappone al software on-cloud.

operai che cercheranno di sapere con precisione la locazione di vari macchinari e di materiale all'interno dell'azienda.

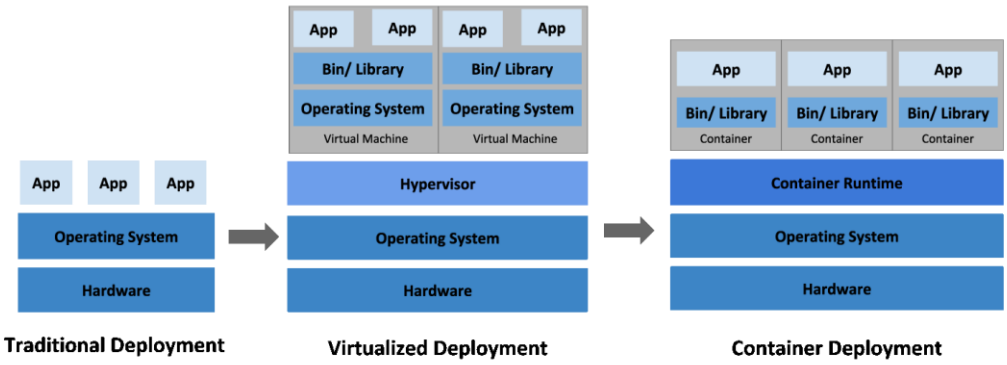


FIGURA 3: STRUTTURA DI UN CONTAINER

2.1.1 Approfondimento

Kubernetes è un progetto open-source di Google formato da più software secondo il pattern orchestrator. Esso è usato per la gestione di cluster[9] di medie e grandi dimensioni e di applicazioni complesse.

Kubernetes permette l'eliminazione di processi manuali sia per lo sviluppo sia per l'esecuzione delle applicazioni e consente di gestire cluster di host con container Linux in modo scalabile, semplificando i processi al fine di gestire al meglio i carichi di lavoro.

Le componenti che si occupano di controllare l'esecuzione di container applicativi sono raggruppate nel *control plan*. Il *data plan* raggruppa le componenti software coinvolte nelle funzionalità e gestiscono il carico di lavoro del cluster. Il controllo del sistema avviene specificando uno stato desiderato (*desired state*).

[9] È un insieme di pc connessi tra loro tramite una rete telematica. Lo scopo del cluster è distribuire un'elaborazione molto complessa tra vari computer aumentando la potenza di calcolo a costo di un prezzo maggiore.

Per utilizzare un cluster è necessario conoscerne la struttura. I termini *pod, service, replication controller e deployment* si riferiscono al meccanismo che distribuisce i containers su un cluster Kubernetes. I termini *ETCD, server API, scheduler, kubectl deamon e kube proxy* vengono, invece, utilizzati per configurare un cluster.

- Un *Pod* è l'unità di esecuzione più piccola di Kubernetes. Se un pod (o il nodo su cui viene eseguito) fallisce, Kubernetes può creare automaticamente una nuova replica di quel pod per continuare le operazioni. I pods possono includere uno o più contenitori (come i contenitori di Docker) e rappresentano i processi in esecuzione su un cluster. Limitando i pods ad un singolo processo, Kubernetes può generare report sull'integrità di ogni processo in esecuzione nel cluster.
- Un *Service* Kubernetes è un'astrazione logica per un gruppo distribuito di pods in un cluster (che svolgono tutti la stessa funzione). Poiché i pods sono effimeri, un servizio consente di assegnare un nome e un indirizzo IP univoco (clusterIP) a un gruppo di pods che forniscono funzioni specifiche (servizi Web, elaborazione delle immagini, ecc.). Finché il servizio esegue quell'indirizzo IP, esso non cambierà.
- Il *Deployment* viene utilizzato per indicare a Kubernetes come creare o modificare le istanze dei pods che contengono un'applicazione containerizzata. Le distribuzioni possono ridimensionare il numero di pods di replica, abilitare l'implementazione del codice aggiornato in modo controllato o ripristinare una versione di distribuzione precedente, se necessario.
- *Replication Controller* è responsabile della gestione del ciclo di vita del pod. È responsabile di assicurarsi che il numero specificato di repliche di pod sia in esecuzione in qualsiasi momento. Viene utilizzato nel momento in cui si vuole assicurarsi che il numero specificato di pods o almeno un pod sia in

esecuzione. Ha anche la capacità di alzare o abbassare il numero di pods specificato.

- *Kube proxy* è essenzialmente un proxy di rete che consente agli agenti di comunicare sia con la rete principale sia con quella esterna. KUBECTL è responsabile della comunicazione con il master e si assicura che i contenitori all'interno di quel nodo funzionino correttamente.
- ETCD è l'archivio contenente coppie di valore-chiave del container. Il master deve tenere traccia di tutti gli agenti e se, per esempio, il servizio 1 esegue cinque containers e il servizio 2 ne esegue due, tiene traccia di tali informazioni relative a quel container.

È possibile trovare la guida di installazione di Kubernetes per Windows nell'appendice A.

2.2 Helm

Per rendere il tutto più semplice è possibile utilizzare Helm. Questo è un gestore di pacchetti di applicazioni in esecuzione per Kubernetes. Permette di descrivere la struttura dell'applicazione attraverso comode Helm chart[10] e di gestirla con semplici comandi. Helm è importante perché cambia il modo in cui le applicazioni lato server vengono definite archiviate e gestite poiché l'utilizzo di questo gestore di pacchetti semplifica notevolmente la loro gestione.

Nel mio caso, questi template ci permettono di impostare vari settings riguardanti NEO4J in maniera rapida, andando semplicemente a modificare i parametri del file nelle varie parti che ci interessano. Il tutto sempre da command line. Con il comando seguente sarà possibile impostare i parametri del file :

```
Helm template graphdb –set acceptLicenseAgreement=yes –set
neo4jPassword=Password , > /tmp/neo4j.yaml
```

A questo punto dobbiamo solo applicarli con il comando:

```
Kubectl apply -f /tmp/neo4j.yaml
```

[10] file con estensione .yaml

Per eseguire invece il database a grafo di NEO4J dobbiamo dare il comando:

```
Kubectl exec -t -i nomedeldatabase:neo4j /bin/bash
```

Si aprirà una linea di comando con root iniziale e, se vogliamo scrivere in Cypher, possiamo indicargli cypher-shell.

2.3 KubeCTL

Lo strumento che usiamo spesso da riga di comando per richiamare Kubernetes è KubeCTL[11] e ci consente di eseguire comandi su un cluster Kubernetes. Si può utilizzare KubeCTL per distribuire applicazioni, ispezionare e gestire risorse del cluster e visualizzare i log.

KubeCTL usa la seguente sintassi per eseguire i comandi dalla cmd:

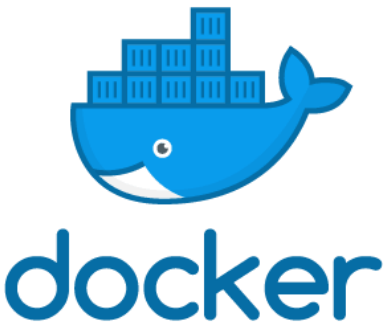
```
kubectl [command] [TYPE] [NAME] [flags]
```

dove command, TYPE, NAME, e flags sono:

- command: specifica l'operazione che si desidera eseguire su una o più risorse, ad esempio create, get, describe, delete.
- TYPE: specifica il tipo di risorsa . I tipi di risorsa non fanno distinzione tra maiuscole e minuscole ed è possibile specificare le forme singolari, plurali o abbreviate.
- NAME: specifica il nome della risorsa. I nomi fanno distinzione tra maiuscole e minuscole. Se il nome viene omesso, vengono visualizzati i dettagli per tutte le risorse, ad esempio kubectl get pods.
- flags: specifica i flag facoltativi. Ad esempio, puoi utilizzare i flag -s oppure --server per specificare l'indirizzo e la porta del server API Kubernetes.

[11] Deriva da kube(rnetes) control

2.4 Docker



Docker nasce all'inizio del 2013 come un progetto open source scritto in Go di dotCloud, un'azienda incentrata sulle Platform–as–a–Service in cloud, come un'estensione della tecnologia sviluppata per eseguire e monitorare il proprio cluster di server. L'azienda nei mesi successivi cambia il proprio nome in Docker Inc., in seguito all'unione con la Linux Foundation e nel gennaio 2020 conta più di 2500 star su GitHub ed oltre 3500 fork.

È una piattaforma aperta per lo sviluppo e l'esecuzione di applicazioni. È molto simile a Kubernetes, ma la più grande differenza sta nel fatto che Docker viene eseguito su un singolo nodo mentre Kubernetes è eseguito su un cluster. In pratica mentre Docker lo usiamo su un nostro computer personale, Kubernetes lo eseguiamo su un server.

Altra differenza è che Docker può essere usato senza Kubernetes, mentre Kubernetes necessita di un Runtime del container per fare il suo lavoro. Quindi, nella pratica, mentre Docker fornisce un contenitore su cui eseguire il tutto, Kubernetes viene eseguito dentro al container per sfruttare i suoi algoritmi di gestione e per orchestrare i vari containers su una scala più ampia, gestendo anche l'interazione dei vari container con l'utente. Possiamo vedere le loro differenze in questo modo:

- *Docker senza Kubernetes*

  Da solo ci permette di creare dei contenitori, gestirli in un registro ed eseguirli in modo che tutti comunichino tra loro tramite Docker compose. Quando si utilizza qualcosa in larga scala Docker non basta più.

- *Kubernetes senza Docker*

  Kubernetes non include funzionalità per la creazione o la gestione dei contenitori e non esegue i contenitori. Deve funzionare con un'origine del

contenitore esterno, cioè qualcuno che crei i contenitori e una Runtime. In pratica ciò che Kubernetes fornisce è un framework ricco, flessibile e potente per la definizione delle applicazioni e l'orchestrazione dei contenitori su larga scala.

2.4.1 Approfondimento

È un progetto che automatizza il deployment di applicazioni all'interno di contenitori software, formando un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo Linux. Docker utilizza la funzionalità di isolamento di Linux, evitando l'installazione e la manutenzione di una macchina virtuale.

Implementa API ad alto livello per gestire containers che seguono processi in ambienti isolati. Visto che utilizza delle funzionalità del kernel di Linux, un container di Docker, a differenza di una macchina virtuale, non include un sistema operativo separato ma utilizza la funzionalità del kernel e sfrutta l'isolamento delle risorse (come CPU, MEMORIA e RAM) e i namespace[12] separati, per isolare ciò che l'applicazione può vedere del sistema operativo.

Utilizzando i containers le risorse possono essere isolate, i servizi limitati e i processi avviati in modo da avere una prospettiva completamente privata del sistema operativo. È possibile creare più containers autonomi che utilizzino risorse in modo differente.

È possibile trovare la guida di installazione di Docker per Windows nell'appendice B.

[12] Una collezione di nomi di entità definite dal programmatore usate in egual modo in uno o più file sorgente. Lo scopo è quello di evitare confusione ed equivoci nel caso siano necessarie molte entità con nomi simili, raggruppandoli in categorie.

2.5 Minikube

Minikube è uno strumento che ci consente di eseguire Kubernetes localmente, creando un cluster a nodo singolo sul nostro computer in modo da poter provare Kubernetes senza doverlo mettere su un cluster pubblico.

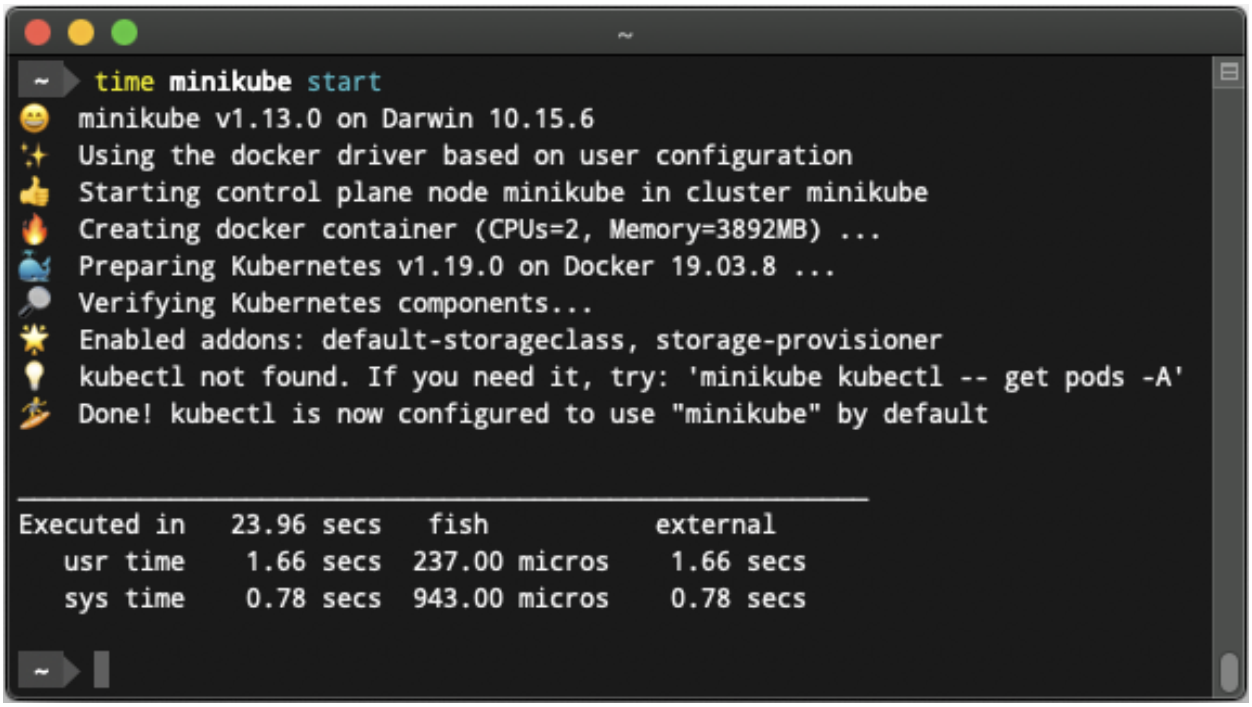


FIGURA 4: SCHERMATA DI PRESENTAZIONE MINIKUBE

2.6 NEO4J

È un gestore di database a grafo, ovvero un database progettato per trattare le relazioni tra i dati come informazioni altrettanto importanti nella gestione dei dati stessi. Ha lo scopo di contenere i dati senza costringerli a un modello predefinito. I dati vengono quindi gestiti tra nodi (il dato in sé) che contengono varie proprietà. Essi possono essere contrassegnati con varie etichette che rappresentano i loro ruoli. Le relazioni invece sono connessioni dirette tra i nodi. Per esempio, guardando la figura 5, possiamo connettere due nodi del tipo *nome di un dipendente* al *nome dell'azienda* collegandolo con la relazione lavora come. Anche le relazioni possono avere delle proprietà, solitamente vengono usate proprietà come peso, costi, distanze, valutazioni, intervalli di tempo e punti di forza. Nel nostro esempio una proprietà di questa relazione è da quando lavora nell'azienda.

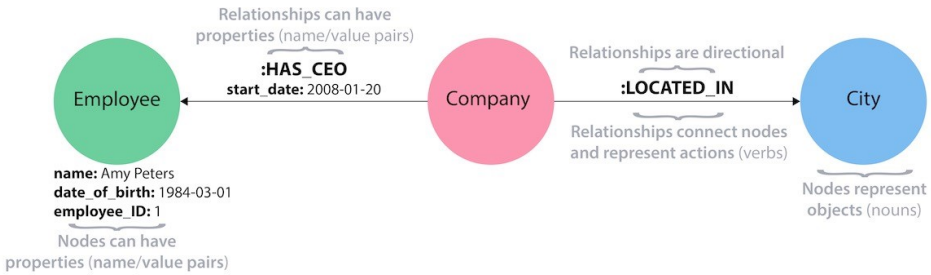


FIGURA 5: ESEMPIO DI NODO E REAZIONE

NEO4J a questo punto gestisce tutte queste informazioni memorizzando i dati in modo efficiente e sfruttando i puntatori per navigare e attraversare il grafico.

Nel nostro progetto con il comando CREATE abbiamo già prodotto vari nodi chiamati Beacon con un ID unico che hanno come proprietà la latitudine e la longitudine, che

indicando la loro posizione nello spazio. I diversi nodi sono legati dalla proprietà RSSI, ovvero la potenza del segnale che c'è tra i vari Beacon BLE.

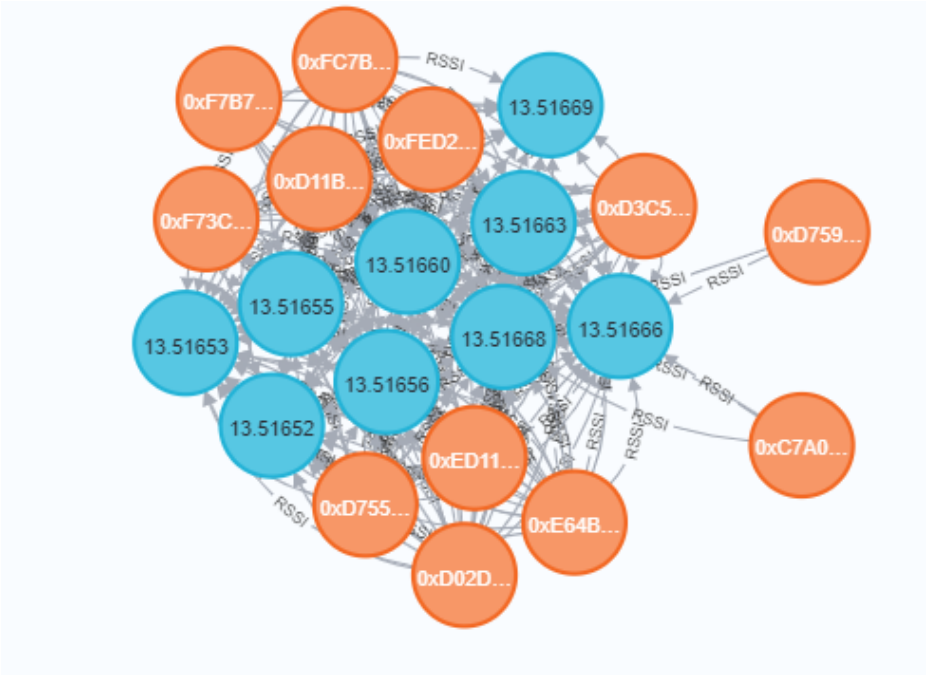


FIGURA 6: QUESTO È IL NOSTRO DATABASE. IN ARANCIONE I NODI BEACON, IN BLU IL NODO POSIZIONE LEGATI DALLA PROPRIETÀ DI RSSI.

2.6.1 Approfondimento

NEO4J è un software per database a grafo open source sviluppato in java. Esso può essere usato in due modalità: SERVER ed EMBEDDED. Nel primo caso si accede tramite "REST", un'architettura per sistemi distribuiti, tramite query da remoto. Nel secondo caso un dispositivo viene utilizzato per archiviare NEO4J all'interno di una java virtual machine con file jar.

Entrambi i metodi supportano la modalità batch per l'importazione massiva di dati da altri database o da file, ma per uso comune si basa sulle transitions. Infatti, una volta aperta una transition è possibile creare nodi e assegnarvi delle proprietà, ossia

valori corrispondenti ai tipi di dato elementari di java e identificati grazie ad un nome. È possibile unire i nodi tramite le relazioni i cui limiti sono definiti dal programmatore.

NEO4J integra un sistema di indicizzazione che permette di memorizzare a piacere dei nodi facendo riferimento ad una etichetta assegnata liberamente.

Per muoversi tra i nodi si usano vari linguaggi di programmazione come GREMLIN e CYPHER[13].

La struttura a grafo di NEO4J si mostra estremamente comoda ed efficiente nel trattare strutture come alberi estratti da file XML, filesystem e reti. L'esplorazione di queste strutture risulta in genere più veloce rispetto ad un database a tabelle perché la ricerca dei nodi in relazione con altri è un'operazione primitiva e non richiede molti passaggi. Ogni nodo contiene l'indice delle relazioni entranti o uscenti, quindi la velocità di scorrimento del grafo non risente delle dimensioni complessive ma solo della densità dei nodi.

Il più grande svantaggio è quello di essere legato ai linguaggi di programmazione sopra indicati, infatti, se sfruttassimo query SQL il sistema si appesantirebbe. Altro svantaggio è che NEO4J non può memorizzare dati binari come audio, video o grossi blocchi di testo.

È possibile trovare la guida di installazione di NEO4J per windows nell'appendice C.

[13] Esempio di linguaggio Cypher per creazione di nodi, relazioni e proprietà:

```
MATCH p= (company:Company)-[DEVELOPERS]->(game:Game)<-[PUBLISHES]
WHERE company.uid = 'company-electronics_arts'
Return p
```

2.7 Orchestrazione dei containers con NEO4J tramite Kubernetes e Docker

Quando si parla di un gran numero di containers si incontrano sfide legate ai sistemi distribuiti con la scalabilità (*scaling*), replica (*replication*), tolleranza agli errori (*fault tolerance*) e comunicazione tra containers (*container comunication*).

A questo scopo si attua il processo di orchestrazione (*orchestration*): l'idea è quella di passare dall'avvio di un solo container su una macchina, all'avvio di molti container su una flotta di macchine che li gestisca.

Per fare ciò utilizziamo le API di Docker in un cluster. Quando si ha un solo container che riproduce un solo host, si ha bisogno di una sola interfaccia cmd di Docker, mentre se si hanno molti containers che aumentano di numero bisogna avere delle Docker API che comunichino con gli altri containers. Aumentando il numero di containers aumenterà anche la difficoltà dell'operazione. Quello di cui si ha bisogno, quindi, è di una singola ed unificata interfaccia che possa comunicare con tutte le macchine Docker. Per fare ciò abbiamo bisogno di un manager costituito da uno scheduler ed un servizio discovery (ricerca). Lo scheduler si occupa di decidere quale applicazione lanciare in base al container oppure di quanta memoria disporre. Il servizio di discovery, invece, permette di identificare l'entrata o l'uscita di un nodo da un cluster. Tutto questo avviene tramite un meccanismo di token.

Ci sono tre tipi di scheduler in Docker:

- Spread strategy: Usato solo con pochi nodi attivi.
- Bin pack strategy: ottimizza i nodi con più pacchetti, così invece di dividere i container tra i nodi, cerca di riempire un nodo prima di passare al successivo, riducendo la frammentazione ma aumentando il rischio di perdita di informazioni.
- Random strategy: è l'accesso randomico ai containers.

2.8 NGINX

NGINX[14] è un web server leggero ad alte prestazioni. Può funzionare su molti sistemi operativi come Unix, Linux, macOS e Windows.

NGINX fornisce rapidamente i contenuti statici con un utilizzo efficiente delle risorse di sistema. È possibile distribuire contenuti dinamici HTTP su una rete che utilizza i gestori FastCGI[15] per gli script, e può servire come bilanciatore di carico.

NGINX utilizza un approccio asincrono basato su eventi nella gestione delle richieste, in modo da ottenere prestazioni più prevedibili sotto stress, in contrasto con il modello del server HTTP Apache che usa un approccio orientato ai thread o ai processi nella gestione delle richieste.

Sono molti i campi di applicazione di questo tool, raccolti nel loro sito ufficiale. Eccone alcune che ci permettono di capire il suo campo di utilizzo con Docker:

- Server virtuali basati su nome e IP;
- Supporto per connessioni keep-alive[16] e pipeline[17];
- Formati di log di accesso, controllo degli accessi in base all'indirizzo IP del client, tramite password (autenticazione HTTP Basic) e dall'esito della sotto-richiesta;
- Scrittura di log in buffer, rotazione veloce dei log e logging syslog[18].

Useremo NGINX per montare il file HTML dove è stata sviluppata l'interfaccia utente dalla quale visualizzare le posizioni dei Beacon e le posizioni degli oggetti da trovare. Monteremo NGINX in un container gestito da Docker in modo che si interfacci con il container di NEO4J.

[14] Pronunciato come engine-x.
[15] FastCGI è un protocollo che permette di interfacciare programmi interattivi CGI con un server web. Lo scopo principale di FastCGI è quello di ottimizzare le risorse del sistema nell'interfacciamento tra il programma CGI e il server web, permettendo al server di gestire più richieste di pagina web assieme.
[16] Keep-alive è un metodo che consente una conversazione server-client HTTP sulla stessa connessione TCP, invece che aprirne una per ogni nuova richiesta.
[17] L'HTTP pipelining è una tecnica in cui vengono inviate più richieste HTTP su una singola connessione TCP senza aspettare le risposte corrispondenti.
[18] SYSLOG (System Log) è un protocollo di rete appartenente alla suite di protocolli Internet utilizzato per trasmettere attraverso una rete semplici informazioni di log.

3. Setup dei containers

3.1 Container per NEO4J

La prima sfida che ho dovuto affrontare è stata quella della creazione di un container che avviasse un determinato programma.

Prima di tutto ho abilitato Kubernetes in modo da sfruttare le sue funzionalità di gestione dello spazio; è possibile abilitarlo nelle impostazioni stesse di Docker. Questo ci fa ben capire, come sottolineato nei capitoli precedenti, come Docker e Kubernetes siano due facce della stessa medaglia. Fatto questo il comando che ho usato per la creazione del database è il seguente:

```
docker run -p7474:7474 -p7687:7687 -e NEO4J_AUTH=neo4j/1234 neo4j
```

Il comando include la definizione della porta di accesso da remoto *(-p7474:7474*) e la porta Bolt per l'accesso al database da un altro container; contiene l'autenticazione (*NEO4J_AUTH=neo4j/1234*) e la versione di NEO4J da installare (*neo4j*). Non essendo specificata verrà installata direttamente la versione più recente, identificata con latest.

Il nome assegnato al database è casuale, nel mio caso *jovial_carver*. Ma può essere benissimo scelto in partenza con l'aggiunta del flag *--name nome_container*.

Una volta dato il comando, il container si avvierà automaticamente dopo aver installato NEO4J al suo interno.

A questo punto, trasporto i file necessari al funzionamento all'interno del container tramite il comando:

```
docker cp c:\path\to\local\file container_name:/path/to/target/dir/
```

In pratica seleziono la cartella di partenza del file da copiare e indicando il nome del container seleziono la cartella di destinazione. Quindi sempre da cmd do il comando:

```
docker cp C:\Users\varef\Desktop\beacon.dump  jovial_carver:/var/lib/neo4j/import
```

A questo punto ci sono un paio di osservazioni da fare: prima di tutto la cartella import non è una cartella qualsiasi. Essa è una delle poche che sono state montate dal sistema, non possiamo infatti scegliere una cartella a caso ma dobbiamo obbligatoriamente scegliere una delle cartelle elencate nella schermata di Docker.

Inoltre, dato che il comando non restituirà errore anche se non è andato a buon fine, è necessario controllare che ci sia tutto nella cartella */import*. Per farlo, apro la linea di comando da dentro Docker premendo il pulsante CLI come in figura 7.

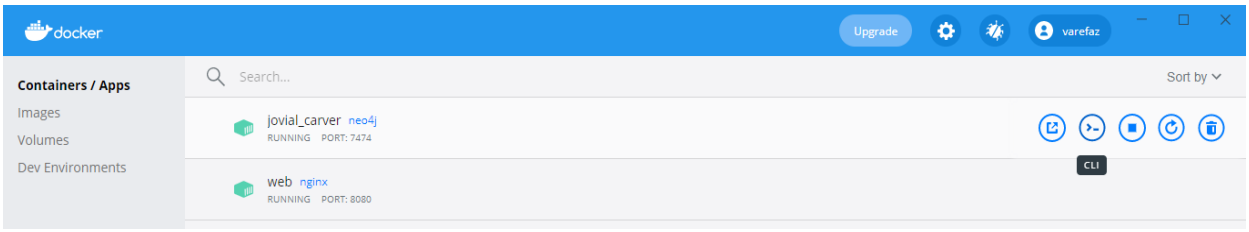


FIGURA 7: I CONTAINER E L'ACCESSO AL CLI

La schermata che si aprirà è una linea di comando in cui è possibile scrivere direttamente i comandi di Docker[19], oppure posso aprire la linea di comando bash per cercare tra i file del container. Per farlo do il comando:

```
/bin/bash
```

[19] Infatti, finora abbiamo dato i comandi da command line di Windows, quindi ogni comando era preceduto dalla dicitura Docker nome_comando contenuto_comando. Questo perché dovevamo richiamare il programma in esecuzione.

A questo punto sulla sinistra trovo il nome del percorso in cui siamo, come è possibile vedere in figura 8.

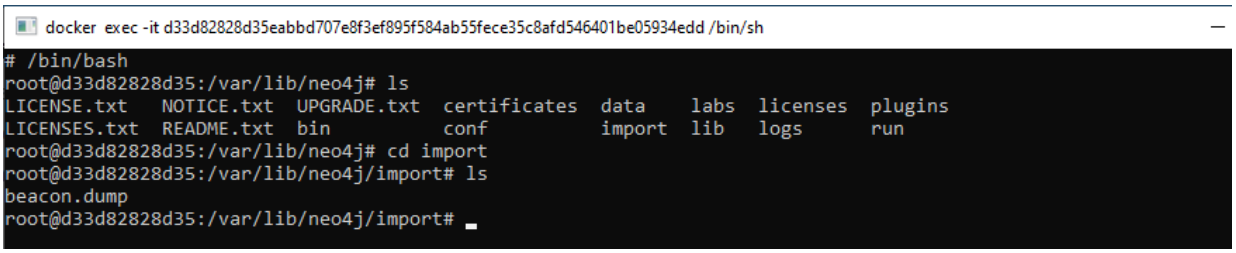


FIGURA 8: RICERCA DEI FILE NECESSARI

Se tutto è andato a buon fine, mi aspetto di trovare, muovendomi tra le cartelle con i dovuti comandi[20], il file nella cartella import. Quindi posso procedere con il caricamento del database tramite il comando:

```
docker exec jovial_carver neo4j-admin load --database=beacon --from=/var/lib/neo4j/import/beacon.dump
```

Esso sfrutta il richiamo alla funzione exec che, al contrario di run[21], esegue il comando in un container già esistente. Inserisco il nome del container in cui lo voglio eseguire e, a questo punto, do il comando necessario a NEO4J per eseguire il caricamento del dump inserendo da quale cartella deve prendere il file dump. Se tutto va come previsto il dump verrà caricato come in figura 9.

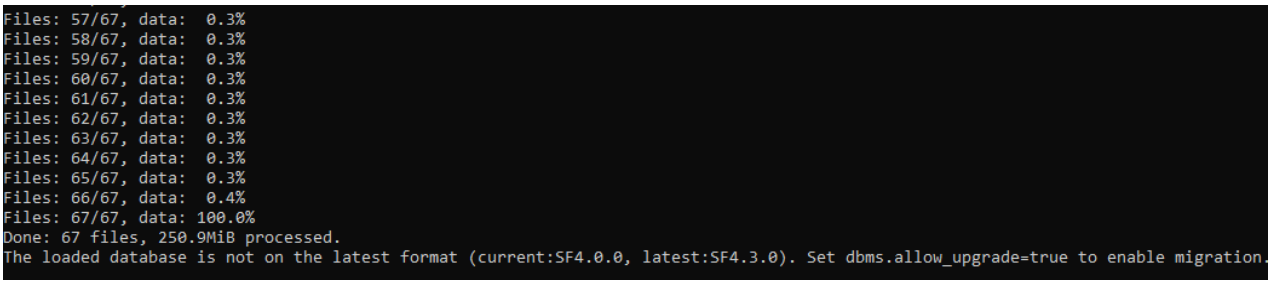


FIGURA 9: CARICAMENTO DEL DATABASE NEL CONTAINER

[20] I comandi utilizzati sono: cd (change directory) sfruttato per spostarsi in una cartella con nome noto, ls (list) ci dà una lista di tutte le cartelle e file presenti nella cartella in cui siamo. Questo mi permette di sapere sempre il nome della cartella in cui voglio arrivare.

[21] Docker run ha una duplice funzione: esegue un comando creando però un nuovo container.

Ora, per verificare se tutto è avvenuto correttamente, apro nuovamente la cmd tramite il pulsante CLI, do il comando per aprire la linea di comando bash e cerco la cartella /data/databases. Dentro questa trovo un elenco di cartelle, tra cui è presente anche quello che ho appena caricato.

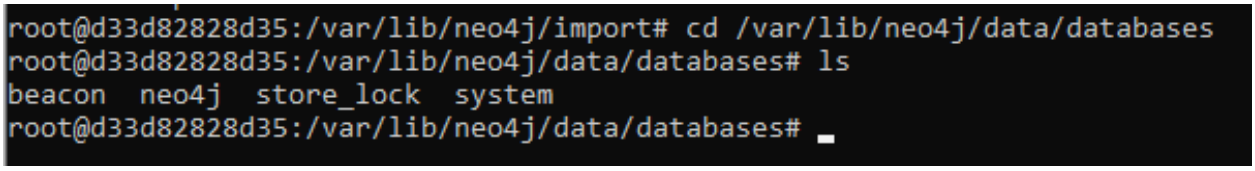


FIGURA 10: RICERCA DEL DATABASE TRAMITE CARTELLE

In questo modo il database e il container sono subito pronti all'uso.

3.2 Container per NGINX

Come per il container NEO4J adesso creo quello per NGINX con il comando:

```
docker run -p 8080:80 --name web nginx
```

Verrà creato un nuovo container nominato web con porta di accesso 8080. Posso quindi raggiungere la pagina internet tramite indirizzo localhost:8080. Apparirà una pagina diversa rispetto a quella di NEO4J. Questa è una pagina di presentazione chiamata *index.html*. Il mio scopo è quello di sostituire questo file con uno modificato da me per far funzionare tutto l'algoritmo di localizzazione.

Prima di procedere modifico il codice del nostro file HTML nella sezione in cui richiamo il database, dandogli la porta bolt corretta (ovvero 7687), inserisco anche i dati di accesso utilizzati precedentemente per la creazione dell'altro container.

```
525 var markerLayers = new L.LayerGroup();
526 var timeLayer = new L.LayerGroup();
527 var driver = neo4j.driver("bolt://localhost:7687", neo4j.auth.basic("neo4j", "1234"));
528
```

FIGURA 11: ACCESSO AL DATABASE

Una volta salvato, procedo con il comando per copiare e incollare il file HTML che ho accuratamente rinominato *index.html*.

```
docker cp C:\Users\varef\Desktop\index.html  web:/usr/share/nginx/html
```

Anche qui la cartella non è casuale, ma è esattamente il percorso dove NGINX prende il file da avviare: */usr/share/nginx/html*

Quindi *Docker cp* è il comando per copiare i file da macchina locale a container, dandogli il percorso in cui si trova il file sul pc e infine il nome del container seguito dal percorso in cui incollare il file.

Al termine, aprendo il browser, all'indirizzo *localhost:8080*, si potrà visualizzare la veste grafica del file HTML, avendo così l'interfaccia grafica pronta all'uso.

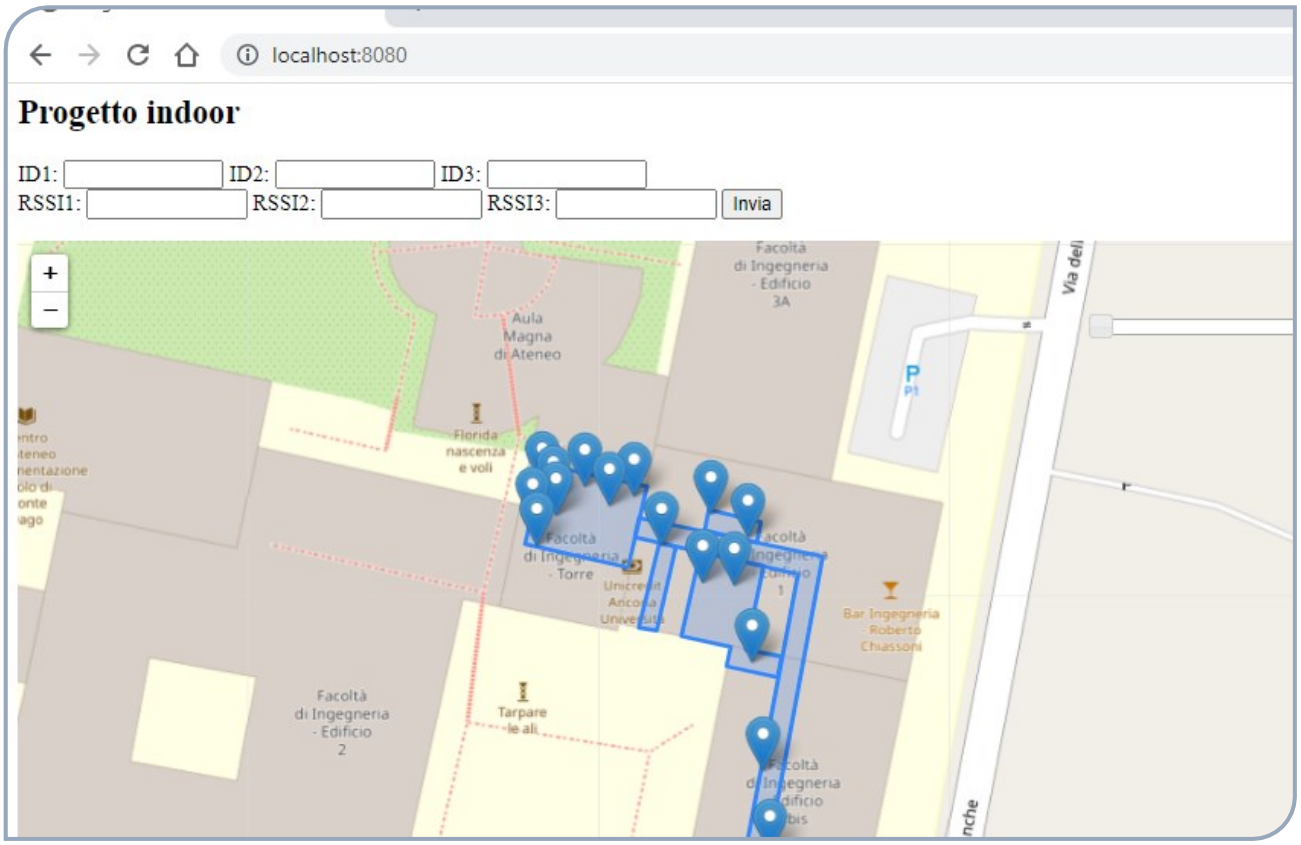


FIGURA 12: SITO CREATO DA NGINX

4. Programmazione del file HTML

La sfida più grande è stata quella di inserire all'interno del file HTML l'algoritmo di localizzazione, facendo apparire sulla mappa un marker rosso che indicasse la posizione del dispositivo da cercare.

Precedentemente programmato in java, bisognava capire come inserirlo sotto un altro linguaggio di programmazione.

```
var markerLayers = new L.LayerGroup();
var timeLayer = new L.LayerGroup();
var driver = neo4j.driver("bolt://localhost:7687", neo4j.auth.basic("neo4j", "1234"));

var session = driver.session();
      session
        .run('MATCH (b1:Beacon{lat:b1.latitude, lng:b1.longitude})-[r1:RSSI] -> (p1)'+
             'MATCH (b2:Beacon{lat:b2.latitude, lng:b2.longitude})-[r2:RSSI] -> (p2)'+
             'MATCH (b3:Beacon{lat:b3.latitude, lng:b3.longitude})-[r3:RSSI] -> (p3)'+
             'WHERE ID(p1) = ID(p2)' +
             'AND ID(p1) = ID(p3)' +
             'RETURN abs(r1.rssi-b1.rssi) as qwe, abs(r2.rssi-b2.rssi) as asd, abs(r3.rssi-b3.rssi) as zxc, ' +
             ' ID(p1) as pto1, ID(p2) as pto2, ID(p3) as pto3, ' +
             '(abs(r1.rssi-b1.rssi) + abs(r2.rssi-b2.rssi) + abs(r3.rssi-b3.rssi)) as tot, ' +
             'p1.lat, p1.lng ' +
             'order by tot, qwe, asd, zxc limit 1')
        .subscribe({
            onNext: function (record) {
            //console.log(record);
            var marker = new L.Marker(new L.LatLng(record.get('latitudine'), record.get('longitudine')), {icon: redIcon});
            marker.bindPopup('<b>numero:</b> ' + record.get('tot.id') + '<br>' + '<b>latitudine: </b>' + record.get('tot.latitudine').toString() + '<br><b>longitudine:</b> ' + record.
            markerLayers.addLayer(marker);
          },
            onCompleted: function livello() {
                        //costruzione dei piani
                        var Q165 = L.layerGroup([layerQ165, markerLayers]);

                      //array piani
                        var overlayMaps = {
                                          "Q165": Q165
                        };
                      L.control.layers(baseMaps, overlayMaps).addTo(map);


        session.close();
```

FIGURA 13: PARTE DELLA FUNZIONE SESSION

L'idea è stata quella di creare una funzione che eseguisse e contemporaneamente plottasse il marker sulla mappa. Infatti, la funzione session si occupa in primo luogo di eseguire tramite il comando *.run* l'algoritmo di localizzazione scritto in Cypher, in modo che potesse essere eseguito direttamente nel container di NEO4J.

In particolare, l'intero algoritmo si basa sulla somiglianza tra i dati. I parametri acquisiti durante la prima operazione offline della tecnica del fingerprinting, che sono salvati nel database, vengono confrontati con i nuovi valori acquisiti nell'operazione di rilevamento online. La somiglianza è valutata tramite la distanza euclidea tra i valori di RSSI. Di conseguenza la posizione del target è associata alla minima distanza calcolata.

Con i comandi preceduti da MATCH troviamo tutti i nodi posizione associati ai nodi con etichetta Beacon. Ognuno di questi ha delle proprietà che sono latitudine, longitudine e valore di RSSI. Poi, considerando i punti di training[22] in comune con ciascun Beacon, si calcola la distanza euclidea tra il valore attuale di RSSI e il rispettivo valore memorizzato. Questo viene reiterato per ogni Beacon rilevato. Organizziamo infine il risultato precedente in ordine crescente, in questo modo si associa la posizione del ricevitore alle coordinate geografiche del risultato più piccolo ottenuto.

```
run('MATCH (b1:Beacon{lat:b1.latitude, lng:b1.longitude})-[r1:RSSI] -> (p1)'+
    'MATCH (b2:Beacon{lat:b2.latitude, lng:b2.longitude})-[r2:RSSI] -> (p2)'+
    'MATCH (b3:Beacon{lat:b3.latitude, lng:b3.longitude})-[r3:RSSI] -> (p3)'+
    'WHERE ID(p1) = ID(p2)' +
    'AND ID(p1) = ID(p3)' +
    'RETURN abs(r1.rssi-b1.rssi) as qwe, abs(r2.rssi-b2.rssi) as asd, abs(r3.rssi-b3.rssi) as zxc, ' +
    ' ID(p1) as pto1, ID(p2) as pto2, ID(p3) as pto3, ' +
    '(abs(r1.rssi-b1.rssi) + abs(r2.rssi-b2.rssi) + abs(r3.rssi-b3.rssi)) as tot, ' +
    'p1.lat, p1.lng ' +
    'order by tot, qwe, asd, zxc limit 1')
```

FIGURA 14: QUERY NEO4J

In secondo luogo, con la funzione *.subscribe* viene plottato il risultato sulla mappa, prendendo le caratteristiche di latitudine e longitudine della variabile *tot* calcolate con l'algoritmo precedente.

```
.subscribe({
    onNext: function (record) {
    //console.log(record);
    var marker = new L.Marker(new L.LatLng(record.get('latitudine'), record.get('longitudine')), {icon: redIcon});
    marker.bindPopup('<b>numero:</b> ' + record.get('tot.id') + '<br>' + '<b>latitudine: </b>' + record.get('tot.latitudine').toString() + '<br><b>longitudine:</b> '
                                                                                                + record.get('tot.longitudine').toString());
    markerLayers.addLayer(marker);
```

FIGURA 15: CREAZIONE DEL MARKER

Infine, per testare tutto l'ambiente di lavoro, ho inserito le caselle di testo e il bottone "invia" che prenderanno in input tre valori plausibili di RSSI e tre valori corrispondenti all'ID fittizio di ogni Beacon per calcolare una ipotetica posizione e aggiungere sulla mappa un marker rosso in corrispondenza della latitudine e longitudine calcolati.

[22] Sono i punti che vengono registrati nella fase offline della tecnica del fingerprinting.

```
<form onSubmit = "return saveVar();">
    ID1: <input type="text" size="12" id="id1" />
    ID2: <input type="text" size="12" id="id2" />
    ID3: <input type="text" size="12" id="id3" /><br />
    RSSI1: <input type="text" size="12" id="rsi1" />
    RSSI2: <input type="text" size="12" id="rsi2" />
    RSSI3: <input type="text" size="12" id="rsi3" />
    <input type="submit" />
</form>
<script>
function saveVar() {
    var xID1 = document.getElementById('id1').value;
    var xID2 = document.getElementById('id2').value;
    var xID3 = document.getElementById('id3').value;
    var rssi1 = document.getElementById('rsi1').value;
    var rssi2 = document.getElementById('rsi2').value;
    var rssi3 = document.getElementById('rsi3').value;
    //alert(rssi1);
    //alert(xID1);
    return false;
}
</script>
```

FIGURA 16: FUNZIONE PER IL SALVATAGGIO DELLE VARIABILI

5. Conclusione e Sviluppi Futuri

In conclusione, tutto il sistema di Indoor Positioning è stato portato in cloud tramite i containers. Ci basterà assegnare degli indirizzi IP statici per poter raggiungere i containers da qualsiasi dispositivo.

Inoltre, avendo inserito l'algoritmo di localizzazione all'interno del file HTML, non è necessario programmare un eventuale terzo container che svolga solo questo compito, rendendo la tecnica più semplice e versatile. Chiunque si approccerà al progetto, potrà sfruttare la funzione già creata per testare l'environment, anche senza avere a disposizione dei Beacon BLE fisici, emulando i dati presi in ingresso tramite le caselle di input. Ovviamente sarà sempre possibile, una volta ultimato il progetto, eliminare le caselle di input e la funzione delle variabili, lasciando all'intero sistema il controllo dell'applicazione, rendendo il tutto autonomo e veloce. Sfruttando un'applicazione Android, la mappa dei Beacon potrà essere visualizzata da qualsiasi dispositivo e, di conseguenza, la posizione di un eventuale materiale o macchinario da ritrovare all'interno dell'edificio. Sarà possibile aggiungere in un futuro la possibilità di visualizzare i vari piani di un edificio, applicando così tutto il meccanismo a qualsiasi livello di un'azienda.

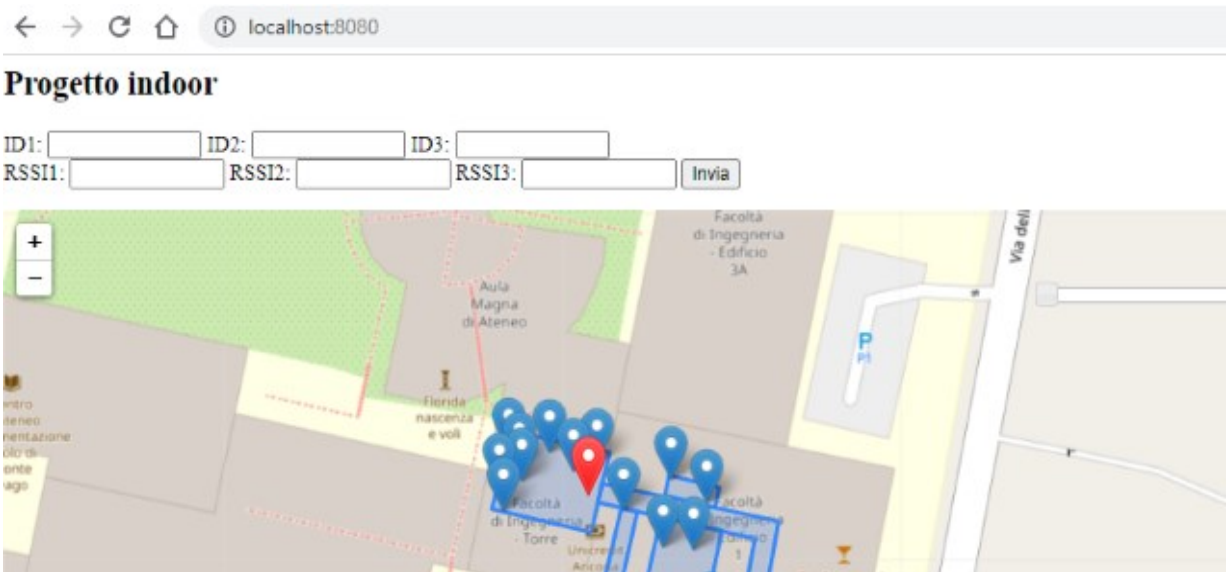


FIGURA 17: SVILUPPI FUTURI DEL SITO

6. Appendice

A. Installazione Kubernetes su Windows

Per scaricare Kubernetes si segue il link, accedendo alla schermata di download:

https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/#install-kubectl-binary-with-curl-on-windows

Tra le varie opzioni ho scelto la prima, che va sotto il nome di **Install kubectl binary with curl on Windows.**

Quindi scarico ed avvio l'ultima versione di kubectl:

https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe

Quando si avvia Kubectl come file exe non darà altro che questa schermata.

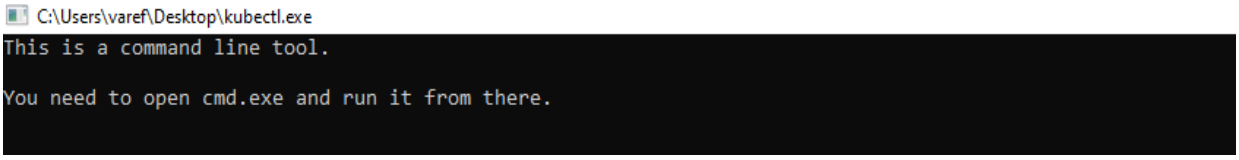


FIGURA 18: KUBECTL

Quindi andiamo su cmd, apriamo il prompt dei comandi, sempre con i privilegi di amministratore, e diamo il comando:

```
curl -LO https://dl.k8s.io/v1.21.0/bin/windows/amd64/kubectl.exe.sha256
```

Partirà il download del checksum. A questo punto con il comando seguente verificare che i due codici siamo gli stessi.

```
CertUtil -hashfile kubectl.exe SHA256
```

Possiamo ora mettere il file dove preferiamo e possiamo controllare la versione con il comando:

```
kubectl version –client
```

Verifichiamo il funzionamento con il comando:

```
kubectl cluster-info
```

Se apparisse un errore del genere allora sarebbe presente un problema nella configurazione.

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it
```

Per risolverlo bisogna installare minikube ma per averlo abbiamo bisogno di Windows package installer e anche winget. Quindi seguire i passi sottostanti.

Aprire il seguente link dello store windows:

https://www.microsoft.com/it-it/p/app-installer/9nblggh4nns1?ocid=9nblggh4nns1

Cliccare su ottieni e poi nuovamente ottieni fino alla installazione, se questo non avviene bisogna prima installare Windows 10 insider.

Quindi nel seguente link inserire la propria mail con cui si accederà a Windows.

https://forms.microsoft.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR-NSOqDz219PqoOqk5qxQEZUMVVCT1IwVEpLSklZS0dDRFZEUjZUOU9ZWi4u

Poi andare su *https://insider.windows.com/it-it/* e cliccare su aggiorna, sotto la voce CANALE SVILUPPATORI. Se avete disabilitato i dati di diagnostica da inviare a Windows riabilitateli seguendo quello che apparirà sullo schermo. Abilitate la spunta su Windows insider. Ora andando su Windows update e aggiornando più volte la lista degli aggiornamenti da fare uscirà un nuovo aggiornamento. A questo punto dopo il riavvio, winget sarà abilitato altrimenti tornare al punto precedente in cui vi ho fatto installare il software dallo store di Windows.

Adesso possiamo installare minikube dando da prompt dei comandi (cercate cmd dalla barra di ricerca su start) il comando:

```
winget install minikube
```

Finita l'installazione procediamo con l'avvio. Quindi diamo il comando:

```
minikube start
```

Aspettiamo la configurazione e diamo il comando per l'integrazione con Kubernetes:

```
kubectl get po -A
```

A questo punto possiamo aprire la dashboard con:

```
minikube dashboard
```

B. Installazione Docker su Windows

Installare Docker desktop dal sito ufficiale e tenerlo aperto durante l'utilizzo di Kubernetes altrimenti si avranno errori di connessione.

https://docs.docker.com/docker-for-windows/install/

È necessario anche un sottosistema wsl2. Se non è presente su Windows, tornare alla appendice A.

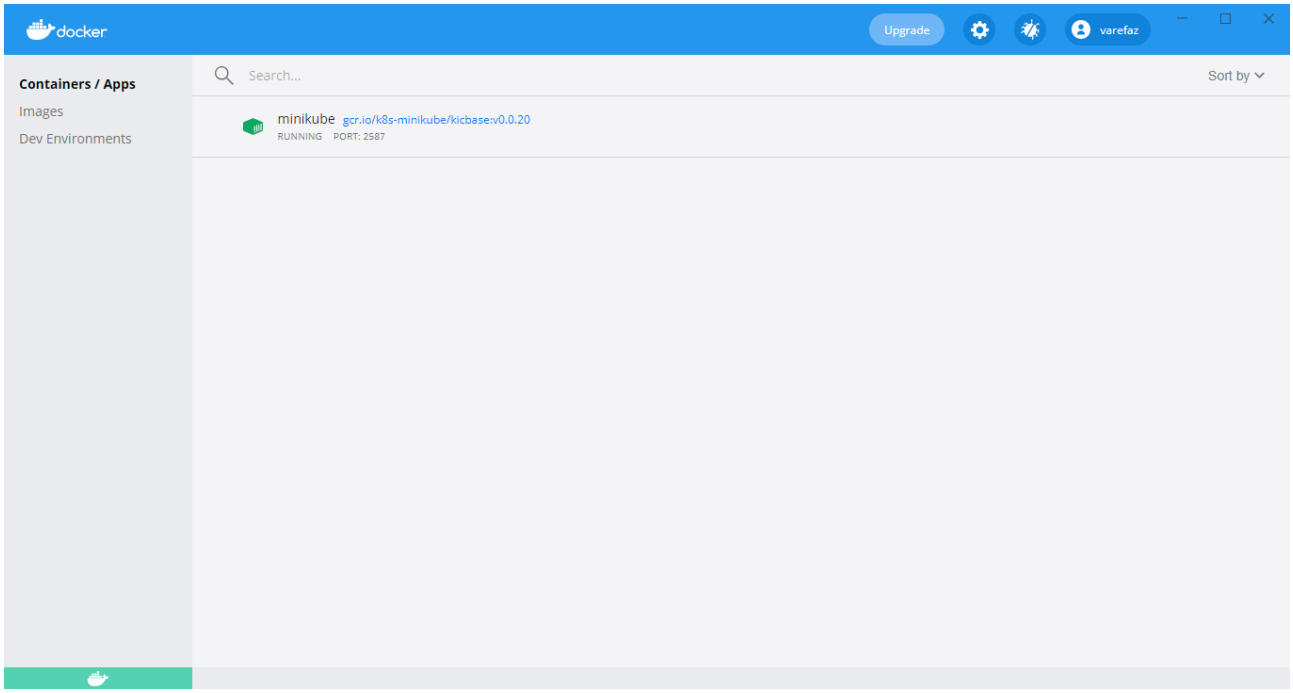


FIGURA 19: DOCKER DESKTOP

C. Installazione NEO4J su Windows

Tramite link, accedere alla pagina di download:

https://neo4j.com/download-neo4j-now/?utm_program=emea-prospecting&utm_source=google&utm_medium=cpc&utm_campaign=emea-search-branded&utm_adgroup=neo4j-desktop&gclid=Cj0KCQjw6-SDBhCMARIsAGbI7UgvrLj2ro1yjZ3dBWDoU3S_Vp9NvN0dMUJ-pZsOBkbZsPnPYxS_xFgaAmCKEALw_wcB

Sulla destra, sotto la scritta "**Download Your Free Copy of Neo4j**" inserire nome, cognome, e-mail e alla voce company inserire student, mentre per la voce country scegliere Italy. Cliccando sul tasto download avremo in pochi istanti il file .exe da installare.

Una volta aperto il file ci chiederà di rifare il procedimento di registrazione (oppure è possibile fare il login se si possiede una chiave di accesso). Quindi compiliamo il form allo stesso modo e clicchiamo registration o comunque il tasto verde in basso a destra. Una volta avviato avremo una schermata del genere:

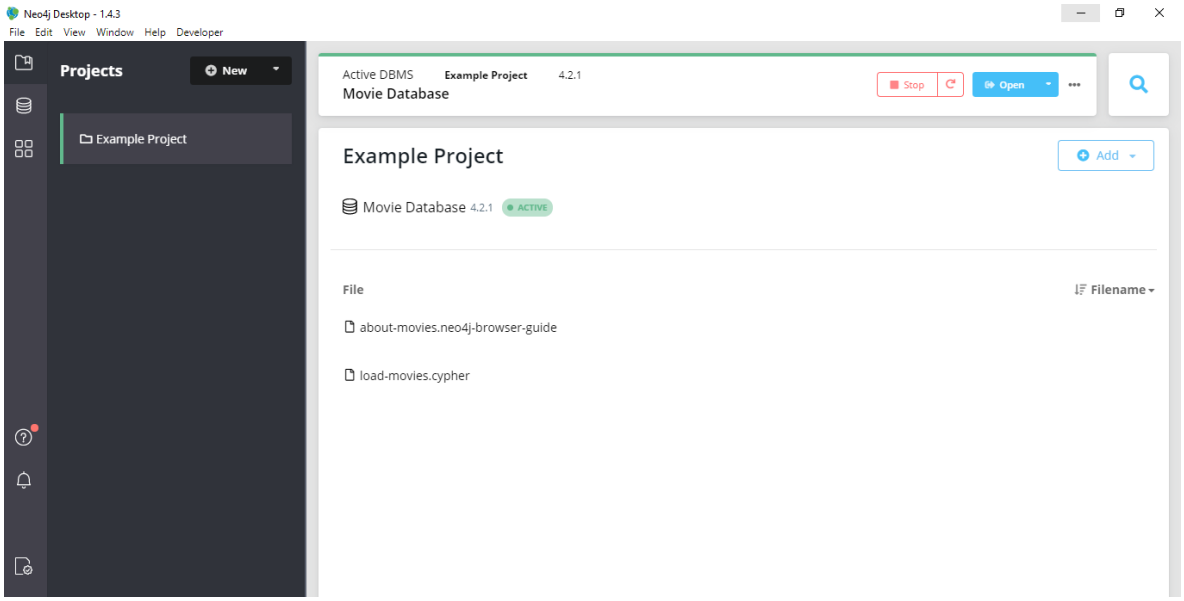


FIGURA 20: SCHERMATA DI NEO4J

Pochi secondi dopo, se presenti, verranno richiesti un paio di aggiornamenti. Dare l'ok e saremo operativi.

Per caricare un dump dobbiamo aprire il terminale senza far partire il DBMS (nel caso sia su ACTIVE premere stop). Quindi cliccare sui tre puntini vicino open (casella celeste), andare su terminal e dare il seguente comando:

```
bin\neo4j-admin load --from=dumps\nomedelfile.dump --database=nomedeldatabase --force
```

Il file che vogliamo caricare dovrà essere presente nel percorso di partenza che ci verrà mostrato sulla schermata del terminale. Per esempio:

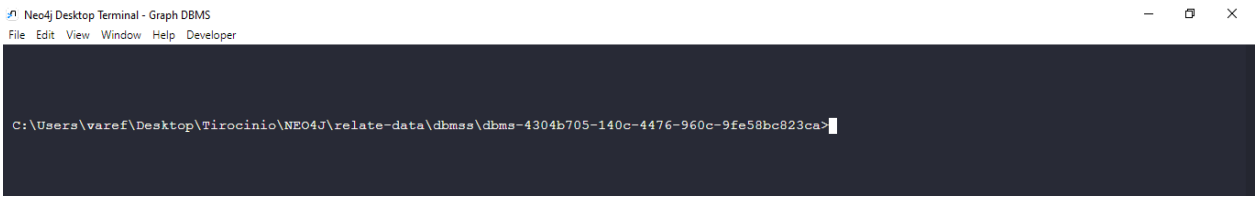


FIGURA 21: TERMINALE CON IL PERCORSO DEL DATABASE

Quindi troviamo la cartella *dbms-4304b705-…* seguendo a ritroso il percorso. All'interno di questa inseriamo una cartella dumps e all'interno inseriamo il file che vogliamo caricare con estensione dump. Fatto questo il file verrà caricato su NEO4J.

Ora andiamo nella schermata principale di NEO4J e diamo start al dbms. Una volta avviato, appena sotto di esso, avremo una lista dei nomi di database presenti di default system (in grigio non selezionabile) e neo4j. Se avremo dato nel comando il nome del database uguale a quello di default (neo4j), il database verrà caricato lì dentro. Ci basterà selezionarlo e sulla destra ci verranno fornite le caratteristiche che ci confermeranno l'avvenuta riuscita del caricamento. Se invece abbiamo dato un nome diverso come nel mio caso (beacon) dobbiamo andare su *create database*

in celeste e mettere lo stesso nome che abbiamo scelto in precedenza. A questo punto, selezionando il database *beacon* avremo una schermata come questa:

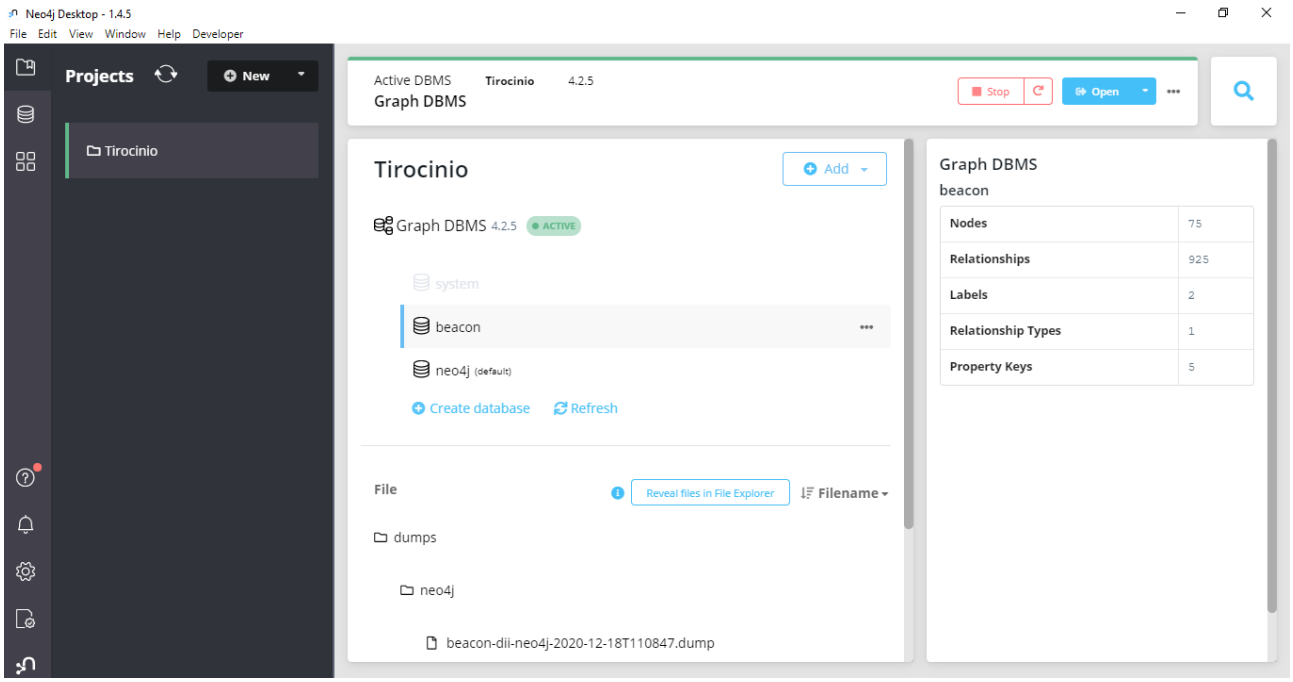


FIGURA 22: DATABASE INSERITI IN NEO4J

Per iniziare a lavorare sul database andiamo su open in alto a destra. Si aprirà una nuova schermata come la seguente:

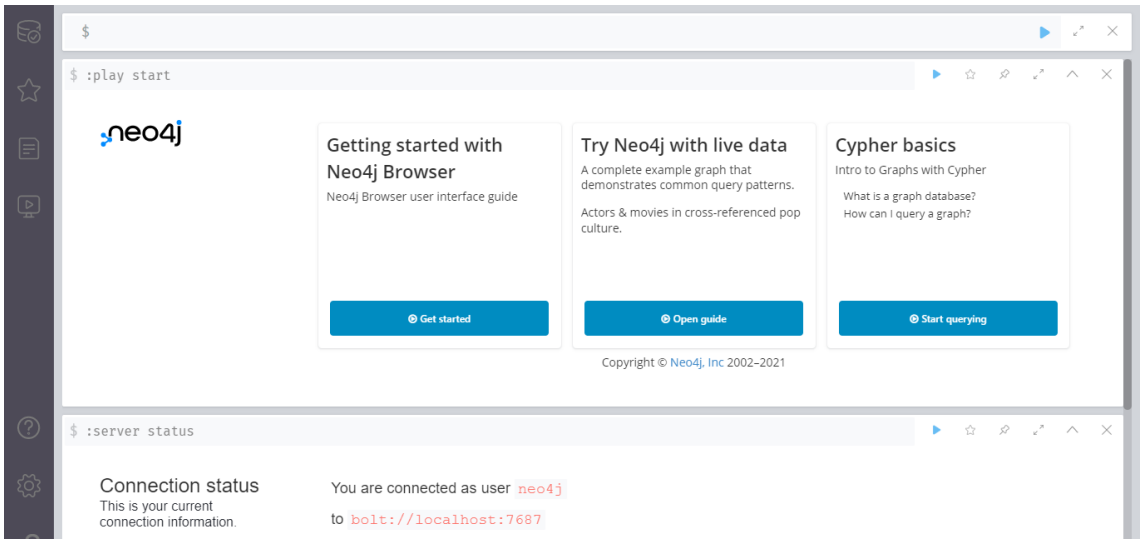


FIGURA 23: NEO4J DASHBOARD

Sulla sinistra clicchiamo sul primo simbolo e dal menu a tendina scegliamo il database su cui vogliamo lavorare.

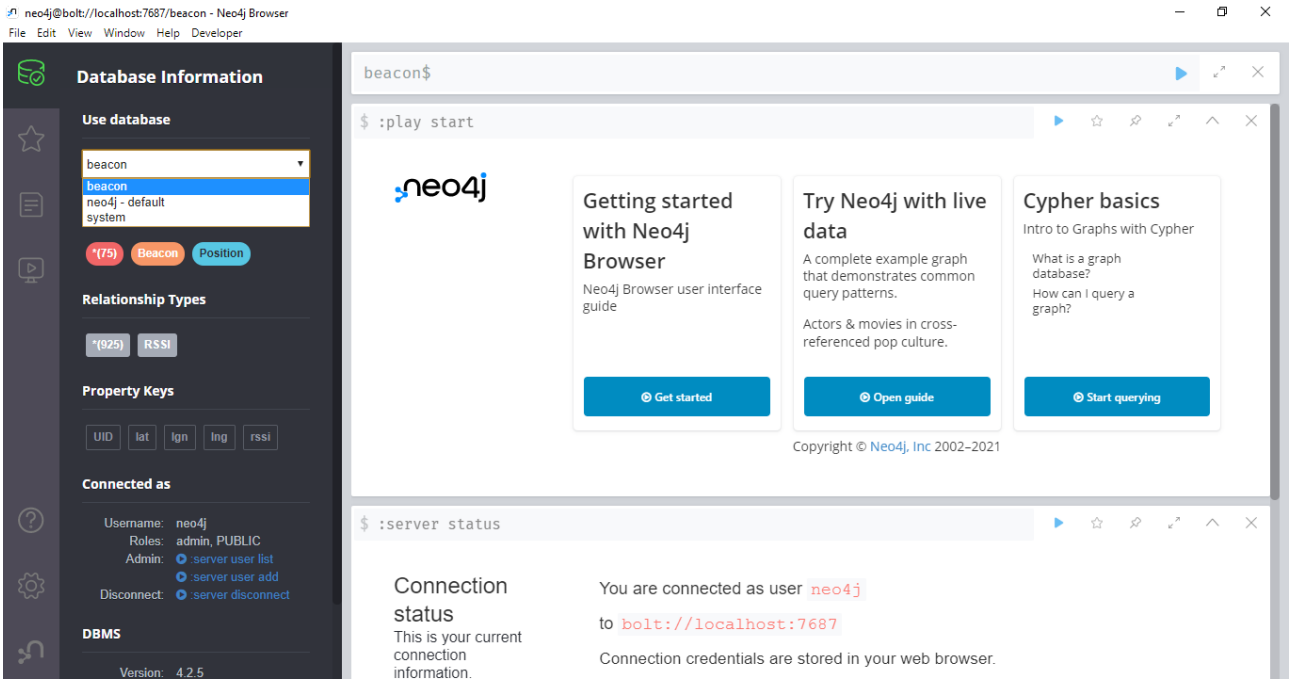


FIGURA 24: SCELTA DEL DATABASE

Una volta selezionato, ci appariranno i numeri che rappresentano i nodi. Cliccandoci ci apparirà l'interfaccia grafica.

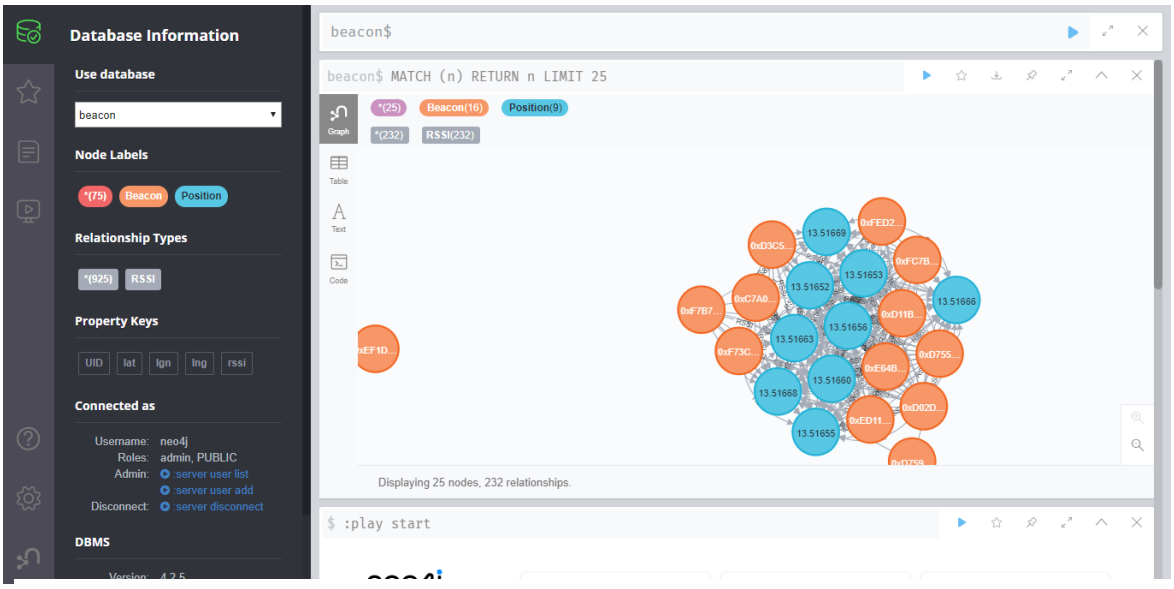


FIGURA 25: IL NOSTRO DATABASE NELLA SUA VESTE GRAFICA

Inizialmente ne appariranno solo 25 a causa del limite impostato. Possiamo modificarlo, cambiando il numero e ponendolo uguale al numero segnato in rosso sulla sinistra (ovvero il numero totale dei nodi).

Per creare un nodo e delle relazioni possiamo inserire i comandi nella barra in alto con il nome del database ed il simbolo del dollaro nel mio caso *beacon*$. Per esempio:

```
CREATE (n:Person {name: 'Andy', title: 'Developer'})
```

7. Bibliografia

- Documentazione Kubernetes: https://kubernetes.io/it/docs/
- Documentazione NEO4J: https://neo4j.com/developer/
- Manuale di Cypher: https://neo4j.com/docs/cypher-manual/current/
- Documentazione Helm: https://helm.sh/docs/
- Documentazione Minikube: https://minikube.sigs.k8s.io/docs/start/
- Documentazione Docker: https://docs.docker.com/
- Documentazione NGINX: https://nginx.org/en/
- Documentazione Kubectl: https://kubernetes.io/docs/reference/kubectl/