
Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale
In Ingegneria del Software
A.a. 2022/2023
Sessione di Laurea Luglio 2023

# Realizzazione di un sistema 'homecloud' per piccole imprese e reti domestiche

Relatori:
Enrico Masala
Marina Mondin
Fereydoun Daneshgaran

Candidato
Riccardo Giambra

# Indice

# Indice delle figure

# Indice delle tabelle

# Introduzione

Alla base di questo studio vi è l'intenzione di analizzare ed implementare un sistema informativo per il gruppo di ricerca della California State University Los Angeles, dipartimento di telecomunicazioni. La necessità di suddetto sistema è legata allo sviluppo di progetti di robotica che vedono la produzione di grandi quantità di dati di sensoristica da analizzare e la scrittura di codice, utile per eseguire vari task legati al settore, che deve essere opportunatamente versionato. In generale serve disporre di sistemi di archiviazione collocati in maniera tale da tutelare la privacy e la sensibilità delle informazioni che contengono.
Le motivazioni che mi hanno spinto ad approfondire tale tema hanno comunque una natura duplice, da una parte si vuole offrire all'università tale strumento in grado di essere un ottimo supporto alla ricerca, d'altro canto, si vuole mettere in evidenza un'alternativa molto semplice ed economica ai sistemi di archiviazione cloud pubblici a cui siamo abituati, proponendo dunque la seconda scelta che ognuno avrebbe di poter installare un cloud privato, gestito in toto in maniera individuale vedendo un costo fisso iniziale per poter acquisire i materiali necessari e successivamente dei costi variabili molto ridotti e legati quasi esclusivamente alla spesa energetica per tenere attivo il sistema, che comunque rappresenta un prezzo quasi irrisorio.
Pertanto, l'obbiettivo di questa tesi di laurea è quello di realizzare una soluzione che riesca ad essere sufficientemente performante e che risponda ai requisiti di ricerca del dipartimento di telecomunicazioni, ma allo stesso tempo richieda del materiale facile da reperire in maniera tale da poter implementare una soluzione analoga anche in contesti domestici o in piccole reti aziendali.
Per poter raggiungere tali compromessi, sono state analizzate varie alternative che spaziano dall'hardware necessario, scendendo fino al dettaglio del tipo di connettori dei dischi di memoria migliori per il contesto, fino al software, coprendo alternative caratterizzate da versioni Enterprise ma anche e soprattutto scelte opensource.
Uno dei principali obbiettivi nell'implementazione sarà quello di creare un ambiente che possa essere plug and play e facile da replicare per sviluppi successivi, grazie ad un utilizzo intensivo di container Docker che permetta la creazione e configurazione di ambienti tramite file descrittori che vengono poi utilizzati dall'engine di Docker compose
L'elaborato sarà quindi articolato in cinque capitoli, iniziando con una descrizione del cloud privato rispetto al cloud in leasing ragionando soprattutto in termini economici, per poi procedere con un approfondimento sull' hardware necessario per poter creare un web server riuscendo a sfruttare quasi esclusivamente quello fornito da un Raspberry PI, con l'aggiunta di dispositivi di archiviazione in termini di dischi. Verranno successivamente trattate le componenti software necessarie per poter erogare i servizi richiesti in termini di cloud storage, version control e data analysis. Nell'ottica di permettere l'accesso a tali risorse anche da remoto, sarà necessario discutere anche l'organizzazione di rete del sistema mediante l'uso di una VPN e tecniche di raggiungibilità basate su IP statico o DDNS. L'ultimo capitolo descriverà in maniera pratica l'infrastruttura progettata ed implementata per poter rispondere alle richieste dell'università.

# Capitolo 1: Backbone

Negli ultimi tempi il termine “Cloud” è sulla bocca di tutti ma molto spesso la sua descrizione tecnica è qualcosa che resta nelle “nuvole”. Il termine Cloud è nato come uno slang nel settore tecnologico, spesso infatti quando si rappresentano architetture di rete, l’internet viene rappresentato come una nuvola, man mano i processi elaborativi sono passati su internet, è diventato comune dire “spostare le cose al Cloud”. Secondo una definizione più formale fornita dal dizionario Treccani, il cloud computing è una tecnologia *che permette di elaborare e archiviare dati in* rete (https://www.treccani.it/enciclopedia/cloud-computing).

Il cloud è frutto di una serie di necessità della società attuale a cui la tecnologia risponde con opportunità in grado di risolverle. Nel dettaglio, al giorno d’oggi l’evoluzione tecnologia porta alla necessità di avere a disposizione sempre hardware più performante in grado di compiere task complessi, inoltre gli stessi dati da gestire ed immagazzinare tendono ad aumentare sempre più il livello di dettaglio per cui è necessario disporre sempre di storage più elevato. Basti pensare banalmente al peso di una immagine, fino ad una decina di anni fa queste avevano una risoluzione di 480p (ossia il lato corto dell’immagine aveva 480 pixel), oggi le foto raggiungono risoluzioni full HD (ossia 1080p) mentre nei prossimi anni vedremo, e già vediamo, foto a 2k, 4k, 8k. È facile notare, anche con questa misura grossolana, come in pochi anni la necessità di storage sia aumentata esponenzialmente e il trend non accenna a fermarsi, anzi, viene anche incrementato da fattori social.

Dall’altra parte c’è l’opportunità che la tecnologia offre, ossia costi di memorie sempre minori per cui è possibile comprare dischi da centinaia di Gb o qualche Tb a poche decine di euro.

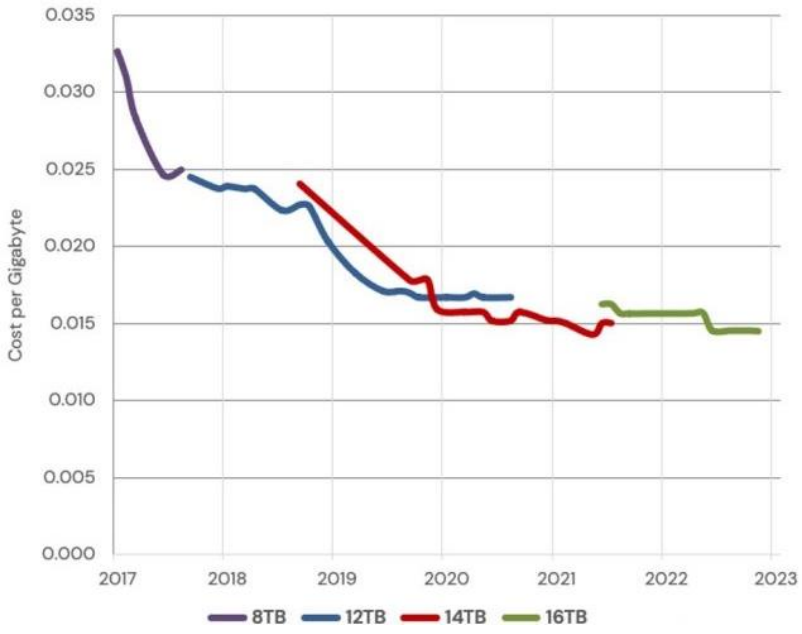


*Figura 1: Trend del costo per Gigabyte delle memorie (in decrescita costante)*

Ma la soluzione non sarebbe comunque sufficiente, perché la necessità della società è quella di poter avere tutti i propri dati sempre a portata di mano, accedervi quindi anche tramite il proprio smartphone eventualmente.

Anche in questo caso la tecnologia risponde con un'altra opportunità, ossia la rete. Ad oggi, infatti, è possibile accedere anche tramite rete cellulare ad una velocità verosimilmente intorno ai 100Gbps (con la rete 5G in rapida diffusione), per cui è possibile collocare l'hardware necessario in un determinato punto geografico ma poi averne accesso come se fosse una nuvola che segue l'utente. Il discorso poi può essere facilmente ampliato non solo allo storage ma anche alla potenza di calcolo, introducendo il concetto di macchina virtuale (VM).

### *Macchina Virtuale*

Si tratta di una organizzazione logica delle risorse che permette di lanciare sulla stessa macchina ospite varie istanze che si comportano ciascuna come se fosse un computer reale con il proprio hardware a disposizione. Queste vengono lanciate in modalità sandbox, quindi anche se l'hardware fisico risulta condiviso, le VM non interagiscono tra loro, per cui è impossibile che vi sia visibilità di file o documenti da una all'altra. Grazie a questa tecnologia di virtualizzazione è possibile per un cloud provider avere su uno stesso server fisico tanti server virtuali per cui è come se un grosso datacenter contenesse al suo interno numerosi datacenter virtuali. Tutto ciò mette le basi per poter creare un vero e proprio business dove le risorse informatica vengono erogate come se fossero un servizio, caratterizzato da una grandissima affidabilità, dove è possibile introdurre ridondanza e avere intere copie di backup della VM su altre VM

## *1.1Cloud as a Service*

Intorno al cloud, viste le sue grandi potenzialità, si è creato un vero e proprio business, che eroga il cloud come se appunto fosse un servizio. È possibile quindi classificare sulla base del modello di servizio in tre categorie:

- Infrastructure as a Service (IaaS): viene fornita solo l'infrastruttura, in altri termini, il server a cui si accede tramite macchina virtuale, l'esempio tipico sono i servizi EC2 ed S3 di AWS. Volendo fare un paragone edile, è come se una impresa stia noleggiando il terreno su cui fare poi una casa.
- Platform as a Service (PaaS): viene fornita anche la piattaforma, in termini di sistema operativo, ambiente di programmazione o servizi comuni (come ad esempio il database). Viene usato dagli sviluppatori per rilasciare. Per mantenere l'analogia con il modello edile, è come se una impresa, oltre al terreno, stia noleggiando anche i mezzi e la strumentazione per costruire la casa.
- Software as a Service (SaaS): viene fornito un software a cui l'utente accede tramite una interfaccia grafica e per il quale serve una configurazione minimale. È dedicato direttamente agli end user e gli esempi principali sono i servizi forniti da Dropbox, Google Drive ecc. A questa categoria rientrano anche quelle applicazioni che permettono di usare un servizio senza scaricare direttamente l'app sul proprio dispositivo, ad esempio la suite di Microsoft office accessibile completamente dalla sua versione online. Concludendo il paragone edile, in questo caso è come se l'utente finale stia affittando una casa bella che costruita.

Negli ultimi tempi, sta aumentando l'interesse verso un quarto modello di servizio, il cosiddetto Function as a Service (FaaS), chiamato anche Serverless Computing. In questo caso vengono suddivise le applicazioni cloud in componenti più piccoli (funzioni) che vengono usate solo quando necessario. Forzando un po' la mano con il paragone edile, è

come se si pagasse ogni camera solo quando la si usa, e inoltre nel caso in cui fossero necessarie più risorse, le funzioni possono aumentare o duplicarsi (come se, nel caso ci fossero più commensali, vengano aggiunti più posti nella sala da pranzo). Il modello è chiamato serverless perché le applicazioni non girano su un server dedicato ma appunto, sulla base delle esigenze, acquisiranno più o meno risorse, o nessuna, nel caso non ci fossero richieste.

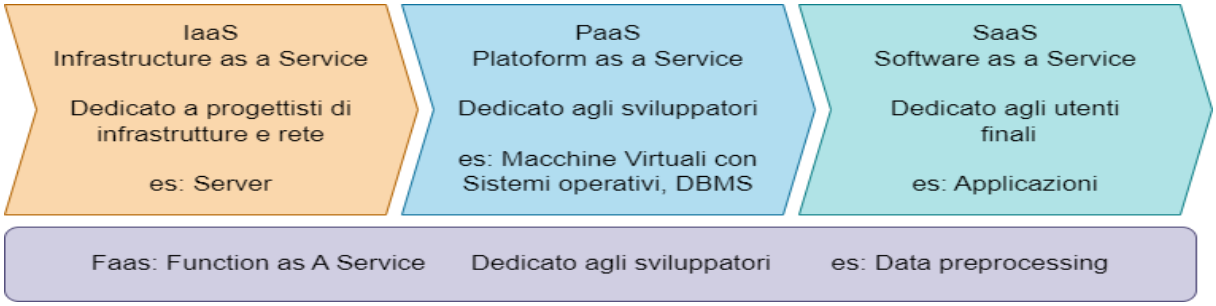


*Figura 2: Rappresentazione grafica delle varie tipologie di cloud service*

### *1.1.1 Cloud pubblico*

Il concetto dietro al cloud si potrebbe dire relativamente semplice, tuttavia, la sua implementazione a livello di hardware potrebbe non essere alla portata di tutti, inoltre un utente potrebbe non sapere a priori di quante risorse avrà bisogno, per cui da una parte potrebbe puntare a creare un sistema sovradimensionato, dall'altra parte potrebbe fare uso di un sistema cloud pubblico. Questa idea fu concepita per la prima volta da Amazon, che mise a disposizione le proprie risorse in leasing tramite il servizio di Amazon Web Services (AWS). Di base all'utente vengono allocate un insieme di risorse che questi potrà incrementare con dei semplici click solo se ne avrà bisogno. Viene quindi introdotto il modello pay-per-use nelle risorse informatiche.

Naturalmente l'accesso a tali risorse ha un costo, ci sono vari fornitori di cloud oltre AWS (ad esempio Microsoft Azure) ma per una analisi economica ci si limita a guardare le tariffe che offre AWS, in quanto rappresenta il fornitore con più esperienza del settore. Si indicano i prezzi relativi a storage, potenza di calcolo e trasferimento di rete con dovuti arrotondamenti (per difetto):

- Storage (S3): 0,02 $ per GB al mese (https://aws.amazon.com/it/s3/pricing/)
- CPU (EC2-a1): 0,02 $ per ora per core (https://aws.amazon.com/it/ec2/pricing/on-demand/)
- Rete (inclusa in EC2): 0,09 $ per GB al mese (https://aws.amazon.com/it/ec2/pricing/on-demand/) [Solo in uscita, in ingresso è gratuito]

Supponiamo di creare un caso di studio con 2 TB di storage e CPU con 4 core, si raggiunge un costo mensile 40,96$ per lo storage e 57,60$ per la CPU. È possibile risparmiare somme ingenti (anche fino al 70%) facendo una prenotazione delle risorse e quindi dichiarando con precisione di quanto si necessita in anticipo. Supponendo di avere un risparmio del 70% e avendo tali spese (approssimate per difetto), su base annua, il costo si aggira intorno ai 350$. Non è inoltre nota la locazione geografica della macchina in utilizzo che sarà installata in uno dei centri di AWS sparsi per il mondo, è possibile inoltre il servizio di Content Delivery Network (CDN) che permette di distribuire il contenuto su vari nodi per ridurre la distanza media tra l'utente che accede al servizio e la macchina su cui è rilasciato.

### *1.1.2 Cloud Privato*

La struttura proposta fino ad ora era caratterizzata da un modello pay-per-use che permette di pagare un determinato prezzo su base mensile. Un risultato completamente diverso si potrebbe ottenere acquistando un server fisico e installarlo a casa propria o nella propria azienda; quindi, piuttosto che avere delle risorse condivise pubblicamente tra vari utenti, le risorse sono interamente dedicate all'utente o all'azienda che lo istalla. Una scelta del genere naturalmente ha come principale svantaggio il fatto che l'intera manutenzione del sistema è a carico del proprietario. Tuttavia, vi sono molti vantaggi prima di tutto, dal punto di vista economico, tale soluzione richiede una spesa principalmente iniziale. L'esempio di prima non era casuale: il Raspberry PI 4 usato per gli esperimenti di questa tesi è un modello a 4 core con 8gb di RAM, che costa circa 80$, al quale saranno collegati due dischi da 2 TB (connessi in modalità RAID) del costo di circa 40$ ciascuno. Oltre questo, gli unici costi mensili da affrontare restano la connessione dati e la spesa elettrica per alimentare il sistema (che deve restare always on). Si nota subito che, seppur le spese di AWS fossero state arrotondate per difetto e vi si fosse applicato uno sconto del 70%, il risparmio è più che evidente, inoltre il sistema costruito può tranquillamente restare attivo per anni, mentre naturalmente con AWS sarà necessario pagare mese per mese.

Bisogna comunque considerare che i servizi di AWS garantiscono elevate affidabilità dei loro sistemi per cui è estremamente improbabile che si verifichino dei fallimenti, mentre banalmente se andasse via la luce, un sistema cloud privato andrebbe giù, ma comunque resterebbe possibile alimentare il tutto con un gruppo di continuità che permetta di sopperire ad eventuali imprevisti di questo genere.

Un vantaggio non indifferente del cloud privato è legato proprio al fatto che tutti i dati sono contenuti in dischi sotto il completo controllo dell'utente proprietario e nessun altro, il che lo rendo un ottimo strumento per poter salvare dati sensibili, magari progetti su cui una azienda sta lavorando che ne rappresentano il core business. Se in qualche modo i server di AWS o di Google Drive venissero attaccati e ci fosse un data breach, anche i propri dati ne sarebbero vittima, senza che l'utente proprietario ne abbia alcuna colpa o responsabilità.

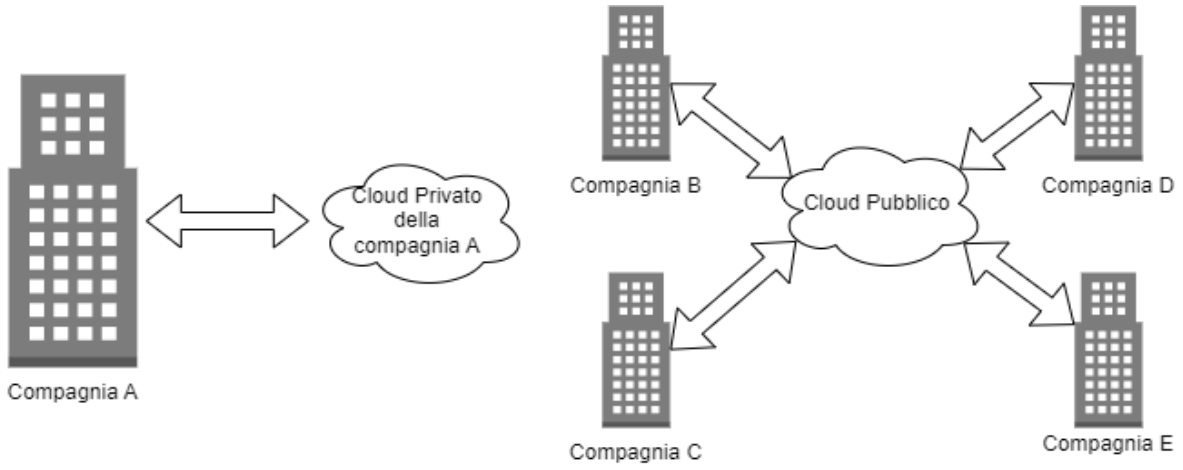


*Figura 3: Confronto tra cloud pubblico e privato*

### *1.1.3 Cloud ibrido*

Questa soluzione è un'unione delle due precedenti, una azienda potrebbe decidere di implementare alcuni servizi sul proprio cloud privato (includendo quei servizi che rappresentano probabilmente il core business dell'azienda stessa) mentre altri su cloud pubblico (ad esempio sfruttando strumenti come Teams, Google Meet per le call). Si potrebbe anche sfruttare il cloud pubblico come backup del cloud privato.

### *1.1.4 Multi Cloud*

Ultima variante che semplicemente consiste nell'usufruire di vari servizi di Cloud erogati da fornitori differenti, eventualmente in combinazione con implementazioni di cloud multi-ibrido.

### *1.1.5 Cloud privato virtuale*

Spesso viene usata questa terminologia per indicare un servizio di cloud pubblico, in cui vengono creati degli ambienti completamente isolati per i propri utenti. L'utilizzo del termine privato in questo caso può risultare fuorviante ma appunto, essendo definito come virtuale si fa riferimento, comunque, al fatto che comunque si opera su una architettura pubblica.

## *1.2 Caso di studio*

Dalla classificazione discussa fino ad ora emerge che naturalmente non esiste la soluzione sempre ideale ma è necessario al solito analizzare il caso di studio per poter progettare e implementare la soluzione adatta e commisurata per rispondere alle esigenze. Nel caso in oggetto di questo elaborato, il dipartimento di telecomunicazioni della California State University necessita di uno spazio privato e sicuro nel quale salvare le informazioni sensibili relative ai propri progetti. Il Cloud pubblico in questo caso, seppur rappresenta una semplice soluzione nel suo modello SaaS, potrebbe però non offrire la garanzia di privacy e sicurezza che offre un sistema privato installato direttamente nella rete dell'università o in piccole reti azienali. Per rafforzare la sicurezza e non esporre pubblicamente le applicazioni sarà necessario instaurare una VPN. Le richieste, in termini di servizi, riguardano uno spazio di memorizzazione di abbondante capienza, un sistema di version control e la possibilità di accettare flussi di dati di sensoristica su cui effettuare analisi e produrre report. Per poter riuscire ad includere tutti questi servizi, sarà necessario prima di tutto disporre della memoria per poter ospitare una grande mole di dati, per cui si ricorrerà all'implementazione di una NAS.

### *1.2.1 Virtual Private Network (VPN)*

Si tratta di una particolare architettura di rete che permette di avere i vantaggi di una rete privata, come ad esempio l'organizzazione personalizzata degli indirizzi e l'isolamento dalla rete pubblica, tra host che non appartengono alla stessa rete, il tutto attraversando in maniera sicura la rete pubblica. Per comprendere meglio, è necessario introdurre come avviene lo scambio di dati attraverso internet. Nei mezzi fisici, passano sequenze di bit, a queste viene associato il significato logico di pacchetto, diviso in header e payload. Il primo

contiene metadati e info di routing, mentre il secondo rappresenta la vera e propria informazione trasmessa. La struttura logica più diffusa è quella indicata dallo standard OSI, che introduce fino a sette livelli di astrazione, dove a partire dal livello applicazione, il pacchetto del livello più basso è ottenuto a partire dal pacchetto più alto a cui viene aggiunto un ulteriore header (che lo incapsula), così via fino al livello fisico, dove non viene aggiunto nessun header ma il pacchetto viene trasmesso nel mezzo fisico di trasmissione usato (wireless o cablato). Alla luce di ciò, ogni layer ha i suoi protocolli che permettono ai vari utenti di accordarsi sul significato dei bit negli header. I protocolli più usati (quasi esclusivamente) sono:

- Livello 2 (Collegamento): MAC (Media Access Control)
- Livello 3 (Rete): IP (Internet Protocol)
- Livello 4 (Trasporto): TCP (Trasmission Control Protocol), UDP (User Datagram Protocol)

Quando si opera nei confini di una Local Area Network (LAN), l'indirizzamento viene fatto tramite MAC e gli IP dei vari dispositivi apparterranno a delle particolari subnet (insieme di IP caratterizzato da una specifica sequenza di n bit iniziali) non raggiungibili dall'esterno. Un esempio è la sottorete 192.168.0.0/16, dove appunto i primi 16 bit (192.168) la caratterizzano.

Quando si parla di VPN, si vogliono ottenere le stesse caratteristiche di una rete privata, in termini di sicurezza ed indirizzamento, tra host dislocati su diverse subnet, per farlo, la tecnica sfrutta il concetto di encapsulation, ossia al pacchetto di livello n che suppone di essere all'interno della rete privata, viene aggiunto un altro header di livello n che permette di viaggiare attraverso alla rete pubblica, fino ad un gateway che si occuperà di rimuovere l'ultimo header inserito e che tratterà il pacchetto come se fosse stato generato da un host all'interno della rete privata.

La VPN può essere stabilità a diversi livelli, quella che sarà analizzata e approfondita nei prossimi capitoli sarà la VPN a livello applicativo, tramite il protocollo WireGuard, questa scelta permette di, operando a livelli alti, non toccare livelli che sono gestiti in automatico dal kernel del sistema operativo, ottenendo quindi una soluzione altamente personalizzabile ma comunque estremamente veloce, sicura ed efficiente.

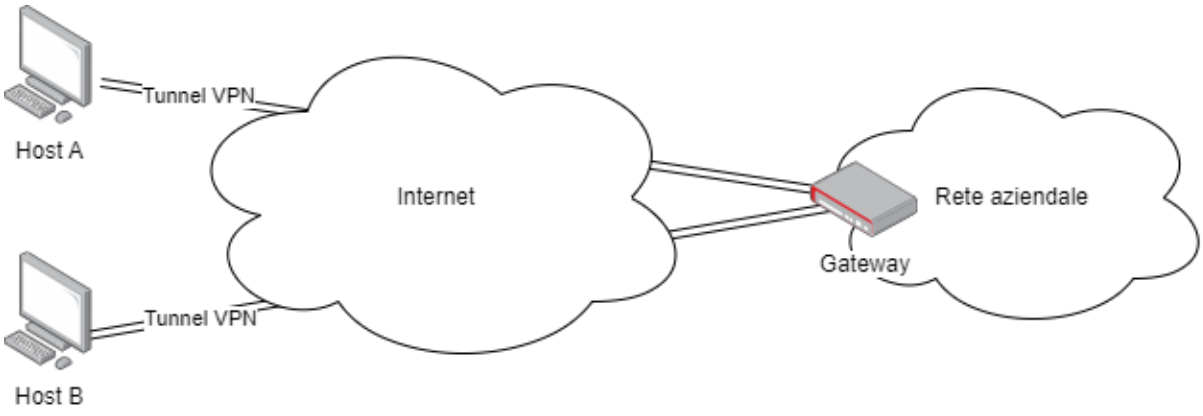


*Figura 4: schema di base di una VPN di tipo Remote Access dove vari utenti (end) si collegano alla rete aziendale (site)*

### *1.2.2 NAS*

Una NAS, ossia Network Attached Storage, è una soluzione per poter accedere a dei dischi da remoto attraverso la rete. Si tratta quindi di una opzione relativamente semplice, efficiente e scalabile. Sono disponibili dei dispositivi plug-and-play che permettono di

offrire tale servizio, si tratta infatti di un rack per dischi che viene collegato verso una interfaccia di rete e subito permette l'accesso alle risorse dei dischi con un sistema di autenticazione, tramite una web app installata di default e l'uso di protocolli dedicati come NFS, SMB o AFP. Questa rappresenta una delle possibili opzioni per il sistema richiesto, tutta via la semplice soluzione plug and play rappresenta solo uno dei servizi richiesti, per cui si preferirà implementare una NAS tramite l'utilizzo di un rack che ha il solo compito di collegare gli hard disk (oltre che offrire una forma di affidabilità sui dati grazie a RAID) e una web app installata sul server in grado di accedervi.

### *1.2.3 Docker*

Docker è un'applicazione sviluppata dall'omonima organizzazione che permette di automatizzare e ottimizzare gli sviluppi e rilasci di ambienti preconfezionati. Il suo compito è simile a quello di una macchina virtuale, dove quindi viene scaricato un sistema operativo con eventuali applicazioni/servizi, vengono configurate variabili di ambiente, esposte porte di rete e così via. Il principale vantaggio rispetto ad una VM è la possibilità di avere un sistema operativo minimale, che abbia quindi le sole funzionalità di cui necessità l'applicazione rilasciatavi.

Un esempio potrebbe essere il rilascio di un'applicazione sviluppata tramite Node, per il rilascio si potrebbe creare un ambiente Linux in cui poi si scarica Node (per la build) ed Nginx (per ospitare il web server). Con la filosofia dei container Docker, basterà invece creare un container che abbia lo stretto indispensabile per poter usare Node, creare la build e quindi spostare tale applicazione buildata in un altro container, con Nginx, che si occupa di renderla disponibile in rete.

Quando si lavora con Docker è necessario avere chiara la sua filosofia; infatti, vi sono due concetti fondamentali su cui questa si basa: immagine e container. La prima rappresenta una descrizione astratta della VM che si vuole creare, mentre il secondo è la vera e propria istanza della VM, si potrebbe paragonare l'immagine ad un programma mentre il container ad un processo, o meglio ancora, l'immagine ad una classe della programmazione ad oggetti mentre il container ad un vero e proprio oggetto.

Uno dei principali servizi messo a disposizione da Docker è Docker hub, uno spazio in cui gli utenti possono caricare le proprie immagini o scaricare quelle degli altri, in maniera simile a GitHub per il codice. Grazie a Docker Hub è possibile individuare le immagini di cui necessita, ad esempio quella di Node o di Nginx, e scaricarla in automatico in fase di build di un container.

Docker inoltre implementa un efficientissimo sistema di caching che permette dunque di eseguire solo le istruzioni necessarie a partire dallo stato precedente del container, questo accelera sensibilmente i tempi di build e rilascio.

È stato messo a disposizione anche uno script Python chiamato Docker-compose che permette di lanciare simultaneamente vari container tramite un file di configurazione unico, permettendo anche di configurare il networking tra le varie VM e inoltre permette di poter fare interfacciare il container direttamente con il file system della macchina host tramite il concetto di Volume, per cui se ad esempio si creasse un container con il DB, sarà possibile indicare in quale direttorio del file system della macchina host debbano essere salvati tutti i dati, piuttosto che lasciarli tenere in pancia al container stesso.

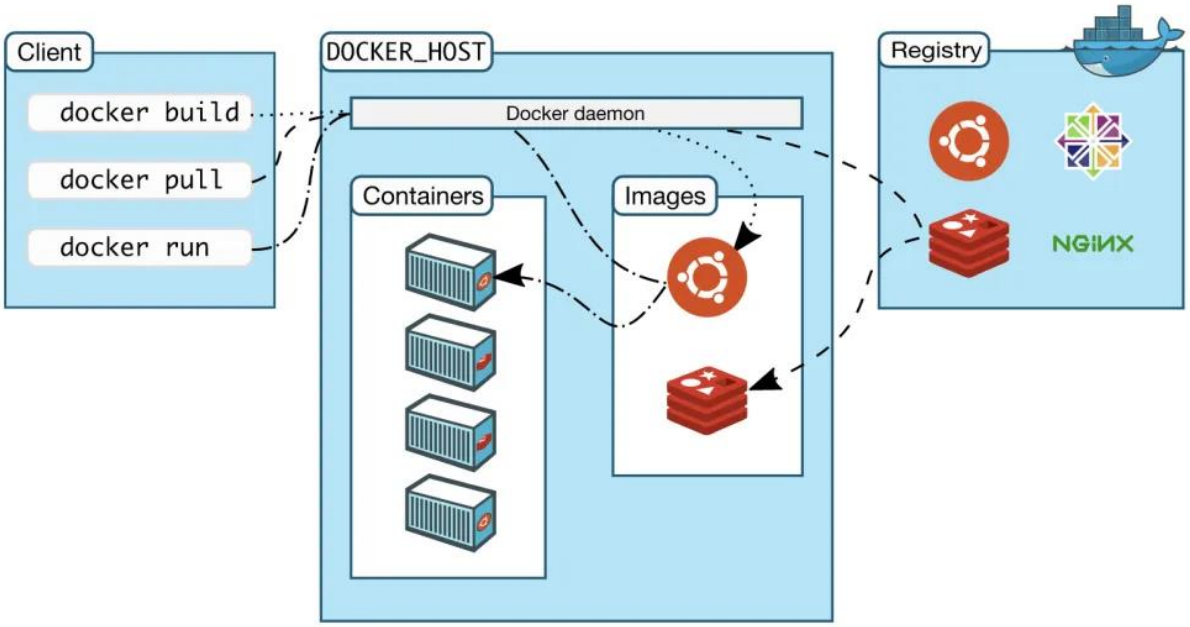


*Figura 5: Flusso di base della gestione di container e immagini Docker, con riferimento a Docker Hub sulla destra e comandi di base sulla sinistra.*

Rispetto al cloud pubblico, l'elemento chiave del cloud privato risiede proprio nella necessità di acquisire del vero e proprio hardware. Forse questo rappresenta per certi punti di vista il principale svantaggio di tale soluzione, i costi infatti sono legati ad acquisire tale materiale, manutenerlo ed alimentarlo. L'idea resta comunque quella di creare una soluzione semplice ed economica, per cui di seguito si analizzano le principali alternative e le conclusioni migliori per il caso in analisi.

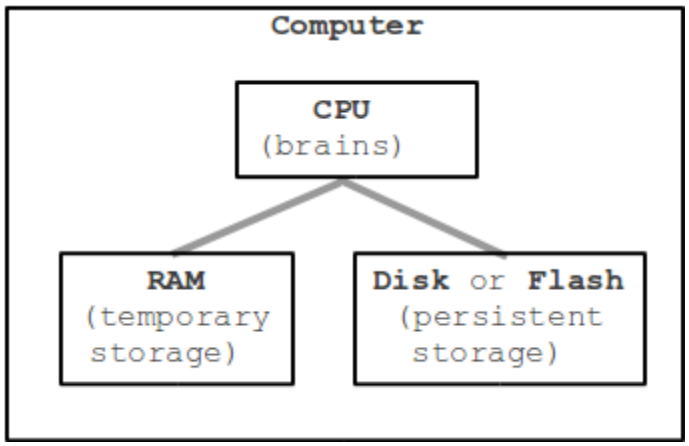


*Figura 6: Schema ad alto livello dell'architettura hardware necessaria*

## *2.1 Unità computazionale*

Per cominciare, si analizzano le alternative legate al "cervello" del sistema. Si cerca quindi un buon compromesso in termini di costi e performance nel tentativo di emulare un vero e proprio web server.

### *2.1.1 Hardware dedicato*

La prima alternativa è infatti quella di acquisire un computer o un server per poter effettuare tutte le operazioni necessarie. Le risorse per tale sistema dovrebbero essere dimensionate in funzione del numero di utenti che il sistema prevede di gestire. Immaginando un contesto piccolo con circa una decina di utenti connessi simultaneamente, è già sufficiente un sistema con 8GB-16GB di RAM e un processore quad-core in grado di poter effettuare in parallelo le azioni richieste da tali utenti.

Il sistema deve essere concepito per poter gestire lo stress dell'always on che un server richiede per essere costantemente raggiungibile, per cui sicuramente è necessario avere un buon sistema di raffreddamento ma bisogna anche valutare correttamente l'alimentazione necessaria al funzionamento in maniera tale da non avere impatto troppo elevato sul costo mensile della soluzione.

Il principale vantaggio di tale scelta è legato alla grande possibilità di personalizzazione, vista la modularità del modello di base di un computer, e naturalmente è relativamente semplice effettuare un upgrade di ogni singolo componente per andare incontro ad evoluzioni del carico a cui viene sottoposto. D'altro canto, questa soluzione può

raggiungere costi fissi di anche centinaia  o migliaia di dollari, per cui magari sembrerebbe più adatta per contesti più grandi rispetto ad un home cloud.

### *2.1.2 Raspberry PI*

Una semplice alternativa di tipo plug-and-play è proprio quella di utilizzare un Raspberry PI. Raspberry PI Foundation è un ente di beneficienza che si pone l'obbiettivo di creare dei dispositivi utili ai fini di studio e ricerca. A partire dal primo modello, Raspberry PI ha creato dei sistemi all-in-one plug-and-play offrendo una distribuzione di Linux personalizzata chiamata "Raspbian", caratterizzata da un efficiente utilizzo delle risorse relativamente limitate (almeno per i primi modelli). L'ultima linea di Raspberry PI prodotta è la numero 4 dove, rispetto al suo predecessore, il Raspberry PI 4 è disponibile in varie configurazioni di RAM e CPU, raggiungendo valori molto competitivi come 8GB di RAM e processore ARM quad-core da 1.8 GHz a 64 bit. Questo genere di architettura è stato proprio concepito per l'hosting di piccoli server always-on, visti i suoi limitatissimi consumi in termini elettrici ma anche la sua bassa tendenza a sviluppare calore, che rendono viabile un raffreddamento passivo e silenzioso, dettaglio che potrebbe sembrare trascurabile ma considerando la sua collocazione domestica, avere al contrario una ventola costantemente accesa potrebbe risultare disturbante. Il suo prezzo si aggira intorno agli 80$ e può tranquillamente essere alimentato dal caricatore di un cellulare a 5V. Sono disponibili 2 modelli, Compact Model e Model B, il primo non ha nessuna periferica di input, per cui vi si può accedere solo tramite SSH quando connesso al WI-FI, il secondo invece è montato su una scheda in cui vi sono 4 porte usb, una porta ethernet e 2 porte HDMI, per cui vi si può collegare mouse, tastiera e monitor oppure accedervi tramite SSH attraverso la porta Ehternet o il WI-FI, inoltre vi sono vari pin a cui è possibile collegare ulteriori dispositivi da alimentare (grazie a GND, 5V e 3.3V) oppure dei veri e propri pin GPIO.

Il sistema nel complesso tende ad operare a temperature molto basse per cui è sufficiente anche un semplice dissipamento passivo, nel caso di Raspberry PI model B resta comunque possibile collegare una piccola ventola per favorire l'aerazione, ma oltre ad essere relativamente superfluo, in un contesto domestico potrebbe risultare fastidioso avere costantemente attiva tale ventola, visto appunto il rumore.

Per quanto riguarda il Raspberri PI CM, è possibile acquistare per 10-20$ il modulo di I/O che introduce tutte le porte presenti anche sul PI model B ma in più ha anche una porta PCI express x1. Quest'ultimo dettaglio avrebbe potuto sbilanciare la scelta verso questo modello, soprattutto per quanto riguarda il collegamento dei dischi, che sarà trattato a breve. Tuttavia, il modello CM4 è relativamente nuovo per cui è completamente introvabile negli store online e fisici al momento della stesura di questa tesi.


*Figura 7: Raspberry PI CM4 inserito nella I/O module board, sulla destra vi è la porta PCIx1*

### *2.1.3 Scelta finale*

Alla luce di tale comparazione, la scelta migliore per il progetto finale ricade proprio nel Raspberri PI, visti i suoi costi ridotti con performance molto competitive. L'ideale sarebbe stato il PI CM4 incluso il modulo di I/O, visti i suoi consumi limitati, ottima potenza di calcolo e numerose porte di input, tra cui la PCIx1, ma essendo ad oggi completamente sold out, ai fini dell'implementazione sarà utilizzato un Raspberry PI model B dove verranno sfruttate le sue porte USB per i collegamenti agli altri dispositivi.


*Figura 8: Raspberry PI 4 Model B selezionato per il caso di studio*

## *2.2 Unità storage*

Obbiettivo fondamentale del sistema è quello di offrire una grossa capacità di storage. Generalmente quando si parla di cloud pubblico, almeno nei piani gratuiti, sono disponibili dai 5GB ai 15GB, espandibili appunto pagando una spesa mensile. L'obbiettivo del cloud privato è quello invece di offrire quantità di spazio nell'ordine dei terabyte o anche decine di terabyte, per cui è necessario selezionare con cautela il tipo di disco che si preferisce utilizzare.

### *2.2.1 SSD*

La prima tecnologia in analisi è quella dei Solid State Disk (SSD). Si tratta di una tecnologia di memoria non volatile simile alla ROM per certi sensi, ogni bit viene salvato attraverso lo stato di porte logiche implementate con transistor. Tutte le scritture e cancellazioni vengono eseguite attraverso fenomeni elettrici, questo garantisce una elevatissima velocità e quindi una bassa latenza del dispositivo. Le cancellazioni però richiedono che il transistor subisca un voltaggio molto intenso, basti pensare che normalmente per scrivere e tenere memorizzato il dato sono necessari circa 3,3-5V mentre una cancellazione richiede una operazione di "flash" con voltaggi superiori ai 20V. A causa di ciò il dispositivo tende a deteriorarsi in caso di numerose cancellazioni. Questa tecnologia ormai è molto diffusa ma, nonostante ciò, normalmente il prezzo per byte è relativamente alto, si parla infatti di circa 0.09$ per GB, contro i 0.03$ per GB di un HDD. Da una parte l'SSD è caratterizzato da una grande affidabilità, vista l'assenza di meccanica ma allo stesso tempo il dispositivo è molto fragile.

### *2.2.2 HDD*

La seconda alternativa più diffusa è quella degli Hard Drive Disk (HDD). Questa tecnologia è caratterizzata dalla presenza di dischi rivestiti di un materiale magnetico che viene inciso per salvare i singoli bit. Ogni disco è suddiviso in piatti, tracce e settori. Il piatto è una delle facce di un determinato disco, la traccia rappresenta una zona del disco caratterizzata da una determinata distanza dal centro mentre il settore è una porzione della traccia. I settori sulle varie tracce dei dischi vengono letti in parallelo dalla testina del dispositivo, per cui è necessario, per effettuare una determinata lettura, che il disco venga posizionato sulla corretta traccia e sul corretto settore. Per posizionare la traccia, la testina viene spostata in maniera radiale ai dischi, mentre per selezionare il settore i dischi vengono fatti ruotare rigidamente tramite lo spindle. Le letture avvengono in parallelo, quindi spesso vi sono 8 piatti da cui si leggono contemporaneamente 8 bit, e quindi un byte. Questo discorso fa comprendere come tale dispositivo abbia delle componenti meccaniche non indifferenti per cui il suo operare è caratterizzato sia da operazioni meccaniche che da operazioni elettriche. Quando quindi si usa un HDD bisogna tenere in considerazione il tempo per poter posizionarsi sul corretto settore, chiamato seek time, e il tempo per leggere tale settore, ricavabile tramite il transfer rate che rappresenta la quantità di GB al secondo trasferiti alla RAM. La presenza della meccanica rende quindi il dispositivo generalmente più lento ma, allo stesso tempo, l'organizzazione in dischi permette di avere un costo appunto più basso delle SSD per GB.

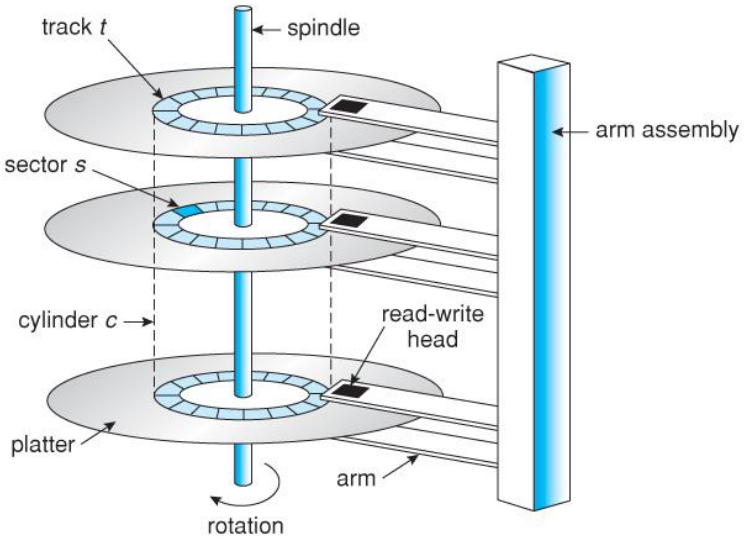


*Figura 9: Schema della struttura interna di un Hard Drive Disk (HDD)*

### *2.2.3 Nastri magnetici*

Questa tecnologia è ormai obsoleta ma consisteva su lunghi nastri che venivano scritti e letti in maniera sequenziale e per cui spesso vi era l'esigenza di un operatore manuale che cambiava i nastri. Viene citata per completezza ma l'utilizzo pratico di tale tecnologia potrebbe essere relativo solo alla creazione di backup, tutta via ad oggi un HDD può tranquillamente svolgere tale funzionalità.

## *2.3 Connettori*

Nella scelta dei dispositivi di storage bisogna tenere in considerazione una attenta selezione dell'infrastruttura necessaria per collegarli al server. La scelta del connettore ha impatti sulle performance, sui costi e sull'affidabilità generale del sistema, in termini di Mean Time Between Failures (MTBF). I connettori analizzati, che del resto sono i più diffusi, sono tutti caratterizzati da una trasmissione seriale piuttosto che parallela. Seppur intuitivamente si possa immaginare che una trasmissione parallela possa essere più veloce, questa viene spesso evitata in quanto i connettori seriali hanno molti più vantaggi. Primo fra tutti il risparmio, infatti l'utilizzo di una sola linea invece che molte permette di risparmiare su cavi e pin, inoltre le trasmissioni seriali, specialmente quelle differenziali, godono di una maggiore robustezza e resilienza al rumore che riduce notevolmente gli errori di trasmissione. La natura circuitale più semplice implica una maggiore difficoltà di gestione per cui sono necessari microcontrollori sui dischi e driver nel server che permettano di serializzare e deserializzare i dati.

### *2.3.1 SATA*

I dischi e quindi i connettori di tipo Serial ATA sono sicuramente i più diffusi in termini di Commodity On The Shelf. Molto spesso, infatti, nei Personal Computer vengono utilizzati proprio questi grazie alla loro capacità di collegarsi direttamente alla scheda madre. Vi sono tre generazioni di connettori SATA (I, II, III) dove le successive sono perfettamente retrocompatibili con le precedenti, naturalmente se venisse collegato un disco SATA III ad una interfaccia SATA I, quest'ultima farebbe da collo di bottiglia nella velocità di trasmissione. Le tre generazioni SATA, infatti, operano a velocità pari rispettivamente a 1.5 GB/s, 3 GB/s e 6 GB/s, questi rappresentano i limiti teorici, mediamente il vero e proprio range operativo è leggermente inferiore. Questo genere di disco è tra i più economici, fissando la dimensione a 2TB, è possibile trovare ad oggi dischi al prezzo di 50$ circa, tuttavia, i dischi di tipo SATA hanno un MTBF relativamente basso che si aggira intorno alle 700 000 ore, che possono anche aumentare a 1 200 000 a patto di provvedere un ottimo sistema di raffreddamento che stabilizzi la temperatura a non oltre 25°. Quest'ultimo sarebbe il principale difetto per operare in uno scenario di server always-on; tuttavia, supponendo una configurazione di tipo RAID, trattata in seguito, si possono raggiungere valori di affidabilità decisamente migliori.

### *2.3.2 SAS*

I dischi ed i connettori di tipo Serial Attacched SCSI sono tra i più usati per i server. Questo in quanto riescono ad offrire un MTBF generalmente più alto rispetto ai SATA, raggiungendo a temperature di 45° MTBF pari a circa 1 200 000 – 1 600 000 ore. Per grosse applicazioni di data storage inoltre permettono di utilizzare cavi più lunghi senza perdita di dati e, nonostante il limite teorico di velocità resti sempre circa 6 GB/s, operano mediamente più velocemente dei dischi SATA. Il difetto principale è legato al prezzo di tali dispositivi che può anche raggiungere il doppio, a parità di volume, rispetto ad un disco SATA. In generale si potrebbe riassumere il confronto fino ad ora affermando che i dischi SAS prioritizzano la velocità rispetto al volume di storage, al contrario dei dischi SATA che prioritizzano lo storage sulla velocità.

### *2.3.3 USB*

Universal Series Bus è lo standard che senz'ombra di dubbio ha trovato maggiore diffusione per i collegamenti dei vari dispositivi ad un personal computer. Viene utilizzato infatti per tastiere, mouse, stampanti, flash drive, external hard drive e così via. Viene praticamente supportato da qualunque dispositivo, per lo meno in una sua variante come USB type C per i cellulari, ma il principio di funzionamento resta sempre lo stesso: vi sono 4 pin che rispettivamente rappresentano l'alimentazione a 5v Val, il positivo della linea seriale D+, il negativo della linea seriale D- e la messa a terra GND. Naturalmente gestisce un canale differenziale ottenuto tramite la differenza tra le due linee dati. Sono disponibili numerose versioni, l'ultima più diffusa è la 3.0 che raggiunge i 5 Gbit/s facendo un grande salto di qualità rispetto ai 480 Mbit/s della versione 2.0 rendendo quindi USB una soluzione viabile per il collegamento di dischi. Vista la sua grande potenzialità in termini di compatibilità, USB nel contesto del cloud privato può vestire il ruolo di ponte per poter connettere dispositivi SAS o SATA alle porte USB.

## *2.4 RAID*

Come citato in precedenza, in sistemi di cloud storage è fondamentale avere una elevata affidabilità. La scelta più sicura per ottenere affidabilità dei dati salvati è quella di offrire ridondanza e quindi banalmente mantenere più copie del dato memorizzato.

Nei sistemi di trasmissione generalmente si offre affidabilità grazie a codici di correzione quali il bit di parità, che permette di rilevare un errore in una data sequenza di bit, aggiungendo solo un bit impostato a 0 o 1 in funzione del numero di bit pari ad 1 nella sequenza in oggetto. Tale tecnica riesce a adattarsi bene al contesto delle trasmissioni in quanto spesso vi sono delle perdite dei dati, per cui con un semplice bit è possibile ricostruire un bit perso per ogni sequenza.

L'idea di RAID, ossia Redundant Array of Indipendent Disk, è sempre legata all'aggiunta di ridondanza ma questa viene effettuata in maniera differente e si adegua al contesto dello storage in cui, visti i costi ormai molto contenuti dei dispositivi di archiviazione, ci si può permettere di duplicare interamente il contenuto di una porzione di spazio in un'altra. Una configurazione molto diffusa è chiamata RAID 0, in questo caso non vi è una vera ridondanza ma comunque l'organizzazione logica del contenuto dei dischi viene effettuata alternando porzioni di spazio dei vari dischi. Questo comporta che nell'ipotesi in cui ci siano due dischi e su uno di questi si manifesti un guasto, la metà dei dati è comunque salva. Questa forma di affidabilità è un buon punto di partenza ma resta comunque viva la possibilità di perdere, nell'esempio fornito, la metà dei dati. Le altre configurazioni diffuse sono RAID 1, RAID 5, RAID 6 che mantengono rispettivamente una, cinque, sei copie di ogni dato su altri dischi. Una ulteriore variante è RAID 10, dove 10 va letto come 1 e 0, in quanto non vi sono 10 copie dei dati, bensì vi è una combinazione tra RAID 1 e RAID 0. È importante sottolineare il significato della I in RAID, l'indipendenza sta infatti nel fatto che le copie dei dati vengono fatte su dischi differenti per cui il fallimento di uno non condiziona la perdita di dati dell'altro. Naturalmente per avere un utilizzo ottimizzato dei dischi nel vettore è necessario che questi abbiano tutti la stessa dimensione, altrimenti lo spazio di archiviazione visto dal sistema sarà pari alla minima capienza dei dischi.

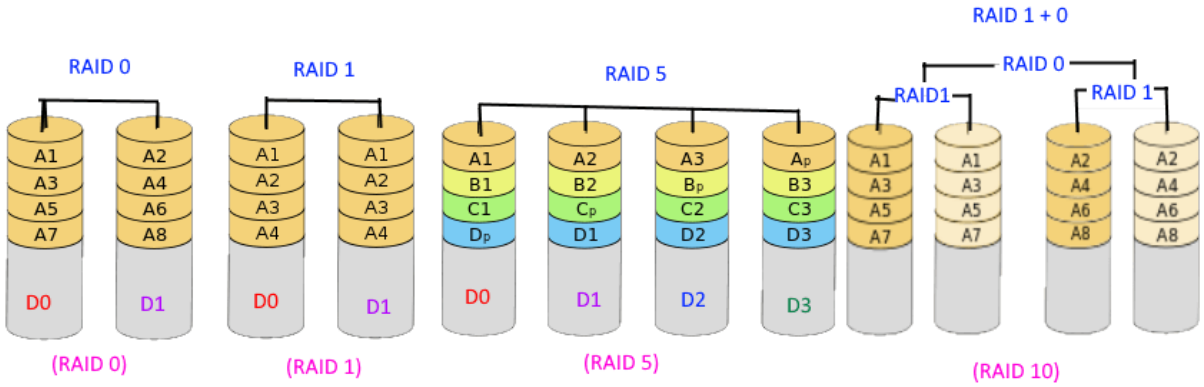


*Figura 10: Varie tipologie di RAID in una visualizzazione grafica dell'organizzazione del mirroring*

Parlando di performance, il MTBF, MTTF (Mean Time To Failure) e MTTR (Mean Time To Repair) rappresentano delle ottime metriche statistiche ed è possibile calcolarle in funzione di quelle degli hard disk di partenza, per semplicità si pone il caso di RAID1. La perdita effettiva dei dati avviene quando entrambi i dischi sono guasti nello stesso istante. Si calcola prima di tutto il $MTTF_1$ che rappresenta il tempo prima che uno dei due dischi abbia un failure:

$$MTTF_1 = MTTF/2$$

Banalmente, infatti, è la metà dell'MTTF di ogni singolo disco, si può anche leggere come l'inverso della probabilità che uno dei dischi abbia un failure. Fatto ciò, si può calcolare la probabilità che l'altro disco abbia un guasto durante la riparazione del primo:

$$PROB_{2\ fails\ during\ 1\ failure} = \frac{1}{MTTF} \times MTTR$$

Infine, la probabilità di perdere i dati è ottenuta dalla combinazione dei due eventi:

$$PROB_{data\ loss} = \frac{1}{MTTF_1} \times PROB_{2\ fails\ during\ 1\ failure}$$

Da cui sostituendo tutti i dati e invertendo si può calcolare il Mean Time To Data Loss in funzione del MTTF e MTTR:

$$MTTDL = \frac{MTTF^2}{2 \times MTTR}$$

Per avere un'idea dele cifre, sostituendo il MTTF di un disco SATA e supponendo un tempo per riparare il disco di 10 ore, si ottiene un valore di MTTDL di circa 3 milioni di anni, per cui si può affermare che già con RAID 1 l'affidabilità è elevatissima.

L'implementazione di RAID può essere effettuata sia via software che via hardware, la prima è generalmente più lenta perché richiedere che ogni operazione di scrittura venga effettuata due volte dal processore, mentre la seconda delega tale gestione ad un microcontrollore ottenendo performance decisamente migliori, quasi trasparenti per il sistema.

## *2.5 Conclusioni*

Tutte le alternative proposte sono valide ma per poter definire la migliore scelta è necessario calarsi nel caso di studio. Parlando dunque di un sistema homecloud che punta ad essere economico e altamente personalizzabile, in grado quindi di erogare numerosi servizi, il Raspberry Pi rappresenta la scelta più equilibrata come unita computazionale, avendo delle prestazioni competitive e dei consumi energetici minimi. Per quanto riguarda lo storage, per le grosse quantità di dati che si intende gestire e salvare, è più sensato puntare alla quantità di dati rispetto che alla velocità, per questo sono stati selezionati HDD con connettori di tipo SATA. Ai fini di avere un moderato quantitativo di memoria, sono stati selezionati 2 dischi da 2TB. Per poter colmare la bassa affidabilità degli HDD SATA si utilizzerà una configurazione RAID 1 tramite un dock disponibile ad un prezzo di circa 100$ che, oltre a tenere al sicuro i dischi e mantenere un ambiente fresco grazie alle ventole, si occupa di implementare RAID via hardware, ottenendo dunque uno speedup non indifferente. L'ideale sarebbe stato disporre di un PI CM4 per poter avere la porta PCIx1 nel I/O module, ma essendo introvabile si farà uso delle porte USB del PI per collegarvi tale dock.


*Figura 11: Dock per gli HDD SATA che implementa RAID 1 via Hardware acquistato per il progetto*

In seguito, vi sarà una descrizione dal punto di vista dei comandi per poter gestire al meglio lo spazio di archiviazione, dato che comunque si vuole garantire che gli utenti dell'applicazione possano lavorare in una zona confinata del sistema, si farà uso di partizioni dei dischi. Grazie all'implementazione RAID 1 via hardware, il sistema sarà completamente trasparente alla gestione della ridondanza e dal Raspberry PI saranno visibili solo 2TB.

# Capitolo 3: Software

Questo capitolo rappresenta gli aspetti centrali dell'implementazione delle varie componenti software. L'obbiettivo è quello di fornire una descrizione delle alternative disponibili per cercare di soddisfare ogni richiesta per poi descriverne la scelta finale.

Una prima ipotesi per l'intero progetto era quella di implementare tutti i servizi come funzionalità di un'unica applicazione React, ma il carico di lavoro rischiava di essere oneroso portando a tempi di consegna molto elevati.

La scelta è stata dunque quella di sfruttare delle applicazioni esistenti nella loro versione self hosted per offrire i vari servizi. Elemento chiave che consente tale scelta è Docker, grazie al quale è possibile scaricare delle immagini di ambienti in cui è rilasciata l'applicazione dichiarata. Bisogna evidenziare il fatto che le immagini Docker dipendono dall'architettura su cui vengono compilate, per cui nella scelta dei servizi, potrebbe essere determinante l'esistenza dell'immagine per l'architettura del Raspberry PI da una fonte ufficiale. In generale è possibile scaricare le immagini da Docker Hub, ma tali immagini potrebbero essere state caricate da chiunque, per cui è buona pratica accertarsi della fonte.

Durante gli sviluppi, per i primi test era stato utilizzato un Raspberry Pi 3 Model B, il quale era caratterizzato da una architettura ARM 32 bit. Il fatto che molto spesso non erano disponibili immagini ufficiali per tale architettura unito al fatto che il singolo gigabyte di RAM offerto da tale modello era troppo limitate, ha portato alla scelta del PI 4.

## *3.1 Storage*

Il primo nonché principale servizio richiesto per il cloud privato è quello dello storage. L'idea di base è quella di permettere l'accesso ad una determinata zona del file system del server, con una accurata gestione dei permessi in maniera tale che ogni utente possa vedere tutto e solo il proprio spazio di lavoro o quello che gli viene esplicitamente condiviso da altri utenti.

Naturalmente tutte le alternative riportate offrono il servizio di base e inoltre sono tutte open source, ciò che le distingue è legato principalmente a:

- user experience, sia per gli utenti che fruiscono il servizio ma anche per l'amministratore che installa l'applicazione
- possibilità di condividere file e cartelle tra utenti della piattaforma
- dispositivi in cui è disponibile l'applicazione client
- possibilità di integrazione di ulteriori features
- livello di sicurezza della soluzione
- funzionalità dell'edizione Enterprise

### *3.1.1 Sea File*

SeaFile è una piattaforma nata con l'obbiettivo di offrire la possibilità di sincronizzare file tra vari dispositivi. Sono disponibili due edizioni, una appunto open source e gratuita l'altra invece è una edizione Enterprise che aggiunge numerose funzionalità.

È possibile scaricare SeaFile tramite web installer, archivio compresso o immagine Docker. Come si vedrà successivamente, da una parte è la soluzione con meno possibilità di download ma allo stesso tempo come detto in precedenza, l'opzione fondamentale per il caso di studio è quella via Docker, che appunto è disponibile.

Dal punto di vista della collaborazione, è possibile lavorare sullo stesso file con più utenti contemporaneamente e la condivisione può avvenire sia con un link pubblico che con uno privato, protetto da password o a cui possono accedervi solo determinati utenti. Oltre a ciò, che comunque è una funzionalità relativamente di base, SeaFile offre la possibilità di creare una "library" che contiene file e cartelle che possono essere sincronizzate tra i vari dispositivi o condivise con altri utenti.

Per quanto riguarda la disponibilità in termini di applicazioni client, oltre alla versione browser, vi è il supporto per Windows, MacOS, Linux, Android ed iOS.

Le funzionalità offerte da SeaFile si limitano esclusivamente al cloud storage e sincronizzazione tra i vari utenti, non vi è quindi la possibilità di integrazione con servizi terzi o l'aggiunta di plugin per aggiungere features.

Il livello di sicurezza di SeaFile è nella media, offre infatti crittografia end-to-end per quanto riguarda la trasmissione ma anche conserva in maniera cifrata il contenuto dei file su disco. Per restare in tema sicurezza, supporta vari metodi di accesso che spaziano da LDAP a Kerberos.

La versione Enterprise di SeaFile offre numerose funzionalità extra come, ad esempio, la gestione degli account role-based ma anche funzioni che migliorano l'esperienza dell'utente come la ricerca all'interno di un file. Aumenta il livello di sicurezza tramite un antivirus e un audit log per tenere traccia di tutti gli eventi che avvengono.

### *3.1.2 OwnCloud*

OwnCloud è nato come una alternativa gratuita per rimpiazzare i servizi di storage proprietari e a pagamento. Anche questa alternativa è caratterizzata da due edizioni, dove la versione Enterprise offre delle funzionalità extra. Le possibilità di installazione sono le stesse offerte da SeaFile a cui si aggiunge il rilascio direttamente in cloud. Anche i dispositivi su cui è disponibile l'applicazione client sono gli stessi rispetto a SeaFile. Per quanto riguarda la possibilità di condivisione, si aggiunge la "guest feature" che permette la creazione di account temporanei con i quali abilitare tutte le funzionalità di collaborazione.

A differenza di SeaFile, OwnCloud offre uno store in cui sono disponibili oltre 200 app che permettono di personalizzare lo spazio di lavoro aggiungendo ad esempio la gestione del calendario o la sincronizzazione dei contatti.

Dal punto di vista della sicurezza le funzionalità sono le stesse di Sea File che vengono estese dalla versione Enterprise che aggiunge un file firewall. Tramite la versione Enterprise è possibile accedere a tool come Collabora Online Office, effettuare un branding dell'applicazione ma anche avere la possibilità di accedere all'app tramite Single Sign-On, che, come si vedrà a breve, rappresenta una importante funzione.

### *3.1.3 NextCloud*

Nextcloud è nata a partire da un gruppo di sviluppatori che si sono distaccati da OwnCloud, per cui la soluzione può sembrare simile ma vi sono numerose funzionalità

extra che rendono tale alternativa una delle più solide e diffuse. In questo caso esiste un'unica versione gratuita ed opensource che include tutte le funzioni, a cui si aggiunge eventualmente un supporto tecnico per il contesto Enterprise.

Oltre a tutte le possibilità di download viste con Seafile ed OwnCloud, spesso sono disponibili delle NAS che dispongono di Nextcloud nativamente, per una soluzione plug-and-play. Le funzionalità di collaborazione sono quelle di base che riguardano quindi la creazione di cartelle/file condivisibili tramite link pubblici o protetti.

Nextcloud è disponibile nella sua applicazione client per tutti i dispositivi citati per le precedenti soluzioni, a cui si aggiunge anche Windows Mobile.

Per quanto riguarda invece la possibilità di integrazione, anche in questo caso è disponibile un app store che conta oltre 120 app, ma tra tutte, sono degne di nota "Nextcloud Talk" e "Nextcloud group". Con la prima è possibile avere un sistema di chat tra gli utenti e fare delle vere e proprie video call, mentre la seconda gestisce una casella di posta, calendario e contatti. Queste funzioni rendo Nextcloud un competitor gratuito ed open source a piattaforme come Google drive ed Hangouts o Microsoft teams.

Le funzionalità di sicurezza offerte sono le stesse di OwnCloud ma appunto vi è il vantaggio di avere tutto nella versione gratuita, tra cui appunto la possibilità di integrare Single Sign-On.

| | SeaFile | OwnCloud | NextCloud |
|---|---|---|---|
| **Installazione** | Web Installer, archivio compresso, immagine Docker | Web Installer, archivio compresso, immagine Docker, Cloud | Web Installer, archivio compresso, immagine Docker, Cloud, dispositivi plug-and-play |
| **Condivisione** | Condivisione documenti tramite link pubblico/privato | Condivisione documenti tramite link pubblico/privato, Guest Feature | Condivisione documenti tramite link pubblico/privato |
| **App Client** | Windows, MacOs, Linux, Android, iOS | Windows, MacOs, Linux, Android, iOS | Windows, MacOs, Linux, Android, iOS, Windows Mobile |
| **App Store** | N/A | Store con oltre 200 app | Store con oltre 200 tra cui "Talk" |
| **Sicurezza** | Crittografia e2e, Cifratura file | Crittografia e2e, Cifratura file | Crittografia e2e, Cifratura file |
| **Enterprise** | Account role-based, ricerca file, antivirus, log | Collabora online, branding, SSO, file firewall | Solo supporto tecnico |

*Tabella 1: Confronto tra le varie alternative per Cloud Storage*

## *3.2 Database*

Il secondo requisito riguarda la possibilità di poter salvare, oltre a file di grandi dimensioni, anche dati provenienti da sensoristica, per cui è necessario progettare una soluzione che permetta di caricare i dati e visualizzarli. Inizialmente era stata valutata l'alternativa di tenere tale funzionalità all'interno dell'applicazione React che fa da Hub, successivamente si è notato che per poter permettere di ottenere una soluzione completa di numerose tipologie di grafici e visualizzazioni potesse tornare utile appoggiarsi anche in questo caso a servizi disponibili tramite immagini Docker.

Resta comunque necessario disporre di un database che faccia da backend, di cui a breve vi sarà una analisi delle principali alternative. Prima di ciò, bisogna scegliere come permettere ai sensori di collegarsi al database per cui vi sono due alternative:

- Creazione di una API: tramite questa alternativa è possibile interagire con il database tramite richieste http o https che vengono gestite da un web server, limitando la visibilità verso il database alle sole operazioni gestite dal server stesso
- Interazione diretta con DB: questa scelta invece permette di collegarsi direttamente al database su cui effettuare login e inviare i dati nella massima libertà.

Trattandosi comunque di un accesso che sarà sempre amministrativo, si conclude che può essere più efficacie non restringere le funzionalità del DB ma permettere un accesso diretto tramite una autenticazione e autorizzazione gestita direttamente a livello di DBMS, almeno per quanto riguarda il caricamento dei dati. È possibile in generale che i sensori possano inviare richieste HTTP ma sicuramente è più complesso per questi implementare HTTPS o accedere ad una remote site VPN, anche per questo emerge che l'accesso al Database sia la soluzione più versatile.

A questo punto si fa necessario selezionare il miglior DBMS per il caso di studio, per cui in seguito vi è l'analisi delle principali alternative sul mercato, sempre disponibili tramite Docker ed open source.

### *3.2.1 SQLite*

Citato solo per completezza, SQLite è un in-file DBMS relazionale. La sua prima versione è stata rilasciata nel 2000 da Dwayne Richard Hipp ed è stata pensata per essere un'ottima soluzione per ambienti di sviluppo o test. Tutto il contenuto del database viene salvato su un file per cui rappresenta una alternativa server-less.

Seppur sia supportato da tantissimi linguaggi di programmazione, implementa solo le funzionalità di base del database relazionale, per cui non ha nessuna forma di replicazione dei dati e non effettua partizionamento, inoltre non gestisce le politiche di accesso, il che lo rende una soluzione non adeguata al caso di studio.

In generale SQLite è la perfetta scelta per effettuare delle prime configurazioni per determinati progetti, nel ranking di popolarità dei DBMS è in top 10 per utilizzo

### *3.2.2 MySQL*

MySQL è tra le alternative per DBMS relazionali più diffusa, seconda per ranking dell'engine. Permette l'uso anche di altri modelli di database come quello basato su documenti o spaziale. La sua prima versione è stata rilasciata da Oracle nel 1995 e, parlando di sistemi cloud, è disponibile da vari provider nella modalità DBaaA (Data Base as a Service).

Implementa un partizionamento orizzontale tramite sharding (ossia la divisione dei dati in frammenti chiamati shards che potrebbero essere distribuiti su vari nodi per aumentare l'affidabilità del sistema).

In questo caso inoltre è implementata la replicazione tramite Multi-source o Source-replica replication.

A differenza di SQLite, sono disponibili in questo caso anche la possibilità di implementare degli script server-side, anche detti Stored-Procedure, tramite una sintassi dedicata.

Infine, su MySQL sono gestite le access control policy tramite un sistema di creazione utenti e assegnazione di privilegi.

### *3.2.3 MariaDB*

Come Nextcloud per OwnCloud, anche MariaDB è un progetto che ha preso avvio da un gruppo di sviluppatori di MySQL. In generale MariaDB è perfettamente compatibile con MySQL per cui è semplice effettuare migrazioni da un sistema all'altro.

MariaDB nasce quindi nel 2009 ed è sviluppato da MariaDB Corporation e MariaDB Foundation.

Anche in questo caso, vi sono dei fornitori di DBaaS e il Database è supportato da quasi tutti i sistemi operativi. Le funzionalità sono praticamente analoghe a MySQL, soprattutto per quanto riguarda il Database relazionale, la principale differenza potrebbe essere relativa alle stored-procedure che in questo caso sono compatibili con la sintassi del PL/SQL

In generale, si può interpretare MariaDB come una versione aggiornata di MySQL con retrocompatibilità.

| | SQLite | MySQL | MariaDB |
|---|---|---|---|
| **Rank** | 2 | 10 | 13 |
| **Modelli** | Relazionale | Relazionale, Document Store, Spatial | Relazionale, Document Store, Spatial, Graph |
| **Primo rilascio** | 2009, Dwayne Richard Hipp | 1995, Oracle | 2000, MariaDB Foundation e MariaDB Corporation |
| **Partizionamento** | N/A | Sharding | Shahrding |
| **Replicazione** | N/A | Multi-source, Source-replica | Multi-source, Source-replica |
| **Stored Procedure** | N/A | Si, proprietario | Si, PL/SQL |
| **Controllo accessi** | N/A | Si | Si |

*Tabella 2: Confronto tra le varie alternative per Database*

### *3.2.4 Data Analysis*

Questo paragrafo in qualche modo vuole sintetizzare la scelta del servizio di front end per la visualizzazione dei dati. Un'alternativa molto famosa su cui ispirarsi è AirTable, appunto però questo servizio non è disponibile per applicazioni selfhosted; tuttavia, al suo interno permette di effettuare grafici a partire da tabelle tramite l'inserimento di codice SQL o la compilazione di menu a tendina secondo l'approccio no-code. Le principali alternative a questo punto che prevedono la variante selfhosted sono:

- Nocodb: interessante soluzione che permette login tramite SSO, tuttavia è un progetto molto recente che al momento supporta un numero limitato di funzioni e grafici
- Baserow: esclusa a priori perché non disponibile per architetture ARM, tuttavia rappresenta un'ottima alternativa selfhosted ad AirTable, replicando molte funzionalità e visualizzazioni
- Grafana: alternativa disponibile per l'architettura in esame, implementa numerosi grafici e la possibilità di effettuare sia query no-code che query SQL per reperire i dati. Permette la creazione di grafici real-time che aggiornano i loro valori nell'intervallo stabilito. È possibile, inoltre, creare dashboard condivisibili privatamente o pubblicamente e per gli accessi, vi è la possibilità di integrare il SSO. Infine, è disponibile un community appstore che permette di scaricare nuove tipologie di visualizzazione o crearne di nuovi tramite script.

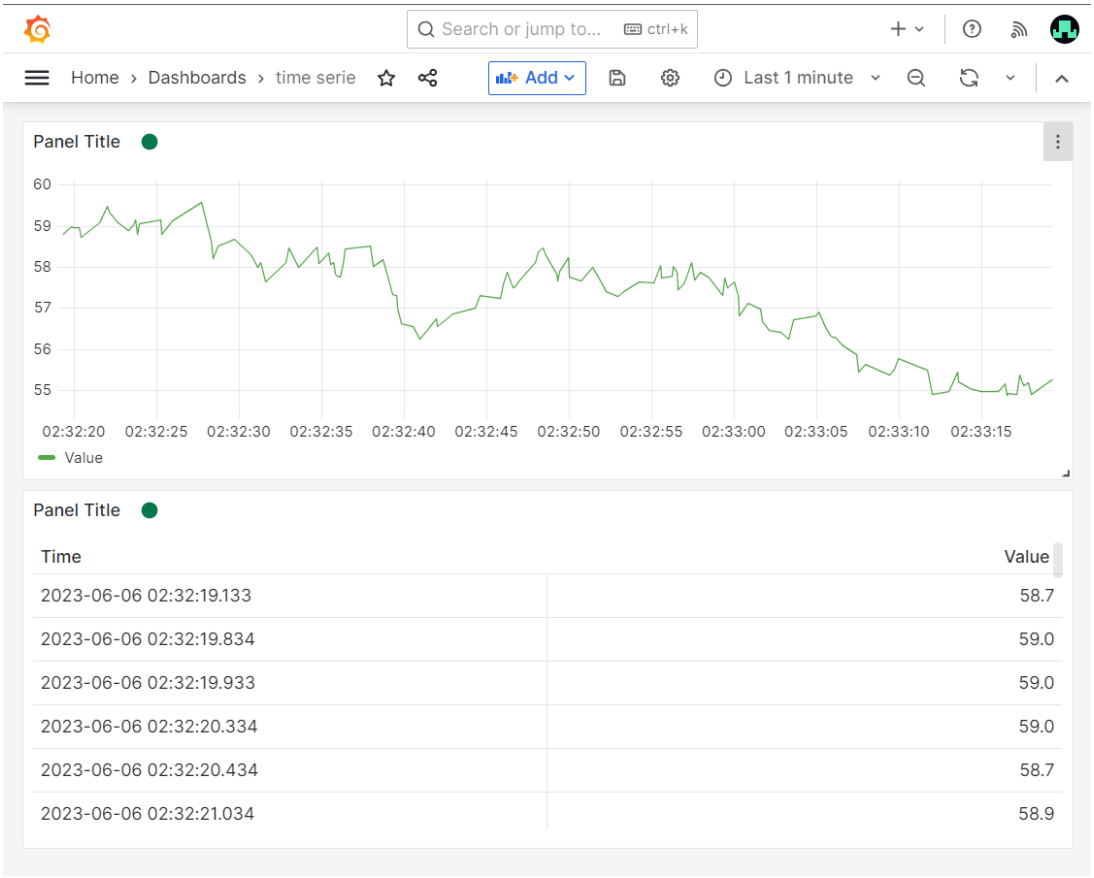


*Figura 12: Esempio di dashboard realizzata con Grafana con dati aggiornati in tempo reale*

## *3.3 Version Control*

Nella gestione del codice, ormai è divenuto un must disporre di un sistema di version control. L'idea dietro tali sistemi è quella di permettere la creazione di un albero che tenga in considerazione i vari aggiornamenti del codice. In linea di principio quindi, ogni volta che si eseguono delle modifiche al codice, viene creato un nuovo nodo all'interno dell'albero che segna un nuovo stato del progetto. Lo strumento utilizzato per tale scopo è GIT, che tramite l'operazione di commit crea il nuovo nodo.

La gestione dell'albero è molto ottimizzata, infatti vengono conservate le modifiche da un nodo ad un altro in maniera differenziale, per cui non tutti i nodi contengono tutto il codice del progetto, ma gran parte contengono solo le differenze con il precedente.

Altro aspetto fondamentale di GIT e del version control in generale è la possibilità di creare dei branch, per cui, in base alle scelte di design degli sviluppatori, tendenzialmente in ogni branch viene sviluppata in maniera indipendente una nuova funzionalità. La creazione del branch consiste dunque nella copia dell'intero codice da cui viene staccato e un nuovo flusso di modifiche che seguono a partire da questo punto. Successivamente è possibile effettuare l'operazione di merge per fondere il branch nel ramo di partenza.

Una delle più grandi potenzialità del sistema GIT sta proprio nella sua funzionalità di collaborazione, tramite la sincronizzazione di commit e branch in un server remoto, grazie al quale i vari utenti possono lavorare in contemporanea tendenzialmente su branch differenti. Per abilitare tale funzionalità è quindi necessario che vi sia un server GIT remoto, inoltre, per comodità degli utenti, è necessario un sistema user-friendly per visualizzare lo stato del codice e possibilmente abilitare la possibilità di funzioni extra come l'approvazione del codice da mergiare nei rami più importanti, che va sotto il nome, in base ai provider, di Pull/Merge Request.

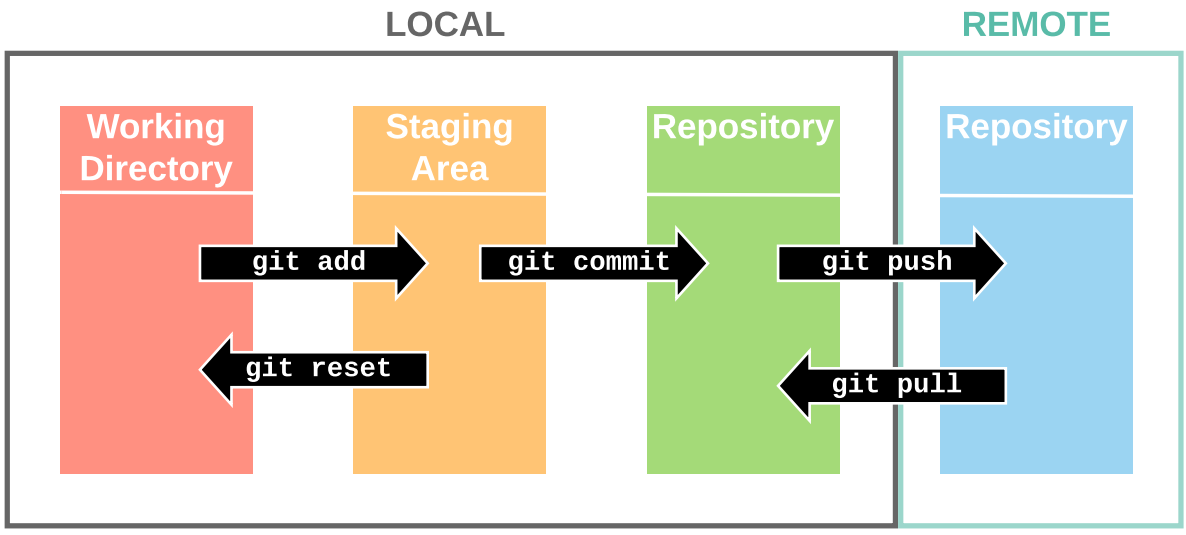


*Figura 13: Grafico sulle principali funzionalità di GIT, inclusa la gestione di un repository remoto*

Nei paragrafi successivi saranno quindi analizzati i vari tool disponibili per creare una web UI per GIT, che includono anche l'installazione del rispettivo GIT server.

### *3.3.1 GitHub*

GitHub rappresenta il servizio di GIT server ed UI più famoso. Permette l'accesso a numerose funzionalità che, oltre a quelle di base citate nel paragrafo introduttivo, spaziano dalla creazione di catene di CI/CD (Continous Integration Continous Deployment) fino all'offerta di servizi educativi come webinars.

Trattandosi del servizio più diffuso, praticamente è compatibile con molti altri servizi, offrendo numerose integrazioni, tutta via, non si tratta di un prodotto opensource per cui è complicato creare delle estensioni personalizzate.

GitHub ad ogni modo non offre la possibilità di creare una istanza self hosted, permetterebbe infatti di usare il tool "Self-host runner" dove eseguire GitHub action in un ambiente privato, ma comunque restando connesso e condividendo i dati con il server cloud pubblico centrale. Per questa ragione, viene citato solo come principale esempio di servizio di UI in quanto il più famoso ma naturalmente non rappresenta una valida alternativa per il caso di studio in analisi, almeno nella sua versione di base.

### *3.3.2 GitLab*

GitLab è un'ottima alternativa a GitHub che però offre la sua variante self hosted diventando quindi un buon compromesso per il progetto descritto da questa tesi. Come GitHub offre, oltre alle funzionalità di base per un GIT server, numerosi servizi aggiuntivi ed integrazioni con altre applicazioni come ad esempio Microsoft Teams.

Il sistema in generale è scalabile, infatti supponendo un sistema con 64 core di CPU e 64 GB di RAM sarà possibile sopportare il carico di circa 40 000 utenti, per cui il prodotto riesce ad essere competitivo in contesti Enterprise.

GitLab è disponibile in due versioni, la Community Edition, gratis ed opensource, e la Enterprise Edition, acquistabile tramite abbonamento e ricca di funzionalità extra, come ad esempio la ricerca globale nel codice o la disponibilità di repository template.

In generale quindi, GitLab è una soluzione per contesti medio grandi o per aziende che puntano a scalare in futuro.

### *3.3.3 Gitea*

Probabilmente Gitea rappresenta l'alternativa meno famosa rispetto alle precedenti citate ma ha delle caratteristiche degne di nota, soprattutto ai fini del caso di studio. Tali caratteristiche vengono spesso riassunte con i termini "lightweight" ed "easy-to-use". Questi aspetti mettono dunque in risalto il fatto che Gitea si occupa quasi esclusivamente della gestione del GIT server, a cui però si aggiungono funzionalità collaborative, come code review e gestione degli issues.

Tramite Gitea è anche possibile creare delle catene di CI/CD e delle wiki per documentare il codice, oltre che svariate integrazioni con altri servizi come, ad esempio, la possibilità di SSO tramite Oauth2 ed OIDC.

Anche Gitea riesce a scalare bene fino a contesti di medie dimensioni, tuttavia il servizio è ideato per contesti più piccoli ma soprattutto, offrendo un numero di servizi ristretto all'essenziale, è possibile avere eccellenti performance anche rilasciandolo su server meno potenti.

La caratteristica dell'easy-to-use si nota soprattutto guardando l'interfaccia grafica dell'applicazione, il suo aspetto minimalistico permette agli utenti di individuare a colpo

d'occhio la funzionalità necessaria, senza sommergerli con numerose opzioni ed impostazioni.

A differenza dei servizi citati in precedenza, Gitea è completamente gratuito ed opensource, è possibile, a discrezione dell'utente, effettuare donazioni alla società sviluppatrice. Nonostante ciò, cosa non scontata generalmente per alternative completamente gratuite, è disponibile un supporto per integrare Gitea a varie piattaforme di sviluppo, come VSCode tramite dei plugin sviluppati e manutenuti dal gruppo di awesome-gitea.

| | GitHub | GitLab | Gitea |
|---|---|---|---|
| **Selfhosted** | No, solo actions o Enterprise | Si | Si |
| **OpenSource** | No | Si | Si |
| **Integrazione** | Compatibile con numerosi servizi | Compatibile con numerosi servizi | Compatibile con alcuni servizi |
| **Enterprise edition** | Si, permette di avere selfhosted | Si, ricerca codice e repository templati | N/A |
| **Punti forti** | Famoso, numerose integrazioni ed automatizzazioni | Altamente scalabile a contesti grandi | Lightweight e easy-to-use |

*Tabella 3: Confronto tra le varie alternative per Version Control*

## *3.4 Conclusioni*

Tenendo a mente il caso di studio e l'hardware a disposizione, uno degli aspetti centrali riguarda proprio il cloud storage, sul quale appunto si vuole avere un buon compromesso di performance e funzionalità. L'alternativa selezionata è quindi quella offerta da Nextcloud, vista la sua natura opensource ma anche e soprattutto la presenza di un app store molto variegato con cui estendere le funzioni dell'intero sistema senza aggiungere ulteriori servizi. Si supponga ad esempio di voler gestire anche un mail user agent per poter accedere alle proprie caselle di posta, basterebbe aggiungere tramite utenza amministrativa l'applicazione Mail, con cui sarà possibile sincronizzare e mandare nuove mail.

Dal punto di vista del database, naturalmente è esclusa l'alternativa di SQLite in quanto pensata solo per ambienti di sviluppo. La scelta tra MariaDB e MySQL nel pratico ha un impatto minore, per cui è stato selezionato MariaDB in quanto più moderno. Mentre per quanto riguarda la data analysis, Grafana rappresenta la migliore scelta, grazie alla sua leggerezza nella configurazione di base ma comunque la possibilità di essere estesa tramite l'app store o plugin custom, inoltre una caratteristica fondamentale è quella di lasciare libero l'utente di utilizzare sia query no-code che vere e proprie query SQL per estrarre i dati.

Infine, per il version control torna utile avere una soluzione più leggera e gratuita per cui Gitea copre il ruolo in maniera più che adeguata, riuscendo a gestire tutte le funzionalità di base di GIT aggiungendo lo stretto indispensabile in termini di collaborazione.

Tutti i servizi riportati gestiscono una propria sessione utente, ma per potervi accedere, sarebbe necessario creare 3 utenze, con eventualmente 3 password differenti.

Questo aspetto può ledere l'esperienza stessa dell'utente, per cui un aspetto sottolineato fino ad ora nella scelta dei servizi sta nella possibilità di integrazione con servizi di Single Sign-On, che sarà uno degli argomenti del prossimo capitolo, dedicato al networking e alla sicurezza dell'intero sistema.

Infine, sarà resa disponibile una applicazione React che faccia da Hub per accedere ai vari servizi, oltre che abilitare l'accesso agli Admin tools descritti successivamente.


*Figura 14: Homepage dell'applicazione React "Hub"*

In questo capitolo si vuole cercare di effettuare una descrizione TOP-DOWN dell'architettura di rete utilizzata per il caso di studio, offrendo dei suggerimenti per alternative che possano adattarsi a contesti differenti.

Nel volere istanziare qualunque forma di web server, bisogna avere una gestione ben precisa della rete, specialmente considerando che, nel caso di studio, il server verrà ospitato all'interno di una piccola rete aziendale privata sotto il servizio di Network Address Translation (NAT).

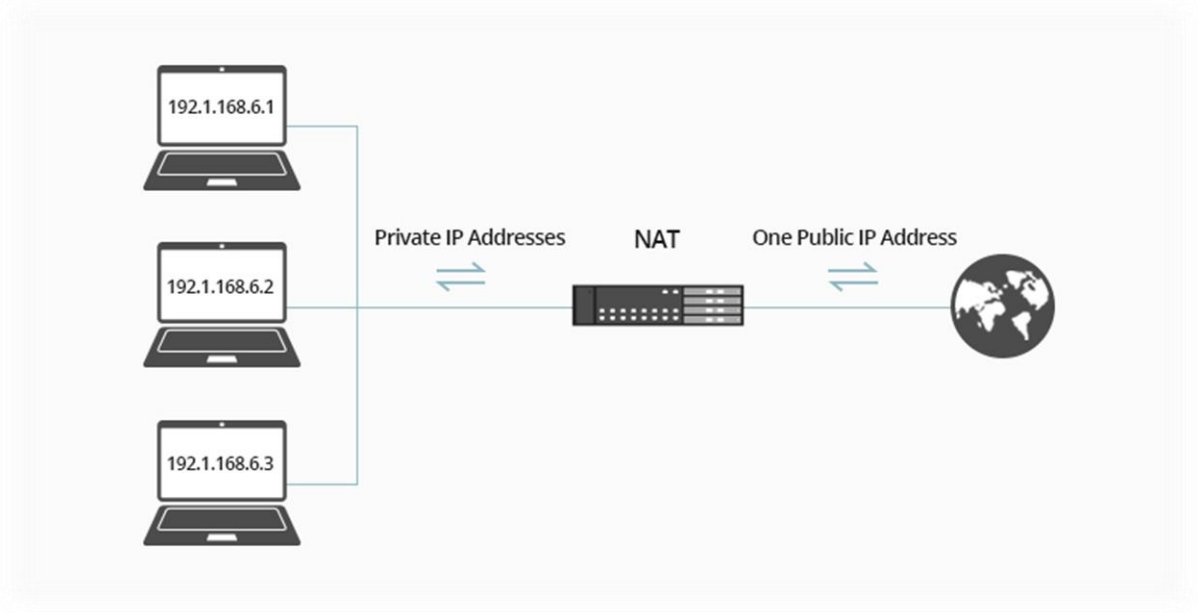


*Figura 15: Schema funzionamento NAT*

È dunque necessario avere una grande cura delle politiche di accesso perché la rete privata sarà esposta verso internet. Il router installato nella propria abitazione, generalmente, dispone di varie interfacce verso la rete privata, caratterizzate da IP in specifici range dedicati (es: 192.168.0.0/24) che non vengono mai esposti pubblicamente, e una interfaccia pubblica verso la rete con un IP, appunto, pubblico.

L'IP pubblico ha l'obbiettivo di creare un indirizzamento efficiente dei pacchetti di rete, per cui spesso i provider possono decidere di cambiare, secondo specifici algoritmi, gli IP assegnati ai vari clienti del loro network, in questo caso si parla infatti di IP dinamico. Alcuni provider offrono la possibilità di avere un IP statico, tramite il pagamento di un contributo extra nella bolletta.

Discutendo sempre dell'organizzazione dei provider, spesso questi sfruttano sistemi chiamati Carrier Grade NAT, questo gli permette di creare delle grandi reti private per cui ad ogni cliente viene assegnato un IP privato (anche nell'interfaccia pubblica). Il funzionamento di tale architettura si basa, come il NAT, sul principio che gli utenti del servizio di rete siano, a livello logico, dei client, piuttosto che server. Nell'architettura client server, la principale differenza logica tra le entità sta nel fatto che le interazioni iniziano sempre con una richiesta del client verso il server. I router di frontiera della rete dell'ISP

verso la rete pubblica si occupano quindi di salvare delle informazioni del pacchetto in uscita così da collegare poi la risposta alla richiesta (l'informazione salvata è generalmente la porta sorgente della comunicazione, che viene mappata su un'altra porta che limiti il numero di conflitti).

La presenza di CGNAT diventa quindi estremamente limitante per soluzioni di cloud privato e in generale rende impossibile l'istallazione di web server. Per contesti del genere diventa quindi obbligatorio richiedere un IP statico all'ISP per poter implementare tali sistemi ed accedervi dall'esterno.

In assenza di CGNAT però è possibile ricorrere a soluzioni differenti rispetto all'uso di un IP statico per poter istallare il sistema. Riflettendo sull'esigenza dietro alla richiesta di un IP statico, tutto nasce dall'obbiettivo di avere una maniera per raggiungere da qualunque parte di internet un determinato punto. In analogia con il servizio postale, per poter spedire una lettera è necessario sapere l'indirizzo del destinatario. Il destinatario saprà poi l'indirizzo del mittente per rispondere, in quanto contenuto nella lettera spedita in precedenza.

Forzando un po' l'esempio, si supponga che per qualche motivo cambi il nome della via del destinatario o il numero civico, il mittente non avrebbe più modo di contattarlo. Questo è dunque il problema relativo all'uso di IP dinamico. Ad ogni modo, si potrebbe sfruttare un indirizzamento differente, sfruttando, tornando al networking, il Domain Name System (DNS), e quindi, invece di basarsi su IP, si usano dei nomi di rete, in generale più facili da ricordare.

Il sistema di DNS è aggregativo e gerarchico, inoltre, per garantire elevati livelli di affidabilità, è anche ridondante. Il sistema può essere inteso come un grande Database che conserva il mapping tra IP pubblici e nomi. Se un client vuole accedere ad un determinato server, non deve ricordarne l'IP ma solo il nome, ad esempio, quando si digita www.google.com nella barra di ricerca di un browser, questo prima di tutto fa una richiesta al server DNS (il cui IP generalmente è impostato staticamente tramite DHCP), questi risponde con l'IP di Google, che al momento della stesura di questa tesi è 172.217.14.68. È possibile effettuare un test anche da linea di comando digitando 'nslookup www.google.com'.

Il DNS resta comunque disponibile per IP statici, ma la sua variante dinamica, il Dynamic DNS (DDNS), permette di mantenere aggiornata la entry relativa ad un determinato nome di rete anche al variare dell'IP. Per poter configurare il DDNS è quindi necessario sfruttare un provider, come ad esempio noip.com, sul quale selezionare il nome di rete ed effettuare il primo collegamento inserendo l'IP dinamico attuale. Successivamente si può automatizzare la procedura di rinnovo dell'IP facendo in modo che periodicamente un host della rete privata mandi una richiesta al server di noip.

È possibile istallare un programma nel server che si occupi di tale operazione, ma spesso i router offrono questo servizio e hanno delle configurazioni preimpostate per DDNS provider famosi, come noip. Naturalmente questa soluzione ha il difetto che vi potrebbero essere periodi di non raggiungibilità pari al più alla dimensione temporale dell'intervallo di aggiornamento, ma, nonostante siano dinamici, gli IP vengono aggiornati molto raramente, perché, a seguito di giri di assegnazione, viene spesso riassegnato lo stesso IP.

A questo punto, la rete privata è raggiungibile tramite la entry DDNS o IP statico, per cui si può passare ad una configurazione ad un livello inferiore nell'approccio TOP-DOWN, che vede come poi dal router si raggiunga il web server, rilasciato su Raspberry PI.

## *4.1 Protocollo VPN*

Quando arriva una richiesta al router, questi deve essere in grado di inoltrarla al dispositivo corretto, ciò è possibile grazie alla funzionalità di port mapping/forwarding, implementata in quasi tutti i router. L'idea è di permettere ad un pacchetto con destinazione pari all'IP pubblico del router, di essere in grado di capire verso quale dispositivo privato questi debba essere inoltrato basandosi sulla porta di destinazione del pacchetto stesso.

Una prima alternativa sarebbe quindi quella di permettere il collegamento diretto dall'esterno verso il Raspberry PI per i pacchetti aventi porta 443, ma appunto questo permetterebbe a chiunque di mandare dati verso la rete privata. Una soluzione più raffinata e sicura è quella di affidarsi ad una VPN.

L'utilizzo di una VPN permetterebbe inoltre di integrare alcune funzionalità di DHCP come l'utilizzo di uno specifico DNS, l'assegnazione di IP e gateway ma anche la diffusione di autorità di certificazione fidate, nel caso sia utilizzata una gerarchia di certificati dove l'autorità root sia selfsigned e non appartenente alle principali autorità di certificazione fidate per default dai dispositivi.

La VPN creerà quindi un tunnel da ogni dispositivo (end) fino al Raspberry Pi (site), instaurando dunque una "remote access VPN". Viene selezionato inoltre un protocollo a livello applicativo, che quindi non richiede modifiche a livello kernel, come anticipato al paragrafo 1.2.1. Si esaminano quindi due delle alternative più famose e diffuse per tale infrastruttura.

### *4.1.1 OpenVPN*

OpenVPN rappresenta l'alternativa più famosa per instaurare delle VPN a livello applicativo. Si rischia di fare confusione quando si utilizza il termine "OpenVPN", in quanto fa riferimento al nome dell'azienda che l'ha sviluppato, al nome del protocollo stesso e al nome del client/server proprietario dell'azienda. Seppur il protocollo sia completamente opensource (con un codice di circa 70000 linee), l'applicazione OpenVPN Access Server presenta un piano base che permette al più due connessioni simultanee o piani a pagamento mensili che ne permettono un numero maggiore. È comunque possibile utilizzare applicazioni differenti per il server come descritto nei paragrafi successivi, dove appunto viene utilizzato il protocollo OpenVPN. Per quanto riguarda il client è comunque possibile usare quello proprietario di OpenVPN disponibile gratuitamente.

OpenVPN permette di creare tunnel sia su TCP che su UDP, nel primo caso si ha una connessione più stabile ma relativamente lenta (intorno ai 10Mbps), mentre nel secondo si possono ottenere connessioni più rapide (intorno ai 130Mbps). I numeri riportati sono in un contesto in cui la baseline è di 300Mbps, per cui si può notare una forte riduzione della velocità.

A livello di sicurezza, il protocollo gestisce numerose cipher suite, che spaziano dal supporto per AES fino a ChaCha20, includendo anche alternative Authenticated Encryption come GCM, ad esempio AES-256-GCM (riconosciuto come il livello di sicurezza più alto). Dal punto di vista dell'autenticazione, supporta la funzione di hash Poly1305, anch'essa riconosciuta come la migliore funzione di hash per sicurezza.

L'anzianità del protocollo, rilasciato per la prima volta nel 2002, in qualche modo contribuisce alla sua compatibilità; infatti, praticamente tutti i sistemi operativi sono in grado di collegarsi ad OpenVPN o di ospitare un server OpenVPN.

### *4.1.2 WireGuard*

WireGuard è un recente protocollo applicativo, rilasciato nel 2015, sviluppato per sfruttare al massimo le potenzialità dell'hardware moderno. Il suo codice è molto più corto rispetto ad OpenVPN, infatti conta circa 4000 linee, ciò gli permette di essere più facile da analizzare e, avendo livelli di complessità minore, più sicuro in termini di bug e vulnerabilità.

WireGuard permette la creazione di tunnel esclusivamente via UDP, che ad oggi, viste la grande affidabilità dell'infrastruttura di rete, si ritiene sufficiente, raggiungendo velocità più elevate di OpenVPN, con uno elevato speedup al punto di raggiungere anche 280 Mbps. Con questo risultato, applicato sempre ad una baseline di 300Mbps, si può notare come si raggiunga quasi il livello di trasparenza nell'uso di questo protocollo VPN.

Dal punto di vista della sicurezza, WireGuard è più limitata, nel senso che implementa solo ChaCha20 per la cifratura e Poly1305. La problematica è dunque legata al fatto che non propone diverse alternative, che un utente potrebbe selezionare per adattarsi a contesti in cui le potenzialità hardware sono minori e il sistema risulterebbe più lento, tuttavia, data l'elevata velocità del protocollo, si può tranquillamente usare il livello massimo di sicurezza senza risentire di eccessivi cali di prestazioni. ChaCha20 in generale ha un livello di sicurezza e performance molto simile ad AES-256-GCM.

Il difetto principale di WireGuard è probabilmente legato alla limitata compatibilità, per quanto riguarda applicazioni per il server, infatti, è stato inizialmente progettato per Linux, riuscendo a sfruttare al massimo le potenzialità del kernel. Probabilmente la minore compatibilità è legata al fatto che comunque WireGuard è un protocollo giovane e ancora non è diffuso agli stessi livelli di OpenVPN, ma sta crescendo rapidamente grazie alla sua velocità e semplicità del codice, che ne rappresentano i punti di forza.

| | OpenVPN | WireGuard |
|---|---|---|
| **Primo rilascio** | 2002, James Yonan | 2015, Jason A. Donenfeld |
| **Lunghezza codice** | 70.000 linee | 4.000 linee |
| **Tunnel supportati** | TCP, UDP | UDP |
| **Velocità (baseline: 300 Mbps)** | 10 Mbps (TCP),<br>130 Mbps (UDP) | 280 Mbps (UDP) |
| **Cifratura** | AES, Blowfish, Camellia, ChaCha20, … | ChaCha20 |
| **Hash** | Poly1305 | Poly1305 |
| **Compatibilità** | Tutti i sistemi operativi | Ottimizzato per Linux |
| **Punto di forza** | Compatibilità | Velocità e semplicità |

*Tabella 4: Confronto tra le varie alternative per protocollo VPN*

## *4.2 Applicazione VPN*

Dal punto di vista pratico, è necessario selezionare quale applicazione sarà incaricata di creare l'endpoint lato server del tunnel VPN. Una caratteristica di questa componente è che, a differenza di tutte le altre componenti software, non verrà istallata tramite docker-

compose. Il motivo di questa scelta è legato a questioni di networking, in quanto, come descritto a breve, docker-compose crea una rete virtuale, a cui non si deve avere accesso libero dall'esterno. Se il tunnel venisse creato all'interno di tale rete, un utente potrebbe accedere a tutti i servizi liberamente, evitando le infrastrutture di login o accedendo direttamente ai database interni. Per questo, qualunque applicazione di server VPN sarà installata direttamente sul PI.

### *4.2.1 OpenVPN Access Server*

OpenVPN AS, come citato in precedenza, è l'alternativa per la gestione del server offerta direttamente da OpenVPN. Nella sua versione gratuita permette il collegamento di al più due utenti simultanei, che potrebbe essere troppo limitante dal punto di vista del caso di studio. Naturalmente l'applicazione gestisce esclusivamente il proprio protocollo, non ha quindi la possibilità di supportare WireGuard.

Il suo punto forte al solito è legato alla compatibilità con ogni sistema operativo, per il resto, rappresenta un'ottima alternativa che copre tutte le necessità di base per utilizzare il protocollo di OpenVPN.

### *4.2.2 SoftetherVPN*

Softether è una soluzione prodotta in contesto accademico nel 2014, dall'università di Tsukuba. L'obbiettivo di tale applicazione è quello di supportare numerosi protocolli di VPN, tra cui OpenVPN ma non Wireguard, offrendo una compatibilità su molti sistemi operativi. Per quanto riguarda la gestione del protocollo OpenVPN, i risultati sono abbastanza simili ad OpenVPN AS, coprendo tutte le funzionalità di base e raggiungendo gli stessi picchi di velocità, il tutto però in una soluzione completamente gratuita ed opensource, con un codice che, per coprire gli ulteriori protocolli VPN, conta quasi 400 000 linee.

Un'importante caratteristica di Softether dal punto di vista della gestione amministrativa è legata alla disponibilità di un server manager che permetta, tramite GUI, la creazione di nuovi tunnel, la gestione degli utenti e la configurazione dei vari protocolli. Tutto ciò è accessibile anche via rete su un dispositivo differente da quello in cui è rilasciato, che torna molto utile dato che, per migliorare le prestazioni, sul PI viene installato un sistema operativo senza interfaccia grafica ma sola command line.

### *4.2.3 PIVPN*

PIVPN è un tool leggerissimo e pensato proprio per essere installato su Raspberry PI. Non dispone di interfaccia grafica e supporta sia OpenVPN che Wireguard. Tramite una procedura guidata avviabile tramite uno script rilasciato dagli stessi sviluppatori, è possibile configurare il tool, in termini di protocollo usato, indirizzo dell'host (IP o nome DNS) e server DNS da far utilizzare agli host.

Terminata tale procedura, è possibile creare nuovi utenti, per i quali sarà creato un file di configurazione dedicato che potrà essere esportato nei relativi client o stampato a schermo tramite un QR code direttamente scansionabile dalla linea di comando (viene costruito con dei caratteri speciali, per cui non è una vera e propria immagine ma viene comunque riconosciuto perfettamente dagli scanner).

## *4.3 Reverse Proxy*

Arrivati a questo step, il server è raggiungibile sia dalla rete privata, tramite IP o DNS configurato staticamente ma anche dall'esterno, tramite VPN in grado, eventualmente, di configurare un server DNS da fare usare agli host. È necessario dunque scendere ad un altro livello dell'analisi. Come trattato nei capitoli precedenti, non sarà usata una singola applicazione per erogare tutti i servizi ma applicazioni differenti per servizi differenti. Per poter dunque accedervi, si potrebbe operare a livello di codice utilizzando degli iframe, ma le applicazioni rischierebbero di non essere ottimizzate per tale scopo, portando a rallentamenti e bug, per cui la scelta migliore è quella di assegnare domini diversi ad applicazioni diverse. Nello specifico:

- Cloud storage: storage.master.project.rg (Nextcloud)
- Version control: vc.master.project.rg (Gitea)
- Data Analysis: data.master.project.rg (Grafana)

Tutti questi nomi di dominio comunque puntano allo stesso IP statico, per cui come potrebbe il PI capire verso quale servizio inoltrare una richiesta in arrivo?

Tra gli header non obbligatori di una richiesta http vi è l'header "Host", nel quale viene inserito il nome di dominio da cui è partita la richiesta. È possibile fare routing basato proprio su tale nome. Nel pratico, quando da browser viene effettuata una ricerca verso, ad esempio, https://storage.master.project.rg, prima di tutto viene fatta una query DNS per ottenere l'IP legato al nome di dominio, successivamente viene fatta la vera e propria richiesta verso l'IP in questione (ossia quello del PI) inserendo nel campo "host" storage.master.project.rg.

La richiesta a questo punto arriva fino al PI e viene smistata all'applicazione corretta grazie all'uso di un reverse proxy. Nell'infrastruttura di rete desiderata quindi il reverse proxy gioca un ruolo chiave, in quanto questo sarà l'unico servizio raggiungibile dall'esterno dalla porta 80 o 443 (HTTP o HTTPS), poi questi internamente inoltrerà agli host posizionati su una rete virtuale non accessibile se non tramite esso (oggetto del paragrafo dedicato al Docker Networking).

Per certi sensi quindi il reverse proxy esercita anche un ruolo di firewall, riuscendo ad aumentare i livelli di sicurezza dell'intero sistema. Il reverse proxy può anche coprire il ruolo centralizzato di layer TLS, incaricando questi della gestione della sicurezza di canale, dato che comunque rappresenta l'endpoint massimo raggiungibile dall'esterno. Ciò permetterebbe di non dover gestire il TLS singolarmente per ogni applicazione, portando a difficoltà di manutenzione. Ad esempio, il rinnovo del certificato avverrebbe solo per il reverse proxy e non per tutte le applicazioni.

In tali sistemi, sono generalmente disponibili altre funzionalità, come ad esempio load balancing o caching, inoltre, essendo un punto medio tra il client e il server, possono effettuare operazioni sui pacchetti in transito come l'aggiunta di header o la redirezione basata sulle URI.

Detto ciò, i prossimi paragrafi mirano a descrivere alcune delle più famose opzioni di reverse proxy disponibili tramite istallazione Docker.

### *4.3.1 Traefik*

Traefik è una soluzione opensource nata nel 2016 nel contesto del cloud-native application proxy. Offre anche un servizio di load balancing e nel suo complesso permette

il monitoraggio tramite una interfaccia grafica web. Questa alternativa è stata selezionati da vari big, come Mozilla, Expedia e Bose. Uno degli aspetti centrali di Trafik nella sua configurazione è la sua scoperta automatica dei servizi. È possibile creare un sistema di base semplicemente etichettando i servizi con delle label che ne specificano il valore dell'header host da utilizzare per il routing. Ad esempio, nella creazione del servizio per Nextcloud, basterebbe aggiungere nella sezione dedicata a tale applicazione la seguente label:

traefik.http.routers.apache.rule=Host(`storage.master.project.rg`)

Successivamente, avviando il sistema, sarà possibile accedere all'interfaccia web di Traefik e visualizzare la entry relativa a Nextcloud.

In generale Traefik è in grado di gestire traffico HTTP, TCP, UDP e tramite l'utilizzo di ulteriori labels è possibile aggiungere configurazioni, in un set che comunque resta generalmente limitato e poco documentato.

### *4.3.2 Nginx*

Nginx è una soluzione sempre opensource ma più vecchia. In realtà è nata con uno scopo differente, infatti Nginx è uno dei principali web server disponibili sul mercato (anche all'interno di questo progetto viene utilizzato un server Nginx per hostare l'applicazione Hub discussa in seguito). Grazie alla sua popolarità in tale settore, ha iniziato via via ad integrare sempre più funzioni di un reverse proxy, come load balancing, caching e gestione del TLS. Anch'esso è utilizzato da grandi aziende come Dropbox e Netflix e in generale vi sono oltre 350 milioni di siti che usano tale servizio di hosting.

A differenza di Traefik, dunque, non è una soluzione nata per il contesto cloud e non ha una interfaccia grafica. La configurazione è leggermente più complessa, in quanto non vi è la ricerca automatica offerta da Traefik ma al contrario bisogna scrivere complessi file di configurazione. Questo ultimo aspetto però ha un vantaggio non trascurabile, ossia la possibilità di poter adattarsi a contesti bare-metal tramite una grande possibilità di personalizzazione.

Giunti fin qui, il peso della difficoltà di configurazione e la mancanza di interfaccia grafica renderebbero tale scelta troppo limitante; tuttavia, è disponibile un servizio chiamato Nginx Proxy Manager che riesce a colmare completamente questo gap, offrendo una interfaccia web al pari di Traefik da cui è possibile effettuare tutta la configurazione tramite checkbox e form, aggiungendo anche uno spazio di configurazione custom per poter personalizzare al massimo e sfruttare tutte le potenzialità e flessibilità di Nginx.

| | Pro | Contro |
|---|---|---|
| **Traefik** | Nata per il contest cloud, scoperta automatica dei servizi, presenza di UI | Configurazione limitata alle labels supportate |
| **Nginx** | Più esperienza nel settore, maggiore personalizzazione, | Assenza di UI (ma c'è Nginx Proxy Manager), file di configurazione complessi |

*Tabella 5: Confronto tra le varie alternative per reverse proxy*

## *4.4 Docker networking*

L'ultimo livello per quanto riguarda il networking è proprio quello gestito da Docker stesso. Quando viene creato un progetto Docker-compose, è possibile specificare, oltre alla

sezione relativa a servizi e volumi, anche una sezione dedicata alle reti, in cui si dichiara una vera e propria infrastruttura di rete. Se non si dichiara nulla, tutti i servizi saranno all'interno di una rete non raggiungibile dall'esterno che prende il nome della cartella in cui è posizionato il file docker-compose.yml.

Sulla base dei nomi dati ai vari container, verrà creato automaticamente un sistema di DNS; tutti i servizi dentro tale rete sono raggiungibili dall'interno tramite il nome che vi si assegna. Per poter accedervi anche dall'esterno, è necessario sfruttare il meccanismo delle porte e selezionare quali porte sono accessibili dall'esterno, dichiarando eventualmente un mapping.

Si può immaginare che ogni container all'interno del docker-compose sia un host, munito di IP, che espone le proprie porte e che poi vi sia un router con servizio di NAT che fa da ponte tra la rete interna a Docker e la macchina host (e quindi la rete a cui questa è connessa).

Questo meccanismo permette dunque di utilizzare un reverse proxy installato che espone le proprie porte verso la macchina host, ma essendo comunque dentro la rete Docker, sarà in grado di accedere ai vari servizi e di inoltrarvi le richieste, anche sfruttando i nomi di dominio assegnati.

Se non si usasse tale sistema, in combinazione con il reverse proxy, bisognerebbe esporre ogni servizio verso l'esterno, che già di per se potrebbe portare a rischi di sicurezza, ma inoltre, dato che la porta 443 per HTTPS è solo una e non è condivisibile, bisognerebbe mappare servizi differenti su porte differenti, portando alla necessità per l'utente di specificarle quando digita l'URL. Ad esempio, se per Nextcloud la porta 443 fosse mappata sulla porta 444, l'utente per accedere al servizio dovrebbe digitare https://storage.master.project.rg:444, che in generale si vuole evitare, per garantire una migliore user experience.

## *4.5 SSO*

Questo paragrafo e i successivi si pongono a metà, per argomento, tra il capitolo 3 e 4. Bisogna infatti prevedere una infrastruttura per permettere l'accesso ai vari servizi, creando e gestendo sessione login, argomenti che dunque rientrano sia nel software da prendere in considerazione ma anche e soprattutto nella parte di networking dedicata alla sicurezza. L'autenticazione nei sistemi attuali gioca un ruolo fondamentale, in quanto si vuole sempre garantire l'identità di ogni utente, e questo va in coppia con il concetto di autorizzazione, secondo il quale ogni utente deve essere in grado di effettuare tutte e sole le operazioni a cui è autorizzato.

La discussione sui metodi di autenticazione e autorizzazione potrebbe essere sviluppata in una propria tesi, in questo contesto ci si limita a mettere in risalto alcuni concetti fondamentali. Di base, i vari servizi selezionati gestiscono il proprio login, ognuno con le proprie credenziali, questo porta a problemi umani della gestione delle password, come ad esempio dimenticarle o annotarle in luoghi non sicuri. In generale, un aspetto fondamentale dall'autenticazione è quello di ridurre al minimo l'utilizzo di password e per rispondere a tale esigenza è stato sviluppato il concetto di Single Sign-On (SSO).

L'idea di base è quella di far sì che vi sia un servizio centralizzato, generalmente chiamato Identity Provider (IP), in grado appunto di fornire su richiesta di un determinato servizio, chiamato Relying Party (che vuole provare l'identità di un utente), un "lascia passare" (implementato attraverso un token crittografico). A questo punto all'utente è solo

richiesto di effettuare il login nell'IP e dare l'autorizzazione all'accesso ai propri dati ai vari Relying Party a cui vuole partecipare. Di fatto, uno degli esempi più famosi di SSO è quello offerto da Google, grazie al quale è possibile registrarsi ed effettuare il login su vari servizi semplicemente con un bottone (tipicamente chiamato "Accedi con Google"). Si viene a questo punto reindirizzati in una pagina di Google che mostra quali dati saranno trasmessi e l'utente dovrà decidere se consentire l'accesso o meno.

L'infrastruttura che si vuole realizzare dunque sarà molto simile a Google ma sarà implementata tramite un servizio rilasciato anch'esso nel cloud privato, le cui alternative saranno oggetto dei paragrafi successivi.

Quello che però necessita un ulteriore chiarimento è il "come" tale infrastruttura funziona. Il grafico generico in termini di autenticazione prevede la suddivisione dell'utente in due tipologie:

- Applicant: utente interessato ad effettuare la registrazione verso un servizio
- Claimant: utente, già registrato, che vuole richiedere, tramite login, i propri dati per condividerli con un determinato servizio

Questo porta alla suddivisione logica (ed eventualmente fisica) in due entità anche dell'Identy Provider:

- Credential Service Provider (CSP): entità in cui è possibile effettuare la registrazione
- Verifier: entità con cui interagisce il Claimant ed il Relying Party per effettuare il login

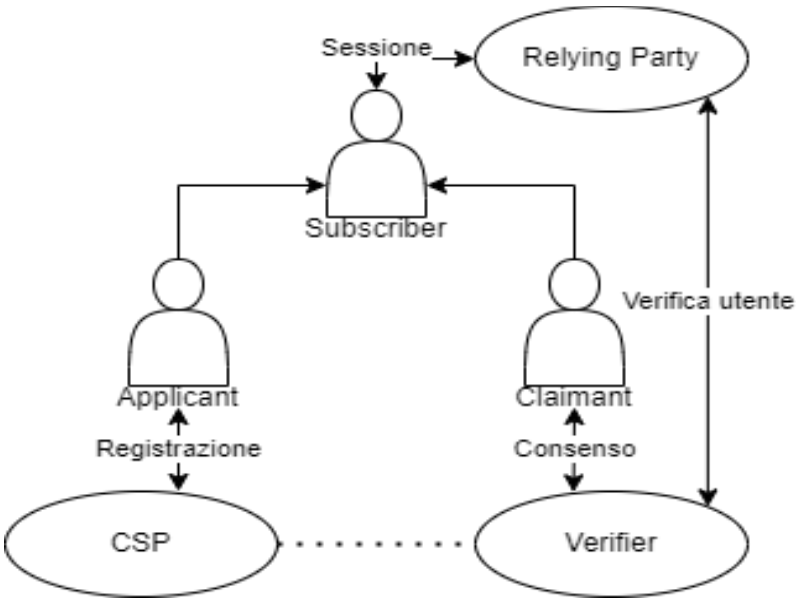


*Figura 16: Schema autenticazione*

La suddivisione troverà un significato più profondo nell'implementazione finale del progetto, in cui effettivamente le entità saranno fisicamente separate. Lo standard più diffuso per SSO è Oauth2, definito nell'RFC 6749. Questo rappresenta un framework di autenticazione tramite HTTP che prevede lo scambio di vari token tra le entità descritte fino ad ora. Sulla base di questo sono definiti dei veri e propri protocolli che implementano lo scambio e la struttura dei token. Quello selezionato per questo progetto è uno tra i più famosi: OpenID Connect (OIDC), sviluppato da OpenID nel 2014. OIDC quindi sfrutta i seguenti token nel formato JWT (JSON Web Token):

- Access Token: descritto già in Oauth2, è un token opzionale con un tempo di vita basso, permette l'accesso all'utente per una specifica risorsa definita nella richiesta
- Refresh Token: anch'esso presente in Oauth2, ha un tempo di vita decisamente più lungo e può essere utilizzato per richiedere nuovi Access Token
- ID Token: specifico di OIDC, contiene i dati che l'utente ha autorizzato a condividere con il RP, come ad esempio e-mail, nome utente o un codice identificatore univoco

Naturalmente questi token giocano un ruolo chiave per l'autenticazione, per cui è necessario garantire sicurezza di canale dove già l'uso del TLS tramite HTTPS risulta sufficiente.

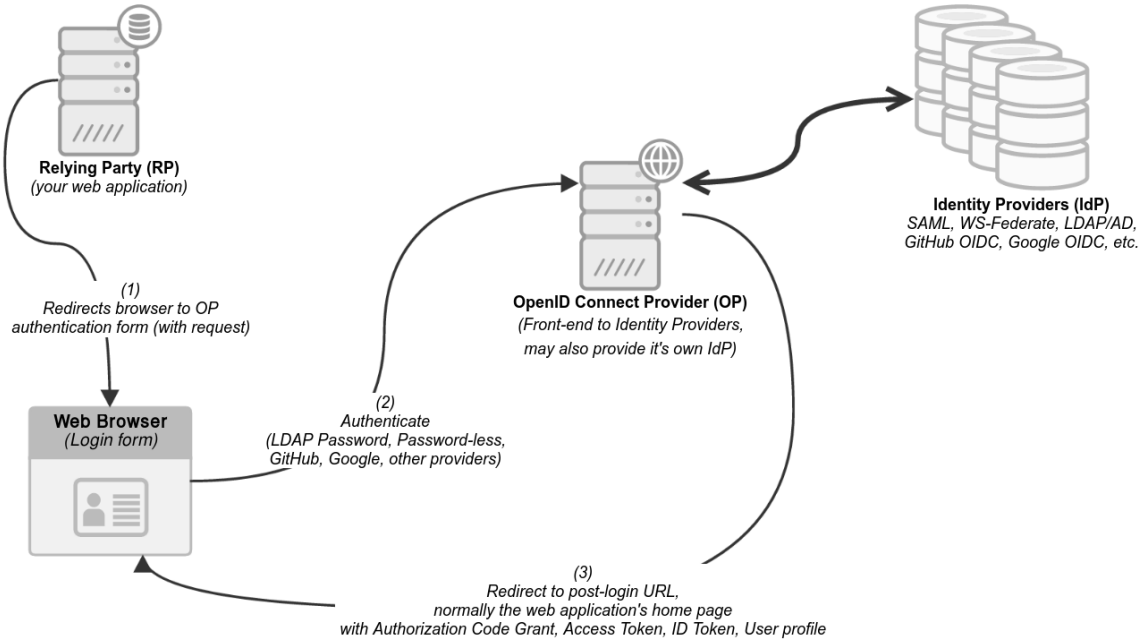


*Figura 17: Schema funzionamento OIDC*

Sia in Oauth2 che in OIDC non viene definita la procedura di login ma solo i metodi che lo sviluppatore potrebbe scegliere. Questi potrebbe selezionare username e password, dati biometrici, generazione di OTP (via applicazione, temporali, SMS, e-mail …) oppure ancora una combinazione di più alternative ottenendo un sistema a più fattori (MFA: Multi Factor Authentication). Il vantaggio di utilizzare il SSO è che tutto ciò sarebbe implementato in un unico nodo centrale.

### *4.5.1 Authentik*

Authentik è uno tra i più famosi progetti per SSO disponibile. Si tratta di una soluzione opensource nella sua versione di base ma presenta anche pacchetti Enterprise con l'aggiunta di supporto tecnico e strumenti di intelligenza artificiale per monitoring. Supporta vari protocolli di autenticazione, tra cui SAML ma anche e soprattutto, per il caso di studio, OIDC. L'applicazione include, oltre alle schermate di login, anche una dashboard per gli utenti in cui questi possono controllare delle statistiche sui propri accessi.

In generale, implementando il protocollo OIDC, riesce a coprire la principale esigenza del progetto, dispone inoltre di una documentazione molto dettagliata con esempi di integrazione con vari servizi, come NextCloud, Gitea e Grafana. Authentik inoltre offre in maniera built-in anche la funzionalità di registrazione degli utenti.

### *4.5.2 Authelia*

Authelia rappresenta anch'esso un progetto opensource, disponibile solo in tale edizione. Questi gestisce vari protocolli di SSO e vari metodi di autenticazione, ad esempio, oltre username e password, permette l'integrazione con google authenticator come TOTP provider e con DUO per autenticazione basata su notifica applicativa.

La sua interfaccia grafica si limita alla pagina di login e logout e in generale copre tutte le esigenze del progetto in termini di OIDC. Anche la sua documentazione è molto dettagliata e presenta esempi di integrazione con numerosi servizi, ma un importante punto a favore per Authelia è la sua funzionalità integrata di Accesso Control List, basata anch'essa sull'header "host" delle richieste HTTP. È possibile, infatti, definire dei gruppi di utenti per i quali è possibile accedere a determinate risorse piuttosto che altre, grazie ad un sistema di regole molto raffinate basate anche su espressioni regolari.

Authelia permette inoltre di avere un backend su LDAP, per poter utilizzare tale sistema come database per salvare le informazioni di utenti e gruppi.

Al contrario di Authentik non ha una funzionalità dedicata alla registrazione, che va gestita operando direttamente con il suo backend, ossia, nel caso di studio, LDAP.

| | Pro | Contro |
|---|---|---|
| **Authentik** | UI per monitoraggio, funzionalità registrazione, documentazione dettagliata, opensource | Enterprise aggiuntivo per supporto tecnico, assenza ACL |
| **Authelia** | Semplice, opensource, supporto per metodi autenticazione, ACL | Mancanza funzionalità registrazione |

*Tabella 6: Confronto tra le varie alternative per SSO Identity Provider*

## *4.6 Conclusioni*

Il progetto utilizzerà una infrastruttura VPN basata su WireGuard, viste le sue grandi potenzialità in termini di velocità che rendono il suo utilizzo praticamente trasparente all'utente. Naturalmente tra le alternative proposte per server VPN, l'unica in grado di supportare WireGuard è PIVPN, che comunque offre ottimi standard di affidabilità ed una grande semplicità di utilizzo.

La gestione del proxy sarà invece affidata a Nginx, nello specifico Nginx Proxy Manager che grazie alla sua UI rende semplice la configurazione di quest'ultimo non limitando lo spazio di personalizzazione.

Oltre alle varie applicazioni di storage, version control e data analysis, sarà presente una applicazione React, chiamata Hub, che permetta una facile redirezione verso i vari servizi ma anche la presenza, per gli amministratori, di una sezione dedicata agli Admin tools.

La scelta tra Authelia ed Authentik è relativamente più complessa, seppur entrambi offrano un sistema perfettamente funzionante per OIDC, ognuno ha le proprie funzionalità peculiari che li contraddistingue.

Il vantaggio principale per Authelia è la sua gestione delle ACL, mentre per Authentik la funzionalità built-in di registrazione. A proposito di quest'ultima, un elemento fino ad ora trascurato anche a livello di design, è stata proprio la registrazione, per cui si è deciso che la scelta più sicura per un sistema del genera sia quella di permettere la creazione di nuove utenze solo agli amministratori dell'applicazione, per cui all'interno dell'Hub è stato predisposto una sezione dedicata ad utenti e gruppi. A questo punto viene meno l'esigenza di una funzionalità di registrazione ma diventa essenziale la funzionalità di ACL per distinguere amministratori da altri tipi di utenti, per cui Authelia diventa la soluzione perfetta.

La gestione delle ACL lato Authelia impone che, prima di accedere ad una risorsa, si verifichi l'identità dell'utente che la richiede. In termini pratici, se un utente volesse accedere alle API che permettono la creazione di nuovi utenti (funzionalità amministrativa), bisognerebbe inoltrare la richiesta, lato server, ad Authelia. Tutto ciò è perfettamente compatibile con l'utilizzo di un reverse proxy, incaricato proprio di questo genere di compiti, l'utilizzo di Nginx permette infatti la flessibilità per eseguire tale operazione.

In conclusione, si può notare come l'infrastruttura di autenticazione veda un CSP e un Verifier distribuiti su entità fisiche differenti. Il Verifier sarà sempre Authelia, mentre il CSP sarà una combinazione tra Hub ed LDAP; allo stesso tempo si nota come il ruolo di applicant sia coperto sempre da amministratori, che si occupano della creazione di utenze, mentre il ruolo di claimant può essere coperto da chiunque sia stato precedentemente registrato.

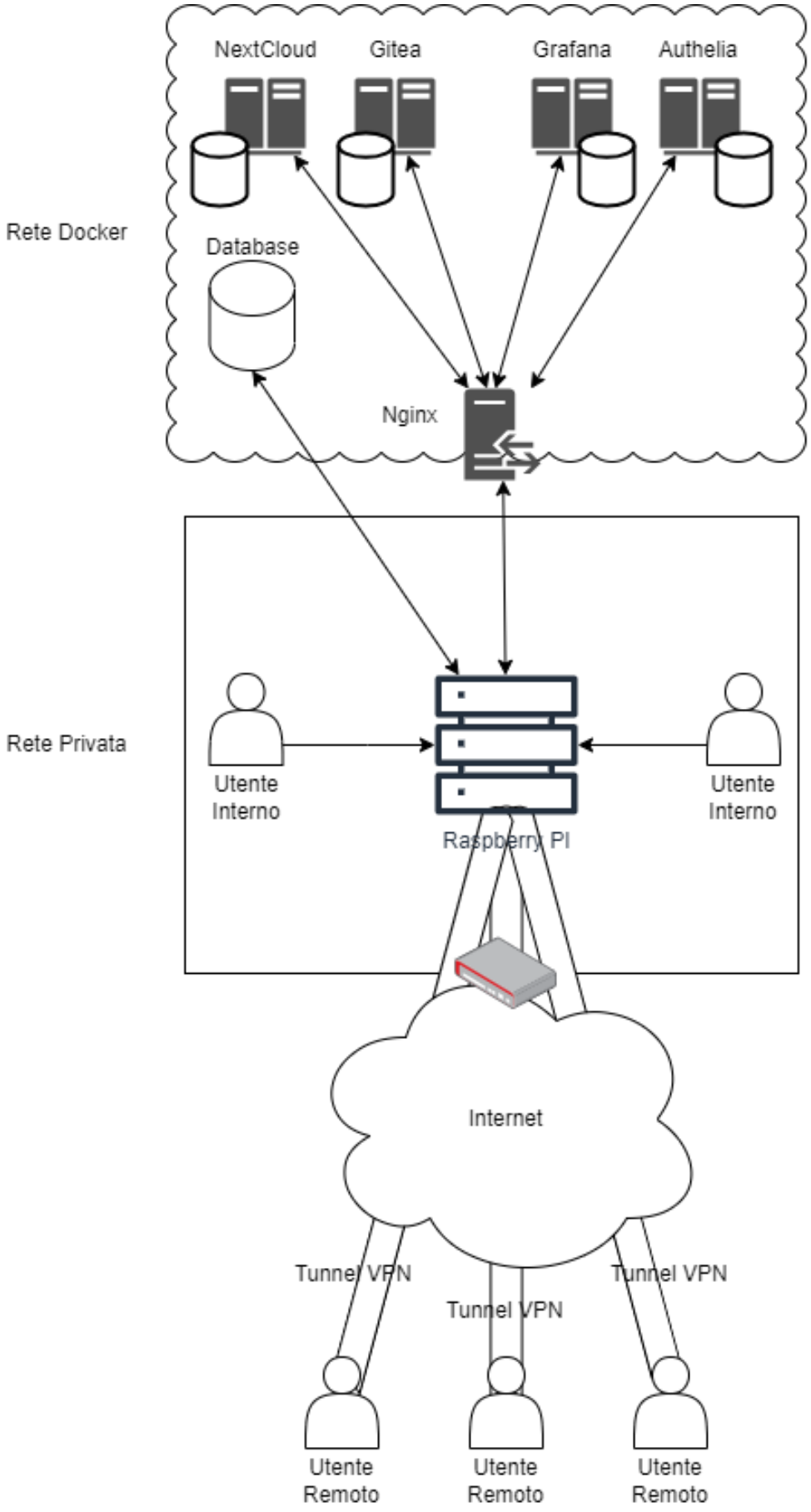


*Figura 18: Schema dell'intera infrastruttura di rete*

# Capitolo 5: Implementazione

Nella conclusione di questo progetto, verranno messi in risalto svariati elementi dell'implementazione vera e propria del sistema. La descrizione si concentrerà sui principali file di configurazione.

L'organizzazione del progetto prevede un folder dedicato all'applicazione Hub, implementata tramite React, e un folder dedicato invece al backend realizzato tramite il framework Expressjs. Hub punta ad essere una applicazione client side responsive grazie alla quale è possibile accedere ai vari servizi già descritti. Naturalmente è possibile accedere ai suddetti anche digitando il corretto nome di dominio nella barra di ricerca, ma Hub consente di facilitare l'operazione creando un ambiente user friendly per navigare.

Altre funzionalità di Hub sono gli admin tools, che permettono la creazione di nuove utenze e l'autorizzazione a quest'ultime ad accedere ai vari servizi. A seguito delle interazioni con la UI, saranno inviate delle richieste al server Express, il quale ha diretta visibilità con il backend del servizio di autenticazione (LDAP), grazie alle configurazioni di rete trattate nel capitolo precedente.

Nel server Express viene utilizzata la API "Ldapjs" per poter interagire con LDAP e quindi creare, modificare o cancellare le utenze.

All'interno del server è presente anche uno script che, all'avvio, effettua un sanity check dell'ambiente di LDAP, verificando l'esistenza dei vari gruppi e oggetti. Questo script è anche incaricato di ricreare l'utenza amministrativa o eventuali oggetti di servizio per LDAP in caso di errata cancellazione. Nel caso in cui un utente privilegiato cancellasse tutte le utenze amministrative, basterà riavviare il server, così da lanciare lo script, che creerà l'utenza "admin".

Per risolvere un problema di CORS, viene sfruttata la potenzialità di Nginx Proxy Manager di creare delle route custom, per cui, se in generale app.master.project.rg punta al servizio dedicato all'applicazione Hub, il path /api punta al server Express. [aggiungi conf]

Oltre alle cartelle dedicate a client e server, nella root folder del progetto sono presenti:

- dei file .env per impostare alcune variabili di ambiente, da usare per la compilazione di Docker (uno per utilizzare i dischi esterni e uno per creare un ambiente di sviluppo che non li usi)
- il file docker-compose.yml, che verrà approfondito nel prossimo paragrafo
- due cartelle di file di configurazione chiamate input e secrets

Quando il progetto viene avviato senza l'uso degli HDD, viene creata un'ulteriore cartella chiamata volumes (ignorata tramite il file .gitignore), dove vengono salvati i volumi persistenti creati da Docker.

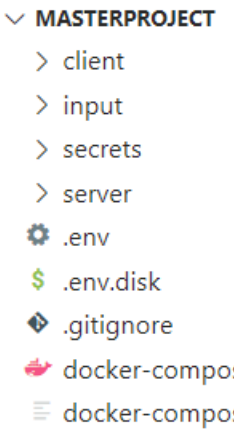


*Figura 19: root folder del progetto*

## *5.1 docker-compose.yml*

L'elemento centrale del progetto è proprio il file di docker-compose.yml. Proprio grazie aa suddetto vi è la creazione dei vari container, in un ambiente centralizzato e configurabile principalmente da questo file. Oltre alla versione, gli oggetti più importanti del file sono sotto gli attributi di "services" e "secrets".

Partendo da quest'ultimo, all'interno del sistema sono contenuti vari segreti, principalmente legati alla gestione delle sessioni di Authelia e ai certificati utili per il TLS. La successiva è l'entry relativa al TLS:

```
secrets:
  tls_private_key:
    file: ./secrets/tls/key.pem
```

Come si può notare, la sintassi permette di creare il puntamento ad un determinato file (./secrets/tls/key.pem) assegnandovi un nome, che poi sarà sfruttato come riferimento nel resto del docker-compose.yml.

Per quanto riguarda i servizi, invece, in seguito vi è l'analisi dei principali dove vengono messe in evidenza le varie componenti della sintassi: prima di tutto, bisogna considerare che vari servizi necessitano di un database applicativo, dove ad esempio vengono salvati dati di sessione. Per cui, quando necessario, nella stretta filosofia Docker è stato creato un container per ogni servizio contenente il relativo database. Per Nginx, ad esempio, è stato creato il seguente database applicativo:

```
  nginxdb:
    image: 'jc21/mariadb-aria:latest'
    container_name: nginxdb
    environment:
      MYSQL_ROOT_PASSWORD: '***'
      MYSQL_DATABASE: '***'
      MYSQL_USER: '***'
      MYSQL_PASSWORD: '***'
    volumes:
      - ${INTERNAL_DB_VOLUME}/nginxdb/data/mysql:/var/lib/mysql
```

Si può notare l'utilizzo di maria-db come immagine di partenza e l'utilizzo di variabili di ambiente interne al container per configurare l'utenza amministrativa, il database di default e la porta attraverso cui accedervi. È da sottolineare l'assenza dell'attributo "ports" in questo caso, in quanto la porta del database NON va esposta verso l'esterno ma deve restare accessibile solo dalla rete interna creata per default da Docker compose. L'unico che può accedere al database interno di Nginx sarà Nginx stesso e nessun utente esterno. Questa configurazione è analoga per tutti i database interni, l'unico database che si differenzia è quello dedicato ai dati di sensoristica, che espone la porta 3306 per permettere ai sensori di caricare dati.

È da evidenziare che, nella sezione dedicata ai volumi, sia presente anche la variabile d'ambiente "INTERNAL_DB_VOLUME", che rappresenta il base path in cui sono salvati i dati dei vari database interni. Nella configurazione del .env viene specificato tale percorso che può essere all'interno della stessa cartella del progetto, con un riferimento relativo a "./volumes", oppure all'interno dei dischi di memoria, con un riferimento assoluto a "/media/disk/db/data".

Per quanto riguarda la funzionalità di server DNS, come detto in precedenza, è stato utilizzato PiHole, con una semplice configurazione che permette di specificare le porte da esporre, ossia la 53, e le variabili d'ambiente per configurare i DNS da contattare per risoluzioni esterne. Oltre a ciò, è presente anche una sezione dedicata ai volumi in cui, analogamente ai casi precedenti, è presente un volume utile alla persistenza dei dati, ma dove compaiono anche dei riferimenti a volumi all'interno della cartella input, come si può notare dal seguente codice:

```
pihole:
  container_name: pihole
  image: pihole/pihole:latest
  ports:
    - "53:53/tcp"
    - "53:53/udp"
  environment:
    - DNS1=8.8.8.8
    - DNS2=8.8.4.4
    - ServerIP=10.23.200.1
    - WEBPASSWORD=***
  volumes:
    - ${PIHOLE_VOLUME}/etc-pihole:/etc/pihole
    - ./input/pihole/etc-dnsmasq.d:/etc/dnsmasq.d
    - ./input/pihole/etc-pihole/custom.list:/etc/pihole/custom.list
```

La cartella input in generale contiene delle configurazioni che potrebbero essere personalizzate in base all'ambiente di rilascio.

In questo caso, ad esempio, nella cartella "etc-dnsmasq.d" sono contenute le informazioni relative alle entry di tipo CNAME del DNS. L'utilizzo di questo tipo di entry è necessario per evitare di dover aggiornare tutte le entry nel caso in cui si volesse cambiare l'IP privato del PI.

```
cname=storage.master.project.rg,master.project.rg
cname=vc.master.project.rg,master.project.rg
cname=auth.master.project.rg,master.project.rg
cname=data.master.project.rg,master.project.rg
cname=app.master.project.rg,master.project.rg
```

All'interno invece della cartella etc-pihole vi è la configurazione del vero e proprio IP a cui punta "master.project.rg". Durante lo sviluppo, infatti, si sono verificati casi in cui era possibile collegare il PI al WIFI e quindi assegnargli un IP nella subnet 192.168.1.0/24. Ma nel caso in cui si fosse dovuto operare in gadget mode e quindi collegare il PI tramite ethernet direttamente al PC, sarebbe stato necessario utilizzato la subnet 192.168.137.0/24, per cui si sarebbe rivelato necessario operare solo in quest'ultimo file e modificarne l'IP.

Una parte delicata della configurazione, che prevede anche la fase di build del progetto, è legata alle immagini custom dell'applicazione Hub (client e server), prevedendo anche l'aggiunta dell'attributo build per specificare la locazione del Dockerfile, da cui appunto costruire l'intera applicazione.

```
server:
    image: server:latest
    container_name: server
    build:
      context: ./server
      dockerfile: ./Dockerfile
    depends_on:
      - openldap
  client:
    image: client:latest
    container_name: client
    build:
      context: ./client
```

Un esempio di Dockerfile è quello riportato in seguito del client:

```
# stage 1
FROM node:14-alpine as build
LABEL name="client"
WORKDIR /
ENV PATH /node_modules/.bin:$PATH
ENV REACT_APP_SERVER_HOST https://app.master.project.rg
ENV REACT_APP_GITEA_URL https://vc.master.project.rg/user/oauth2/authelia
ENV REACT_APP_NEXTCLOUD_URL
https://storage.master.project.rg/apps/oidc_login/oidc
ENV REACT_APP_NOCODB_URL https://data.master.project.rg

COPY package.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent
COPY . .
RUN npm run-script build

# stage 2
FROM nginx:1.17.8-alpine
COPY --from=build /build /usr/share/nginx/html
ENV REACT_APP_SERVER_HOST https://app.master.project.rg
ENV REACT_APP_NEXTCLOUD_URL
https://storage.master.project.rg/apps/oidc_login/oidc
ENV REACT_APP_NOCODB_URL https://data.master.project.rg
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Questo script prevede l'uso ottimizzato di Docker per poter creare una pipeline di build, caratterizzata da due stage: nel primo viene buildata l'applicazione React tramite un ambiente Node, nel secondo viene rilasciata su un server Nginx, che espone la porta 80.

All'interno dell'attributo relativo al server Express js, si può inoltre notare la presenza dell'attributo "depends_on". Questo permette allo script di Docker compose di specificare una dipendenza tra immagini, dato che il server, al suo avvio, proverà a collegarsi ad LDAP per effettuare il suo sanity check, e per poterlo fare è necessario che LDAP sia già up and running.

A questo punto si passa ad Nginx, dove viene creata una semplice istanza cui si aggiunge, tramite le variabili d'ambiente, il collegamento al database "nginxdb". I volumi permettono di specificare i file di configurazione dei vari reverse proxy; questi vengono scritti in automatico da Nginx Proxy Manager e per modificarli è sempre meglio affidarsi alla UI messa a disposizione.

Qui di seguito si riporta, oltre al codice relativo ad Nginx, un esempio di configurazione fatta tramite UI:

```yaml
nginx:
    image: 'jc21/nginx-proxy-manager:2.9.20'
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    environment:
      DB_MYSQL_HOST: "***"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "***"
      DB_MYSQL_PASSWORD: "***"
      DB_MYSQL_NAME: "***"
    volumes:
      - ./input/nginx/proxy_hosts:/data/nginx/proxy_host
      - ./input/nginx/snippets:/config/nginx/snippets:ro
      - ${NGINX_VOLUME}/data:/data
      - ${NGINX_VOLUME}/letsencrypt:/etc/letsencrypt
    depends_on:
      - nginxdb
```

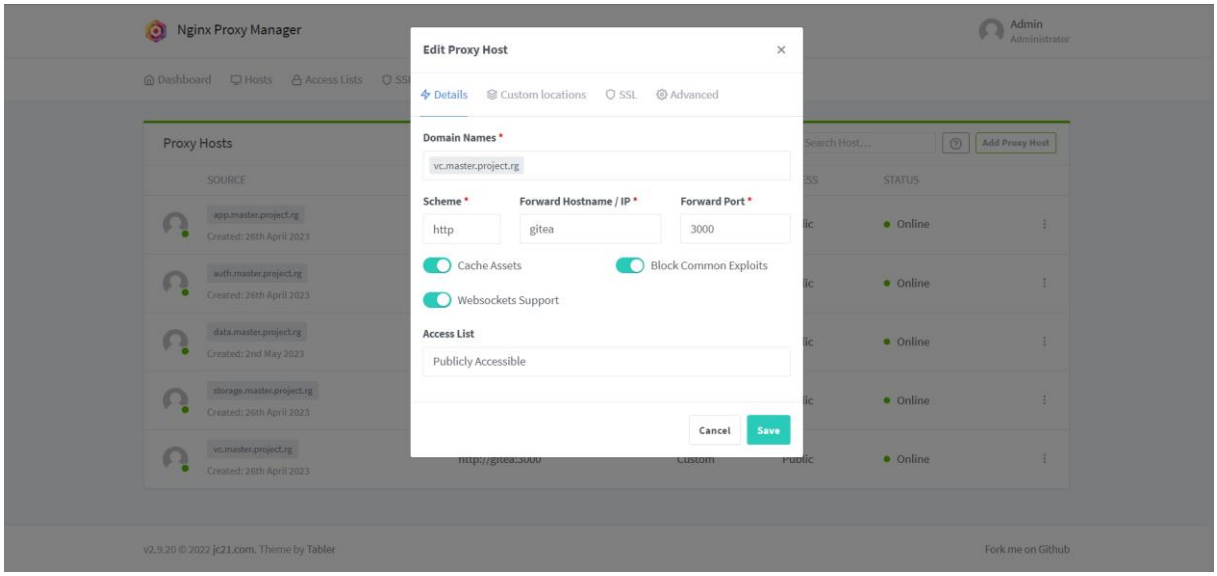


*Figura 20: Screenshot configurazione di base (esempio con Gitea)*

Il prossimo step sarebbe quello relativo ad Authelia ma visto che la sua configurazione, oltre a docker-compose, prevede degli elementi molto importanti in termini di access control, le verrà dedicato interamente il prossimo paragrafo.

L'ultimo punto è quello relativo alle tre immagini di NextCloud, Gitea e Grafana. La loro configurazione all'interno del docker-compose.yml è molto simile, per cui ne verrà analizzata solo una, ossia quella di NextCloud:

```
nextcloud:
    image: nextcloud:latest
    build:
      context: ./input/nextcloud
      dockerfile: ./Dockerfile
    container_name: nextcloud
    volumes:
      - ${NEXTCLOUD_VOLUME}:/var/www/html
      - ${NEXTCLOUD_DATA_VOLUME}:/var/www/html/data
      - ./input/nextcloud/cert:/usr/local/share/ca-certificates
      -
./input/nextcloud/conf/config.php:/var/www/html/config/config.php
    environment:
      - MYSQL_DATABASE=***
      - MYSQL_USER=***
      - MYSQL_PASSWORD=***
      - MYSQL_HOST=***
    depends_on:
      - nextclouddb
      - authelia
      - nginx
      - pihole
```

Oltre ai tipici attributi per volumi, che gestiscono la persistenza dei dati, variabili d'ambiente, per la configurazione del database interno, e dipendenze, per la filiera di build, si aggiunge anche in questo caso l'attributo build. Il Dockerfile, in questo caso, è molto semplice. Viene riportato in seguito, ma esercita un ruolo fondamentale ai fini della configurazione dell'SSO.

```
FROM nextcloud:latest
ADD ./cert/auth.pem /usr/local/share/ca-certificates/auth.pem
RUN chmod 644 /usr/local/share/ca-certificates/auth.pem
RUN cat /usr/local/share/ca-certificates/auth.pem >> /etc/ssl/certs/ca-
certificates.crt
```

Nel pratico, aggiunge alle autorità di certificazione fidate del container, anche quella creata custom per l'intero progetto. Questo è necessario, in quanto i vari servizi, vestendo il ruolo di Relying Party, come specificato nel capitolo precedente, devono interagire con Authelia e, per le specifiche di OIDC, utilizzano https per garantire sicurezza di canale. Quindi, è necessario che i vari servizi si fidino del certificato inviato dal Verifier, ossia Authelia.

Con quest'ultima descrizione si conclude la configurazione dettata dal docker-compose.yml, ad eccezione di Authelia.

## *5.2 Configurazione Authelia*

La configurazione di Authelia contiene in sé vari elementi caratterizzanti, per cui le è stato dedicato questo paragrafo per approfondire al meglio.

Prima di tutto, vi è la sua installazione tramite docker-compose:

```
  authelia:
    restart: always
    container_name: authelia
    image: authelia/authelia
    ports:
      - 9091:9091
    volumes:
      - ./input/authelia/config/configuration.yml:/config/configuration.yml
      -
./input/authelia/config/users_database.yml:/config/users_database.yml
    depends_on:
      - autheliadb
      - openldap
    secrets:
      - authelia_jwt
      - authelia_session
      - authelia_smtp_password
      - authelia_hmac
      - authelia_encryption_key
      - tls_private_key
    environment:
      - AUTHELIA_JWT_SECRET_FILE=/run/secrets/authelia_jwt
      - AUTHELIA_SESSION_SECRET_FILE=/run/secrets/authelia_session
      - AUTHELIA_NOTIFIER_SMTP_PASSWORD_FILE=
/run/secrets/authelia_smtp_password
      - AUTHELIA_IDENTITY_PROVIDERS_OIDC_HMAC_SECRET_FILE=
/run/secrets/authelia_hmac
      - AUTHELIA_STORAGE_ENCRYPTION_KEY_FILE=
/run/secrets/authelia_encryption_key
      - AUTHELIA_IDENTITY_PROVIDERS_OIDC_ISSUER_PRIVATE_KEY_FILE=
/run/secrets/tls_private_key
```

Come si può notare, oltre agli attributi già descritti in precedenza per altri servizi, in questo caso vengono usati i segreti precedentemente configurati. Attraverso l'attributo "secrets" è possibile specificare quali segreti utilizzare, i relativi file saranno poi collocati all'interno della cartella "/run/secrets" del container con lo stesso nome del segreto configurato in Docker. Praticamente la totalità dei segreti gestiti dal progetto è usata da quest'immagine, perché effettivamente è lei a controllare la sessione e l'autenticazione. Authelia poi è in grado di utilizzare i segreti quando questi sono configurati tramite variabili d'ambiente. Quando si usano variabili che terminano per "_FILE", queste punteranno a file contenenti i relativi segreti, altrimenti è possibile custodire i segreti direttamente nelle variabili d'ambiente, omettendo tale suffisso. Per quanto riguarda il design, la prima variante è stata la preferita, in quanto permette di nascondere al meglio i segreti, che

potrebbero anche non essere caricati in un eventuale repository, essendo in una locazione propria.

Authelia richiede che la sua porta sia esposta, in quanto il proxy sarà incaricato di mandare un redirect verso un determinato IP al client. Quando il client invocherà la redirezione, questa non sarà verso un nome di dominio ma verso una coppia IP e porta.

Per approfondire meglio è necessario comprendere come viene configurato Nginx per poter gestire il SSO. Prima di tutto è necessario configurare l'host relativo ad Authelia, oltre alla configurazione analoga a quella mostrata in figura 20, bisogna accedere al tab "Custom" e aggiungere del codice:

```
1  set $upstream_authelia http://192.168.1.115:9091;
2  proxy_pass $upstream_authelia;
3  client_body_buffer_size 128k;
4  proxy_next_upstream error timeout invalid_header http_500 http_502
      http_503;
5  send_timeout 5m;
6  proxy_read_timeout 360;
7  proxy_send_timeout 360;
8  proxy_connect_timeout 360;

9  proxy_set_header Host $host;
10 proxy_set_header X-Real-IP $remote_addr;
11 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
12 proxy_set_header X-Forwarded-Proto $scheme;
13 proxy_set_header X-Forwarded-Host $http_host;
14 proxy_set_header X-Forwarded-Uri $request_uri;
15 proxy_set_header X-Forwarded-Ssl on;
16 proxy_redirect  http://  $scheme://;
17 proxy_http_version 1.1;
18 proxy_set_header Connection "";
19 proxy_cache_bypass $cookie_session;
20 proxy_no_cache $cookie_session;
21 proxy_buffers 64 256k;

22 set_real_ip_from 192.168.1.0/24;
23 real_ip_header CF-Connecting-IP;
24 real_ip_recursive on;
```

Le parti più rilevanti sono quelle in cui viene impostato un IP o una subnet (righe 1 e 22) e la riga "proxy_pass $upstream_authelia" in cui viene specificato il nome dell'host all'interno della rete docker a cui inoltrare le richieste (authelia). Il resto è l'aggiunta di alcuni header per gestire sessione, buffer e timeout.

Successivamente bisogna configurare le applicazioni gestendo due path: "/authelia" e "/", come mostrato dal seguente codice (esempio per Gitea):

```
location /authelia {
    include /config/nginx/snippets/application_authelia_path.conf;
}

location / {
set $upstream_gitea $forward_scheme://$server:$port;
proxy_pass $upstream_gitea;
include /config/nginx/snippets/application_root_path.conf;
}
```

Come si può notare, essendo questa parte abbastanza ripetitiva per i vari servizi, è stato fatto uso di snippet e la parte custom che varia in funzione del servizio è stata esplicitamente indicata: le prime due righe della configurazione del path "/" hanno infatti riferimenti al nome "gitea", ossia il nome dell'host del servizio all'interno della rete Docker. Per quanto riguarda il path "/authelia" la configurazione prevede come già citato vari header per la gestione della cache e dei timeout, la parte più rilevante sono le prime due righe:

```
internal;
set $upstream_authelia http://192.168.1.115:9091/api/verify;
proxy_pass_request_body off;
proxy_pass $upstream_authelia;
proxy_set_header Content-Length "";
         [….]
```

Qui si evince la redirezione del path "/api/verify" verso l'host nella rete Docker "authelia". Il funzionamento di base prevede infatti che all'accesso all'applicazione, il reverse proxy effettui un check della sessione, quindi, atterrando su un qualsiasi path, viene attivata la regola definita da "/", che redirige internamente ad "/authelia" da cui poi si fa il check tramite "/api/verify". A conferma di ciò si può analizzare lo snippet relativo al path "/":

```
auth_request /authelia;
auth_request_set $target_url https://$http_host$request_uri;
         [….]
```

Nelle righe successive vi è una configurazione di cache e timeout, la parte degna è la prima riga, in cui effettivamente si redirige verso il path "/authelia".

Tutte queste configurazioni avanzate di Nginx sono riprodotte in tutti i servizi tranne che in NextCloud. Il motivo di ciò è legato alla ragione stessa per cui sono state fatte le stesse configurazioni. L'idea è quella di sfruttare il servizio di ACL offerto da Authelia, per cui, prima di accedere ad una risorsa, si verifica se l'utente che ne fa richiesta sia autenticato ed autorizzato, quindi è necessario sin da subito fare appello al Verifier. NextCloud è dotato di una configurazione interna che permettere l'accesso solo agli utenti di un determinato gruppo OIDC, che è un controllo necessario, come si vedrà a breve, per verificare l'autorizzazione. Oltre, dunque, a non essere utile ma ridondante l'aggiunta della

configurazione su Nginx, si rischia che l'applicazione mobile non riesca ad effettuare correttamente il login, per cui il tab custom su Nginx di NextCloud è rimasto vuoto.

Il prossimo step dell'analisi di Authelia è legato alla configurazione del container, effettuata tramite "input/authelia/config/configuration.yml", specificato nel docker-compose.yml. Innanzitutto, i vari segreti riportati come variabili d'ambiente vengono automaticamente caricati, altrimenti sarebbe stato necessario specificarli in questo file.

La prima parte riguarda delle proprietà di base dell'istanza di Authelia:

```
default_redirection_url: https://app.master.project.rg

server:
  host: 0.0.0.0
  port: 9091
```

Nel pratico vi è la configurazione del nome di dominio dove indirizzare di default al login, qui è stato inserito quello definito tramite PiHole, e la locazione di rete dell'istanza, che si trova nello stesso container del file di configurazione, per cui si inserisce un indirizzo fittizio e la porta 9091.

Successivamente vi è la configurazione per il backend su LDAP:

```
authentication_backend:
  ldap:
    implementation: custom
    url: ldap://openldap
    start_tls: false
    base_dn: DC=master,DC=project,DC=rg
    additional_users_dn: OU=People
    users_filter: (&({username_attribute}={input})(objectClass=person))
    username_attribute: cn
    mail_attribute: mail
    display_name_attribute: uid
    additional_groups_dn: OU=Groups
    groups_filter: (&(member={dn})(objectClass=groupOfNames))
    group_name_attribute: cn
    permit_referrals: false
    permit_unauthenticated_bind: false
    user: ****
    password: ****
```

In questo caso vi sono riportati tutti i riferimenti ai domini LDAP per la gestione di utenti e gruppi, oltre che l'utenza amministrativa utilizzata solo da Authelia per il login (Verifier) e dal backend di Hub per la registrazione (CSP). È Bene notare che all'interno del servizio di Authelia come per qualunque altro non vi sia l'attivazione del TLS, in quanto questo, come detto in precedenza, è gestito da Nginx.

La prossima parte rappresenta forse il nodo centrale del file di configurazione, si tratta infatti della ACL:

```yaml
access_control:
  default_policy: deny
  rules:
    - domain:
        - "auth.master.project.rg"
      policy: bypass
    - domain: "data.master.project.rg"
      policy: one_factor
      subject: 'group:database'
    - domain: "vc.master.project.rg"
      policy: one_factor
      subject: 'group:gitea'
    - domain: "storage.master.project.rg"
      policy: one_factor
      subject: 'group:nextcloud'
    - domain: "app.master.project.rg"
      policy: one_factor
      subject: 'group:admin'
      resources:
        - '^/api/admin.*'
    - domain: "app.master.project.rg"
      policy: deny
      resources:
        - '^/api/admin.*'
    - domain: "app.master.project.rg"
      policy: one_factor
```

Le regole seguono il formato simile ad un firewall in cui la priorità dipende dall'ordine di presentazione. Gli aspetti principali sono legati alla policy, che assume il valore "un fattore" (username e password) e potrebbe eventualmente essere aggiornata per avere ulteriori fattori di autenticazione, e al subject, con cui si specifica la condizione secondo cui è possibile accedere a tale risorsa. Ad esempio, 'group:gitea' specifica che possono accedere solo gli utenti nel gruppo 'gitea'. Per gestire gli admin tools, è stata protetta la route '/api/admin' per il backend (a livello di front end è stato gestito direttamente su React). Sfruttando la priorità nell'ordine delle regole, le ultime tre possono essere riassunte in linguaggio naturale come:

- Permetti l'accesso agli utenti del gruppo admin a '/api/admin'
- Se sono arrivato qua, sto richiedendo una risorsa terza (ok) o sto provando ad accedere a '/api/admin' senza essere admin (ko), quindi gestisco il secondo caso vietando l'accesso a '/api/admin'
- Se sono arrivato qua, sicuramente non sto accedendo a '/api/admin', per cui, se sono loggato, ho accesso a qualunque risorsa

I prossimi punti sono invece relativi alla configurazione del database (alla quale non si aggiunge nulla essendo abbastanza auto esplicativa):

```
storage:
  mysql:
    host: ***
    port: ***
    database: ***
    username: ***
    password: ***
```

E del notifier:

```
notifier:
  smtp:
    username: master.project.rg@gmail.com
    sender: no-reply@master.project.rg
    host: smtp.gmail.com
    port: 587
```

A proposito di quest'ultimo, bisogna sottolineare la necessità di un'infrastruttura per mandare delle email. Quando viene effettuata la registrazione, l'amministratore inserisce solamente nome utente ed email, poi l'utente stesso potrà provare ad effettuare il primo login e resettare la password tramite un link accessibile tramite la richiesta di una email. Ai fini di ciò è stato utilizzato il server SMTP fornito gratuitamente da Google.

L'ultima parte del file di configurazione riguarda invece OIDC. Per ogni servizio è necessario specificare un attributo all'interno di "clients" analogo all'esempio di Gitea riportato:

```
identity_providers:
  oidc:
    cors:
      allowed_origins: "*"
      endpoints:
        - authorization
        - token
        - revocation
        - introspection
        - userinfo
    clients:
    - id: gitea
      description: Gitea
      secret: '$plaintext$***'
      public: false
      authorization_policy: one_factor
      redirect_uris:
        - https://vc.master.project.rg/user/oauth2/authelia/callback
      scopes:
        - openid
```

```
      - email
      - profile
    userinfo_signing_algorithm: none
```

Oltre ad una parte generica di configurazione di OIDC, in cui è stata configurata la gestione CORS, per ogni client è fondamentale specificare id, segreto (in questo caso censurato), politica di autenticazione (analogamente alla ACL), redirect_uris (dove redirigere a valle dell'autenticazione) e scopes (che saranno le proprietà dell'utente a cui si chiede accesso).

Quando poi bisognerà configurare il servizio per utilizzare OIDC, sarò necessario impostare, tramite file di configurazione o GUI, l'host dell'identity provider (https://auth.master.project.rg) a cui si aggiunge un path (/.well-known/openid-configuration) che permette una discovery automatica dei vari path dei token. Oltre a ciò, bisognerà inserire l'id del client, nel caso in esempio 'gitea', e il segreto come inserito nella configurazione di Authelia.

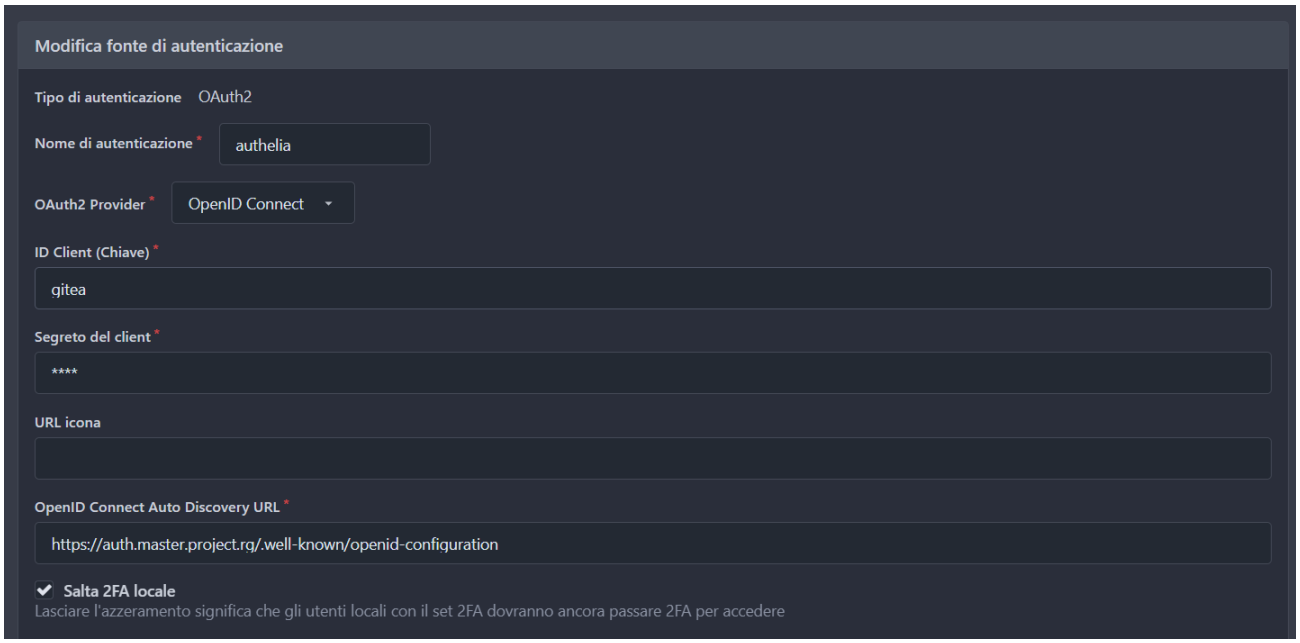


*Figura 21: Screenshot configurazione di Gitea per SSO*

## *5.3 TLS*

L'ultimo aspetto, citato già in varie parti dell'elaborato, riguarda la sicurezza del sistema in termini di comunicazione riservata ed autenticata. In generale, già la VPN permette di implementare meccanismi di crittografia, ma, per poter ottenere un sistema più sicuro e compatibile con l'utilizzo di OIDC, è necessario predisporre il Transport Layer Security (TLS). Già altri aspetti crittografici erano stati trattati parlando di sicurezza dei dati a riposo, grazie alla cifratura effettuata da NextCloud. In questo caso si parla invece di sicurezza di canale, il cui obbiettivo è quello di mitigare il tipico attacco del Man In The Middle (MITM).

Il MITM è un semplice attacco che consiste nell'effettuare uno sniffing dei pacchetti che attraversano la rete, offrendo innanzitutto visibilità ai dati sensibili in transito, ma anche

la possibilità di fingersi un interlocutore e quindi effettuare delle modifiche ai pacchetti o un invio multiplo dello stesso (attacco di tipo replay).

Per poter mitigare questo genere di vulnerabilità, la soluzione è innanzitutto quella di cifrare il traffico. Da qui si aprono varie opzioni legate alla scelta dell'algoritmo di cifratura, in termini di complessità (a cui seguono eventuali ritardi algoritmici) e livello di sicurezza. Oltre a ciò, è necessario garantire l'identità degli utenti coinvolti, al fine di evitare che un terzo utente si spacci per un altro. Infine, bisogna numerare i pacchetti, così da evitare invii multipli o accorgersi di pacchetti mancanti.

La risposta a queste necessità può essere implementata in due maniere, da una parte si potrebbe riscrivere una versione sicura dei vari protocolli, che includa degli header per i metadati crittografici (es: S-http), dall'altra si potrebbe invece implementare un ulteriore livello nella pila ISO-OSI dedicato esclusivamente alla sicurezza di canale. Questa seconda scelta offre il vantaggio di avere una soluzione centralizzata per tutti i protocolli applicativi, che quindi può essere scritta e manutenuta molto più facilmente, limitando al minimo la possibilità di vulnerabilità.

Netscape communication adottò proprio questa strategia nello sviluppo del suo protocollo Secure Socket Layer, con l'obbiettivo di rendere viabile la possibilità di fare commercio in rete (e-commerce). Si rende obbligatoria l'autenticazione del server ma non del client. Il protocollo venne poi standardizzato come TLS, ad oggi la versione più utilizzata è la 1.2 nonostante la 1.3 sia già disponibile.

Nello specifico, TLS effettua diverse operazioni tramite vari moduli:

- TLS Record Module: si occupa delle operazioni crittografiche sul payload, per cui frammenta i dati, vi applica una compressione, calcola l'HMAC (funzione di hashing che permette di ottenere un "riassunto" a lunghezza fissa del payload, utile per poter verificare se un utente malevolo ha applicato delle modifiche), aggiunge eventuale padding, applica la cifratura e aggiunge l'header TLS (che contiene importanti informazioni come un numero di sequenza per mitigare attacchi replay).
- TLS Alert Protocol: utile per segnalare eventuali anomalie rilevate.
- TLS Change Cipher Spec Prtocol: inserito in via precauzionale, serviva per permettere al sistema di riprendersi in caso di deadlock, successivamente rimosso in quanto ritenuto inutile.
- TLS Hanshake Protocol: una parte fondamentale del protocollo è proprio legato alla negoziazione dei parametri. Ai fini dell'utilizzo del TLS è necessario che, a valle della richiesta di una determinata risorsa da parte del client, il server presenti il proprio certificato di chiave pubblica, con cui il client (a seguito di una verifica della validità) potrà effettuare una sfida. Conclusa la sfida i due interlocutori negoziano una chiave simmetrica effimera che resterà attiva solo per la sessione.

Quest'ultimo punto mette in evidenza vari aspetti fondamentali. Innanzitutto, il primo contatto tra le entità si basa sull'utilizzo di crittografia asimmetrica, dove generalmente si utilizzano chiavi RSA. Per poter garantire che una determinata chiave pubblica RSA sia legata all'identità di uno specifico server vi è una infrastruttura di certificati firmati da autorità fidate in maniera eventualmente gerarchica. Il client può risalire la gerarchia fino ad accertarsi che il certificato proposto dal server sia effettivamente autentico. Verificata la validità del certificato, un utente malevolo potrebbe comunque

fingersi il server della comunicazione, per cui il client effettua la cifratura di un pacchetto usando la chiave pubblica del certificato. Solo il server, disponendo della chiave privata, potrà decifrarlo.

Arrivati a questo punto, il client conferma l'identità del server, opzionalmente anche il server potrebbe verificare l'identità del client. Dopodiché, entrambe le entità negoziano una chiave simmetrica da usare per il resto della comunicazione, generalmente con algoritmi come Diffie-Hellman dove entrambi influenzano la scelta.

Dal punto di vista dell'implementazione, bisogna evidenziare che il progetto utilizza una gerarchia di certificati dove la radice è "self-signed", per cui non si fa uso dell'infrastruttura pubblica. Questo porta alla necessità di installare manualmente il certificato della root CA nei dispositivi che vogliono utilizzare l'applicazione. Tutto ciò potrebbe essere facilmente evitato ottenendo un certificato da una autorità pubblica, il che potrebbe essere un facile aggiornamento del progetto.

La generazione delle CA e dei certificati sfrutta la libreria "openssl", per cui sono citati i comandi e una loro descrizione:

```
openssl genrsa -aes256 -out ca-key.pem 2048
```

1. Tramite questo comando è possibile creare la chiave privata della root ca, questa sarà lunga 2048 bit. Il parametro aes256 permette di cifrare il file tramite tale algoritmo, usando come chiave una password inserita dall'utente

```
openssl req -new -x509 -sha256 -days 365 -key ca-key.pem -out ca.pem
```

2. Tramite questo comando è possibile ottenere il certificato di chiave pubblica a partire dalla chiave privata precedentemente creata

```
openssl genrsa -out cert-key.pem 2048
```

3. Con questo comando, analogamente al punto 1, è possibile creare una nuova chiave privata, che sarà quella utilizzata dal TLS

```
openssl x509 -req -sha256 -days 365 -in cert.csr -CA ca.pem -CAkey ca-key.pem -out cert.pem -extfile extfile.cnf -CAcreateserial
```

4. Tramite questo comando, infine, è possibile firmare il certificato generato per il TLS con la CA generata in precedenza

```
cat cert.pem > fullchain.pem && cat ca.pem >> fullchain.pem
```

5. Quest'ultimo passaggio opzionale permette di creare un file che contenga la cascata dei certificati, così da accelerare le operazioni di verifica.

Fatto ciò, l'ultima configurazione è quella legata a Nginx Proxy Manager per permettergli di utilizzare il TLS con la filiera di certificati e chiavi generati. Per farlo, è

necessario accedere al tab dedicato a SSL e creare una configurazione custom, inserendo un nome, il file cert-key.pem (contenente la chiave privata), il file cert.pem (contenente il certificato) e, opzionalmente, il file fullchain.pem (contenente la catena di risoluzioni). Durante la creazione dei vari reverse proxy sarà necessario solo indicare a Nginx di utilizzare la configurazione TLS appena creata.

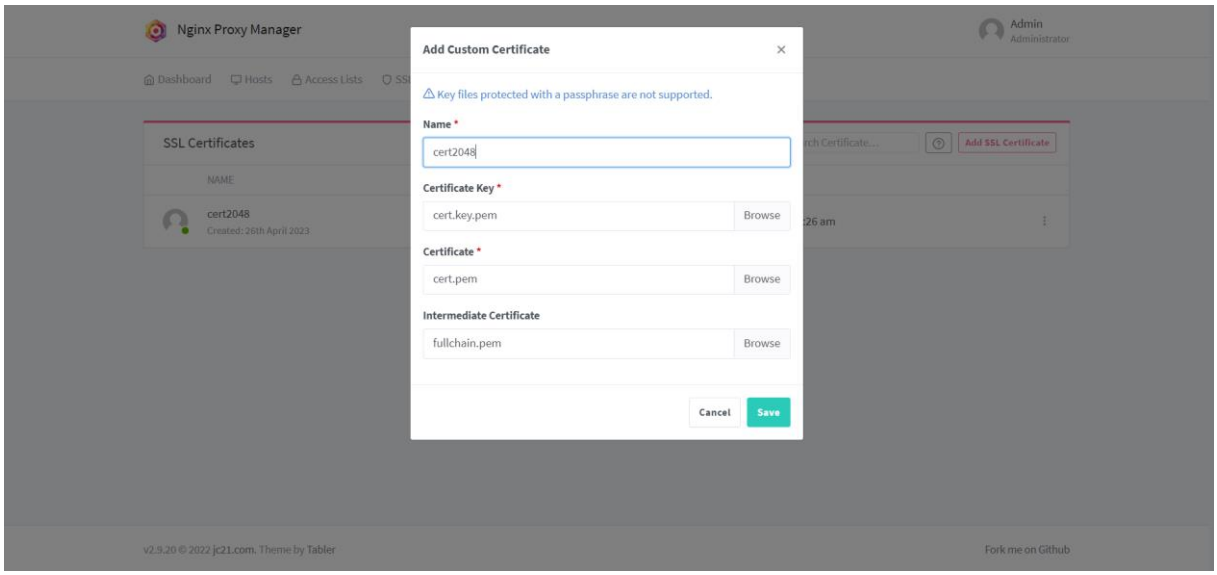


*Figura 22: configurazione TLS su Nginx Proxy Manager*

## *5.4 Configurazione dei dischi di memoria*

Al fine di poter utilizzare i dispositivi di archiviazione è necessario effettuarne una configurazione via hardware, impostando gli interruttori relativi al tipo di ridondanza su RAID 1, e via software creando delle partizioni per suddividere logicamente le varie aree. Nello specifico, sono state create tre partizioni per dati interni (come i database applicativi dei vari servizi), database per dati di sensoristica e file system per lo storage.

Per poter creare le partizioni si è fatto uso del comando Linux “fdisc” con il quale è possibile selezionare le operazioni, in successione:

- n: per creare una nuova partizione
- p: per selezionare il tipo primario di partizione
- 1: per selezionare il numero della partizione (per le successive inserire 2,3)
- <valore intero>: per selezionare il primo blocco della partizione
- <valore intero>: per selezionare l’ultimo blocco della partizione

L’ultimo step è quello di montare i dischi, per cui è stato predisposto il seguente script:

```
sudo mkfs.ext4 /dev/sdb1
sudo mkfs.ext4 /dev/sdb2
sudo mkfs.ext4 /dev/sdb3
sudo mount /dev/sdb1 /media/disk/internal
sudo mount /dev/sdb2 /media/disk/db
sudo mount /dev/sdb3 /media/disk/storage
```

# Conclusioni

Il progetto, in conclusione, si poneva l'obbiettivo di designare una soluzione ai requisiti del dipartimento di ricerca della CalState LA, creando un sistema che potesse essere installato in contesti privati. L'obbiettivo è stato dunque raggiunto con successo, per cui HomeCloud sarà utilizzato dal dipartimento e nella sede della ditta Innotech.

Lo studio e l'analisi dietro quest'elaborato hanno dimostrato con efficacia la fattibilità di sistemi anche molto complessi sfruttando le potenzialità di ambienti virtualizzati tramite Docker, con una richiesta di hardware molto limitata. Secondo una personale proiezione, immagino che, nel breve-medio termine, avere una soluzione del genere nelle proprie case possa diventare la normalità. Per come descritto nel primo capitolo, infatti, seppur da una parte le esigenze di risorse, soprattutto in termini di storage, sono sempre più elevate, sono disponibili sul mercato dispositivi sempre più potenti e performanti in un rapporto sempre decrescente rispetto al prezzo, per cui, nel caso dello storage, è facile ed economico ottenere dispositivi prestanti.

L'istallazione di un sistema di HomeCloud nella propria abitazione permetterebbe, ad esempio, di avere a disposizione centinaia di GigaByte accessibili da qualunque parte del mondo. Nella mia esperienza di viaggio mi sono banalmente trovato nella condizione di non poter effettuare ulteriori foto per avere ricordo dei fantastici momenti vissuti negli States proprio a causa di questo genere di problematica. L'avere una soluzione centralizzata, in un luogo sicuro ed accessibile da qualunque parte del mondo mi avrebbe permesso di gestire al meglio la memoria disponibile sui miei vari apparecchi.

Inoltre, l'utilizzo di sistemi del genere abilita la possibilità di utilizzare in maniera più efficiente le risorse offerte dai propri dispositivi, evitando di dedicarle alla conservazione di numerosi file che molto spesso sarebbero presenti in copie duplicate tra i vari devices. Se, ad esempio, un utente volesse visualizzare una immagine, accedere ad un pdf e così via sia dal proprio cellulare che dal proprio PC, questi dovrà necessariamente averne una copia per ogni dispositivo. Questo fenomeno sarebbe escluso utilizzando il cloud privato, in quanto entrambi i dispositivi accederebbero a tale risorsa salvata in uno spazio remoto, che per essere visualizzata richiederà al massimo di essere salvata temporaneamente ed automaticamente in una cache dedicata.

Ho trovato lo sviluppo del sistema particolarmente appagante, soprattutto nelle scelte di design e nella ricerca delle varie alternative per i vari servizi. Tutto ciò mi ha permesso per la prima volta di conoscere la comunità informatica open source, dove vari esperti mettono a disposizione il proprio materiale agli altri utenti, che possono analizzarlo e testarlo. Mi è capitato personalmente di creare issues su GitHub per chiedere informazioni su alcune implementazioni, trovando subito gli sviluppatori disponibili a dare chiarimenti o valutare l'aggiunta di nuove funzionalità ai servizi.

L'idea di utilizzare solo opzioni open source nasce proprio dal voler, nel mio piccolo, dare supporto e visibilità a questa realtà, caratterizzata dalla collaborazione nell'ottica di lavorare insieme ed offrire al mondo nuovi servizi e nuove opportunità, proprio per rispondere alle necessità dell'uomo e quindi migliorarne la vita. Molti aspetti dell'informatica sono nati con questa filosofia e il mio percorso di studio mi ha insegnato a cogliere questa ideologia. Spero nella vita di avere sempre la possibilità di creare qualcosa di mio, per aiutare il mondo a crescere diffondendo pubblicamente le mie conoscenze e le mie ricerche come ho fatto con questa tesi.

# Bibliografia


[1] Raspberry Pi Cloud Storage Software Solutions, https://www.makeuseof.com/raspberry-pi-cloud-storage-software-solutions

[2] RAID controller, https://www.techtarget.com/searchstorage/definition/RAID-controller

[3] Disk mirroring, https://www.techtarget.com/searchstorage/definition/disk-mirroring

[4] SAS vs. SATA, https://www.diffen.com/difference/SATA_vs_Serial_Attached_SCSI

[5] MariaDB vs MySQL – Key Differences, Pros and Cons, and More, https://www.hostinger.com/tutorials/mariadb-vs-mysql

[6] Tag Cloud Storage Comparison: Nextcloud vs. OwnCloud vs. Seafile, https://www.hongkiat.com/blog/self-hosted-cloud-storage-nextcloud-owncloud-seafile

[7] System Properties Comparison MariaDB vs. MySQL vs. SQLite, https://db-engines.com/en/system/MariaDB%3BMySQL%3BSQLite

[8] Gitea vs GitLab, https://www.programmerhat.com/gitea-vs-gitlab

[9] Github vs Gitea: A Quick Guide, https://hailbytes.com/github-vs-gitea-a-quick-guide

[10] Wireguard vs OpenVPN: which is better?, https://cybernews.com/what-is-vpn/wireguard-vs-openvpn

[11] Softether vs OpenVPN – Which one is better?, https://www.vpnxd.com/softether-vs-openvpn-which-one-better

[12] Traefik vs NGINX, https://www.kubecost.com/kubernetes-devops-tools/traefik-vs-nginx

[13] What is OpenID Connect (OIDC)?, https://auth0.com/intro-to-iam/what-is-openid-connect-oidc

[14] The OAuth 2.0 Authorization Framework, https://datatracker.ietf.org/doc/html/rfc6749

# Ringraziamenti

Mi è doveroso dedicare questo spazio della mia tesi a tutte le persone che mi hanno supportato nel mio percorso di crescita universitaria e professionale.

Un sentito ringraziamento va al mio relatore prof Enrico Masala che mi ha seguito, con disponibilità e gentilezza, in ogni step della realizzazione dell'elaborato e il Politecnico di Torino per avermi offerto la possibilità di vivere una esperienza formativa e di crescita negli Stati Uniti per scrivere questa tesi.

La mia gratitudine va alla prof Marina Mondin e al prof Fred Daneshgaran, che hanno esercitato i ruoli di co-relatori ma anche di punto solido di riferimento durante la parte finale del mio percorso universitario. Avete fatto per me molto più di quello che dovevate e ve ne sarò per sempre grato.

Non posso non ringraziare le due persone che hanno più di tutti contribuito nella mia crescita e nel farmi diventare la persona che sono oggi: Mamma e Papà. Grazie per avermi sempre lasciato carta bianca in ogni scelta della mia vita e, nonostante ciò, aver supportato ogni mia decisione come fosse stata vostra e creduto in me. Mi avete sempre offerto la possibilità di poter sbagliare ma allo stesso tempo la soddisfazione di raggiungere gli obbiettivi che io stesso mi ponevo. Spero di avervi reso orgogliosi.

Ringrazio infinitamente la mia fidanzata Gioia che nel mio percorso è stata sempre pronta a supportarmi e sopportarmi, sembrerà una frase fatta ma la sento più vera che mai. Abbiamo tenuto duro in periodi difficili dettati dalla distanza ma siamo sempre stati capaci di recuperare la fatica di mesi di lontananza con gli attimi dietro la dolcezza di un abbraccio. Ringrazio anche la sua famiglia per avermi da sempre accolto con rispetto e amore.

Ci tengo a ringraziare Andrea, Marco, Federico, Checco, Matteo e tutte le persone con cui ho condiviso, anche temporaneamente, un tetto durante la mia esperienza universitaria. Sappiate che per me il vostro ruolo è stato fondamentale, in quanto avete tutti contribuito nel creare per me un ambiente ideale dove studiare, da una parte stimolante ma anche goliardico quando necessario. Ringrazio inoltre il gruppo, ad oggi, Berlino/Amsterdam/Bruxelles/Madrid/ Barcellona/CATANIA Flex per aver sempre creato momenti di felicità attraverso viaggi nelle varie mete, facendomi scoprire il piacere di viaggiare con tanta bella gente.

Ringrazio anche la mia amica Simona, per essere sempre stata disponibile a lunghe chiamate per parlare e sfogarci su qualunque cosa, il mio amico Giuliano col quale ho avuto, tra le tante cose, il piacere di condividere quotidianamente le nostre esperienze di scambio internazionale, il mio amico Jacopo, per aver trovato la pazienza e il piacere di intraprendere un viaggio per venirmi a trovare e concludere insieme l'esperienza americana e tutti i ragazzi del gruppo di LoL per avere trascorso momenti di divertimento e frustrazione su quel dannato gioco.

Ringrazio infine tutti gli zii, i cugini e i nonni (sia qua giù che lassù). In ogni occasione di riunione ho sempre sentito il supporto di tutti. In particolare, ringrazio la zia Maria per essere stata sempre disponibile e avermi sempre deliziato con le sue specialità e la Nonna Angela, che mi ha sempre accompagnato durante tutte le fasi della mia vita.