POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un prototipo di orchestratore RAN 5G

Relatori:
prof. Casetti Claudio Ettore
prof. Chiasserini Carla Fabiana

Candidato:
Glorioso Marco
matricola: s243809

TIM S.p.A.
Ing. Scarpina Salvatore

Anno accademico 2018-2019

Con affetto a mia madre, la donna che ha reso possibile tutto ciò, supportandomi non solo economicamente ma anche moralmente. Ti ringrazio per essermi stata accanto in ogni momento difficile della mia vita, di essere stata una guida e un modello a cui sempre potermi ispirare. Per tali motivi e per esprimere la mia infinita gratitudine sono lieto di dedicarti questo elaborato e il traguardo che rappresenta.

Grazie

Sommario

L'orchestrazione è attualmente uno degli argomenti centrali della rivoluzione tecnologica che sta vivendo la rete di accesso. L'orchestrazione di rete è un modello di architettura mirato all'automazione della rete di accesso e alla riduzione quanto più possibile dell'intervento umano nella sua gestione e manutenzione. L'orchestratore è quella componente software che ottimizza l'utilizzo delle risorse per poter erogare un servizio al client. Un ulteriore obiettivo dell'orchestrazione è ridurre il time-to-market, cioè il tempo che intercorre dall'ideazione di un prodotto/servizio alla sua effettiva commercializzazione. In questa ottica, l'orchestrazione guadagna un valore in più in quanto permette agli operatori di sperimentare nuovi servizi, testarli rapidamente e, se funzionano, monetizzarli e velocizzare in modo significativo il ciclo di innovazione.

Il lavoro di tesi analizza a fondo il modello architetturale dell'orchestrazione e identifica le aree dove trova maggiormente impiego, ma in particolare si concentra sull'impiego dell'orchestrazione nella rete di accesso. Risulta opportuno notare come già nelle reti LTE è presente un concetto che fa da precursore a quello di orchestrazione, tipico della futura infrastruttura di rete 5G. Questa peculiarità delle reti LTE prende il nome di SON, Self-Organizing Network. Esso è stato introdotto nell'ambito della rete di accesso radiomobile sin dalle prime release LTE ed è una prima risposta alle esigenze degli operatori di far fronte alla complessità di gestione dei processi di configurazione, di ottimizzazione e di affidabilità delle reti.

Dopo aver discusso del modello architetturale di orchestrazione e del suo impiego nella rete mobile, si presentano i possibili use-case derivati dall'impiego delle funzioni di orchestrazione. Il lavoro svolto ha permesso lo sviluppo di un prototipo di orchestratore RAN in grado di monitorare la rete di accesso, impiegando la tecnologia dei Minimization of Drive Test(MDT). Questa tecnologia permette di ridurre l'OPEX dei drive test sul territorio e di svolgere analisi più dinamiche, sia per quanto riguarda l'area geografica da analizzare, sia per l'intervallo di tempo durante il quale effettuare le misurazioni.

L'architettura sviluppata durante il lavoro di tesi coopera con il controllore RAN, che comunica con le celle di basso livello e con gli UE ad esse connessi. Quindi si è definita una comunicazione tra l'orchestratore e il controllore, sviluppando delle API REST e impiegando strumenti di validazione dei dati scambiati. Il sistema di orchestrazione sviluppato fornisce un'interfaccia grafica per un amministratore di rete, tramite questa è possibile inoltrare delle richieste al controllore RAN per eseguire delle funzionalità di alto livello. Le principali funzionalità sono: richiedere gli eNodeB Functions su un'area geografica e in base al tipo di tecnologia, richiedere le celle di un eNodeB e lanciare una campagna MDT sulle celle di un eNodeB per monitorare il loro funzionamento per un intervallo di tempo. L'orchestratore mette a disposizione dell'utente degli strumenti per monitorare la rete ad un livello più alto, distaccandosi dalla molteplicità e dalla varietà delle tecnologie impiegate nella rete. L'interfaccia Web consente di rilevare i cambiamenti della rete attraverso delle richieste inviate periodicamente al controllore per mantenere l'interfaccia sempre aggiornata. Infine i valori delle misurazioni ricevute dal controllore sono raccolti e rappresentati graficamente, in questo modo è possibile analizzare il funzionamento delle celle tramite gli UE impiegati sulla rete. Il sistema realizzato supporta due valori di misurazione RSRP e RSRQ.

Questo lavoro di tesi si pone come prima risposta all'esigenza di impiegare funzioni di orchestrazione nella rete di accesso. Infatti, con l'avvento delle tecnologie di rete di quinta generazione, assisteremo a procedure di densificazione della rete. Pertanto il prototipo sviluppato svolge un primo use case di monitoraggio, permettendo di raccogliere dati di funzionamento della rete di accesso composta da un numero elevato di celle e mette a disposizione i dati raccolti per ottimizzazioni e predizioni future, impiegando possibili tecnologie di AI e ML.

Ringraziamenti

Desidero menzionare tutti coloro che mi hanno aiutato nella stesura con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore contenuto in questo lavoro.
Ringrazio anzitutto il mio relatore il professore Claudio Ettore Casetti e la mia co-relatrice la professoressa Carla Fabiana Chiasserini che mi hanno permesso di svolgere questo lavoro di tesi presso l'azienda TIM e per il supporto continuo durante la stesura dell'elaborato.
Ringrazio successivamente il mio tutor aziendale l'ingegnere Salvatore Scarpina, che ha messo a disposizione il suo tempo e le sue conoscenze per svolgere al meglio questo progetto. Inoltre gli sono riconoscente per i confronti costruttivi e per avermi spronato ad ampliare le mie conoscenze e la mia visione dell'ambiente lavorativo e gli porgo le mie scusa per le occasioni in cui sono stato testardo e poco disponibile all'ascolto.
Un ringraziamento particolare va a mia zia nonché professoressa Santina Tumminello che mi ha guidato, consigliato e supportato in tutta la stesura di questo elaborato, dedicandomi il suo tempo e dimostrandomi tutta la sua professionalità.
Ugualmente ringrazio il mio attuale coinquilino nonché amico Corrado Strano per avermi supportato con pazienza ed interesse.
Un pensiero caro va alla mia famiglia che mi ha sempre incitato e incoraggiato a fare bene e che mi dimostrato sempre di interessarsi alla mia carriera accademica e ai miei progetti. In particolare ringrazio mia sorella che ha sempre creduto in me e che mi sempre dimostrato la sua stima e ammirazione.
Concludo ricordando gli amici e i colleghi che mi hanno accompagnato in questi due anni di laurea magistrale e con cui ho avuto modo di trascorrere tanti momenti felici. In particolare ringrazio Giuseppe Solaro che mi ha ospitato all'inizio della mia permanenza a Torino, dimostrandomi la sua disponibilità e amicizia e inoltre ci siamo sempre sostenuti a vicenda sia durante i momenti più faticosi e impegnativi che hanno caratterizzato il nostro percorso sia nei momenti di gioia e soddisfazione per i traguardi raggiunti.

Con l'augurio che mio padre possa essere fiero di me
e di quello che sono diventato...
Grazie

Indice

Elenco delle figure

Acronimi

AI Artificial Intelligence.

CORS Cross Origin Resource Sharing.

CP Control-Plane.

CU Central Unit.

DU Distribuited Unit.

FTTH Fiber-To-The-Home.

HTML HyperText Markup Language.

Http HyperText Transfer Protocol.

IoT Internet of Things.

ITU International Telecommunication Union.

JSON JavaScript Object Notation.

JWT JSON Web Token.

KPI Key Performance Indicators.

LTE Long Term Evolution.

ML Machine Learning.

MP Management-Plane.

NFVI Network Functions Virtualization Infrastructure.

O&M Operations & Maintenance.

SON Self Organizing Network.

UP User-Plane.

Glossario

API Application Programming Interface. In informatica, in un programma, si intende un insieme di procedure atte all'espletamento di un dato compito; spesso tale termine designa le librerie software di un linguaggio di programmazione.

binding In informatica il binding è il processo tramite cui viene effettuato il collegamento fra una entità di un software ed il suo corrispettivo valore.

CFS Customer Facing Services, sono i requisiti definiti col Cliente nel BSS per un servizio da erogare.

Client entità che può utilizzare risorse o servizi di un fornitore.

closed-loop è un insieme di dispositivi meccanici, elettronici o software che regola automaticamente un processo senza interazione umana.

data-model È un modello astratto che organizza elementi di dati e standardizza il loro rapporto reciproco. Ad esempio, un modello di dati può specificare che l'elemento di dati che rappresenta un'auto sia composto da un numero di altri elementi che, a loro volta, rappresentano il colore e le dimensioni dell'auto.

DMZ Demilitarized zone, in sicurezza informatica, è un'area in cui sia il traffico WAN che quello LAN sono fortemente limitati e controllati.

Document data model dei Document Database, utilizzano una struttura simile al JSON (JavaScript Object Notation), un formato diffuso tra i sviluppatori Software e ogni Document può contenere diversi campi.

eNodeB Evolved Node B, è l'elemento E-UTRA dell' LTE che è l'evoluzione dell'elemento Nodo B in UTRA di UMTS . È l'hardware connesso alla rete di telefonia mobile che comunica direttamente in modalità wireless con i dispositivi mobili(UE).

HMAC keyed-Hash Message Authentication Code, è una modalità per l'autenticazione di messaggi (message authentication code) basata su una funzione di hash, utilizzata in diverse applicazioni legate alla sicurezza informatica.

MDT Minimization of Drive Tests, l'idea che si introduce con gli MDT è quella di utilizzare un insieme di dispositivi mobili attivi per monitorare la rete.

NFV Network Functions Virtualization, virtualizzare intere funzioni dei nodi di rete come blocchi elementari che possono essere interconnessi per implementare servizi di comunicazione.

OAM Operations Administration and Management,insieme dei processi delle attività e degli standard coinvolti nel funzionamento e nella gestione del sistema.

OOP Object Oriented Programming, è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri. in grado di interagire gli uni con gli altri.

OPEX OPerating EXpense, ovvero spesa operativa è il costo necessario per gestire un prodotto, un business od un sistema.

provisioning Il Provisioning implica il processo di configurazione e dotazione di una rete per consentirgli di fornire nuovi servizi ai propri utenti.

QoS Quality of Service, è utilizzato per indicare i parametri, gli strumenti e le tecniche per ottenere la desiderata qualità del servizio dalla rete.

RAN Radio Access Network, è una parte del sistema di telecomunicazione mobile. Implementa le tecnologie di accesso radio e risiede tra i dispositivi come i telefoni cellulare, i computer o qualsiasi macchina controllata a distanza e fornisce la connessione con la sua Core Network CN.

RFS Resource Facing Services, è una funzione di rete fornita da uno o più dispositivi di rete che possono essere fisici o virtualizzat.

RSRP Reference Signals Received Power, è una misura del livello di potenza ricevuto in una rete di celle LTE.

RSRQ Reference Signal Received Quality, è un tipo di misura che indica la qualità del segnale di riferimento ricevuto. La misura RSRQ fornisce informazioni aggiuntive quando RSRP non è sufficiente per effettuare una decisione.

SDN Software-Defined Networking, suggerisce di centralizzare l'intelligenza di rete in un componente separato, disassociando il processo di forwarding dei pacchetti (Data Plane) da quello di routing (Control Plane).

time-to-market il tempo che intercorre dall'ideazione di un prodotto/servizio alla sua effettiva commercializzazione.

TLS Transport Layer Security, è un protocollo crittografico usato nel campo delle telecomunicazioni e dell'informatica che permette una comunicazione sicura dalla sorgente al destinatario (end-to-end) su reti TCP/IP fornendo autenticazione, integrità dei dati e confidenzialità operando al di sopra del livello di trasporto.

TMSI Temporary Mobile Subscriber Identity, è l'identità che viene più comunemente inviata tra il cellulare e la rete.

UE User Equipment, si usa per indicare qualunque dispositivo utilizzato direttamente da un utente.

UI User Interface, ovvero ciò che si frappone tra una macchina e un utente, consentendone l'interazione reciproca.

user-experience esperienza d'uso, s'intende ciò che una persona prova quando utilizza un prodotto, un sistema o un servizio.

Capitolo 1

Introduzione generale

1.1 Esigenze attuali

Oggi l'industria delle telecomunicazioni senza dubbio attraversa un momento critico. Nel 2017 essa contava 4.8 miliardi di abbonati mobili e cioè il 65% della popolazione mondiale. Questo dato nel 2018 ha conosciuto un tasso di crescita decisamente inferiore confrontato con il volume e la varietà del traffico prodotto dai dispositivi mobili.
Quali le conseguenze?
La crescita dei guadagni si è arrestata, ma di contro le esigenze degli operatori e dei consumatori sono cresciuti. Basti pensare al numero crescente di dispositivi connessi che compongo IoT, all'introduzione di applicazioni a bassa latenza e ad altre che richiedono alta affidabilità e velocità. Gli operatori mobili sono perciò sotto pressione per rispondere a questa richiesta continua di risorse, nel tentativo di mantenere da una parte i costi contenuti e dall'altra nel tentativo di riuscire a fornire nuovi servizi competitivi. Senza dubbio si rende necessario un cambiamento, un'innovazione radicale della rete. Non è più possibile usare gli strumenti umani tradizionali. Occorre una rete intelligente, che sappia sfruttare le nuove tecnologie di intelligenza artificiale, per automatizzare e schierare nuove funzioni di rete al bisogno. Ci troviamo di fronte all'avvento della quinta generazione di tecnologie e standard di rete nota con il nome di 5G.

1.2 Dimensioni del 5G

Per inquadrare al meglio questa nuova rete rispetto alle reti precedenti possiamo dire che il mondo attuale 4G è ad una sola dimensione, quella della velocità di trasferimento. Il 5G sarà invece un mondo a più dimensioni, che includerà la

latenza, la velocità, l'affidabilità e l'energia. All'interno di esso ci si potrà muovere in base alle esigenze delle applicazioni, rispettando però i vincoli imposti dagli organi di standardizzazione. Il triangolo proposto dall'ITU ci aiuta a visualizzare le novità del 5G. Questo triangolo è uno schema che definisce tre principali modalità di funzionamento del sistema che massimizzano alcuni parametri prestazionali non compatibili tra loro.

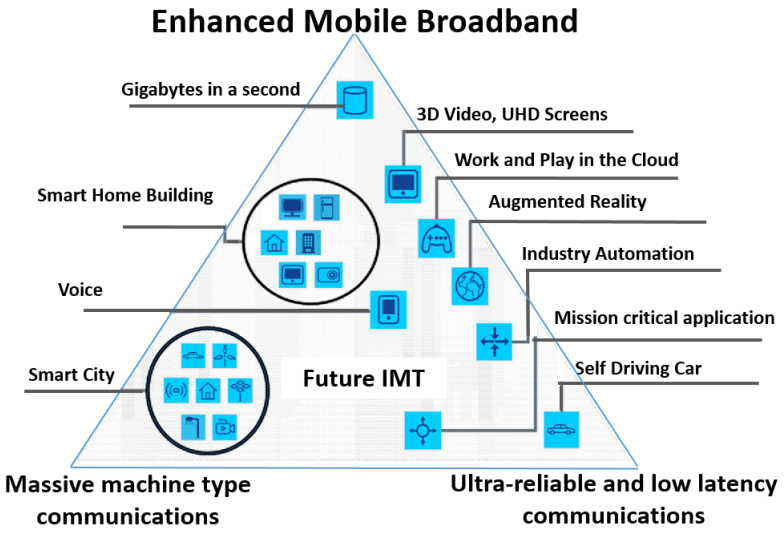

Figura 1.1. Triangolo dimensionale ITU

Analizziamo le dimensioni fondamentali di questa nuova tecnologia.
Le reti 5G consentiranno una velocità del collegamento dati da 1 a 6 Gigabit al secondo, per più dispositivi in una stessa cella, attualmente invece riusciamo a garantire al massimo 1 Gigabit al secondo per terminali di fascia molto alta. In termini di ritardo, la tecnologia 5G consentirà di scendere fino ad alcuni millisecondi nella comunicazione tra dispositivo, rete esterna e ritorno, paragonabile agli attuali collegamenti in fibra FTTH, un notevole miglioramento rispetto alle decine di millisecondi delle attuali reti 4G. In termini di affidabilità, il 5G per la prima volta fornirà delle soluzioni tecniche in grado di garantire in maniera quasi deterministica la consegna dell'informazione entro limiti di tempo stabiliti.

1.3 La struttura del 5G

Per definire al meglio l'argomento, oggetto di questa tesi, si riporta un'immagine dal white paper pubblicato dall'O-RAN Alliance nell'ottobre del 2018 [1] intitolato "Towards an Open and Smart RAN", che ci aiuta a identificare la struttura che si sta delineando per la quinta generazione di rete.

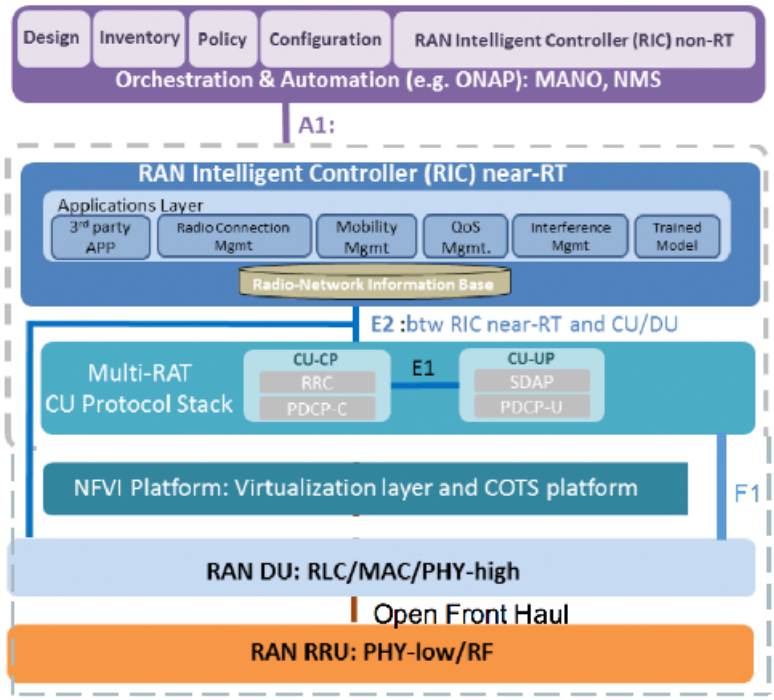

Figura 1.2. O-RAN Alliance Reference Architetture

Nel core della rete, abbiamo assistito a notevoli cambiamenti con l'avvento di SDN e di NFV, cambiamenti che hanno permesso di rendere la rete più agile e dinamica. Il RAN, invece, è rimasto quasi invariato nonostante sia la parte dove si concentrano i costi di mantenimento e gestione della rete. Con l'avvento delle nuove tecnologie di rete l'O-RAN Alliance propone una nuova struttura per l'infrastruttura RAN. Questa struttura si compone di due componenti fondamentali:

1. Un orchestratore di servizi e configurazioni di alto livello che gestisce il MP.

2. Un Controllore di più basso livello che gestisce CP e UP ed espone due interfacce fondamentali A1 verso l'orchestratore ed E2 e F1 verso i livelli fisici sottostanti.

Lo strato di Orchestrazione dovrà assolvere molte funzioni fondamentali nell'avvento della quinta generazione di tecnologie di rete. Tra le più importanti menzioniamo:

- Gestione della rete virtualizzata e non virtualizzata.
- Load balancing.
- Monitoraggio dello stato della rete e delle interferenze per UE.
- Network Slicing, configurazione della infrastuttura di rete al fine di fornire servizi con QoS.
- Utilizzo di algoritmi di *machine learning* e supporto al *training* per i modelli dei livelli sottostanti dove saranno impiegati altri algoritmi di AI.
- Allocazione e ottimizzazione delle risorse di rete.
- Rilevamento e gestione dei guasti nell'infrastruttura di rete.

1.4 Le Finalità della tesi

Le finalità di questa tesi sono molteplici.
In primis definire il concetto di orchestrazione, un pattern di progettazione presente in molti ambienti tecnologici odierni. In seguito illustrare le aree dove l'orchestrazione trova il suo impiego, concentrandosi sulle sue applicazioni di rete nelle attuali tecnologie e in quelle in fase di sviluppo come il 5G. Poi si proseguirà ad esaminare i dettagli e le scelte architetturali che riguardano l'implementazione di un modello di Orchestratore RAN svolto in questi mesi di lavoro. Successivamente analizzare l'interfaccia verso il Controllore, al fine di ottenere uno use case di monitoraggio dello stato della rete e delle interferenze per UE. Infine, valutare le performance di quest'implementazione di orchestratore con la stesura delle opportune conclusioni.

Capitolo 2

Stato dell'arte

2.1 Orchestrazione

L'orchestrazione è attualmente uno degli argomenti centrali della rivoluzione tecnologica che sta vivendo la rete di accesso. L'orchestrazione di rete è un modello di architettura mirato all'automazione della rete di accesso e alla riduzione quanto più possibile dell'intervento umano nella sua gestione e manutenzione. Per comprendere a fondo il concetto di orchestrazione è necessario analizzare la differenza fra "automazione" e "orchestrazione". La prima è circoscritta ad attività parzialmente autonome a basso livello, si tratta di singoli *task*, semplici e isolati. Il termine orchestrazione, viene invece, utilizzato per descrivere l'automazione delle attività di molti task che hanno molte dipendenze tra loro.
Tutte le attuali definizioni di orchestrazione includono l'idea di selezionare automaticamente le risorse per soddisfare le richieste dei *Client*. Il termine Client rappresenta qualsiasi entità che può utilizzare risorse o servizi di un fornitore. Un'attività di orchestrazione può anche utilizzare risorse e servizi per i propri scopi. L'orchestratore è quindi quella componente software che ottimizza l'utilizzo delle risorse per poter erogare un servizio al Client. Ma attenzione a considerare l'orchestrazione come un'unica componente, piuttosto la si può considerare come un pattern di progettazione che si ripete a vari livelli, ad esempio nella rete di accesso potrebbero indicarsi le seguenti componenti di orchestrazione: una per la gestione dei servizi, un'altra per la gestione dell'accesso radio, un'altra ancora per la gestione della core network, una componente per la gestione della rete virtualizzata ed altre ancora. Tutte queste componenti hanno in comune l'obiettivo di erogare *service business* e il dover ottimizzare l'utilizzo delle risorse a loro affidate. Per comprendere ulteriormente questo pattern di progettazione, poniamoci nella situazione in cui l'orchestratore *higher-level* è incaricato ad erogare un servizio

complesso. Questo servizio viene scomposto in vari processi che sono demandati, attraverso gli *endpoint*, ad altri orchestratori subordinati al precedente. Ognuno di questi orchestratori secondari coordina dei processi su alcune risorse sottostanti per completare il processo assegnatogli. L'orchestratore principale non ha conoscenza e visione dei compiti svolti dagli orchestratori subordinati. È necessario soffermarsi sul processo *end-to-end* che si realizza tra l'orchestratore *higher-level*, gli orchestratori subordinati e le risorse sottostanti. Questo processo si identifica con **l'erogazione di un servizio** per gli orchestratori subordinati e come **la configurazione di una risorsa complessa** per l'orchestratore principale. Quest'ultimo, infatti lavora a un livello superiore con risorse più complesse per erogare il servizo al Cliente secondo il modello di orchestrazione illustrato precendemente. Un ulteriore obiettivo dell'orchestrazione è ridurre il **time-to-market**, cioè il tempo che intercorre dall'ideazione di un prodotto/servizio alla sua effettiva commercializzazione. In questa ottica, l'orchestrazione guadagna un valore in più in quanto permette agli operatori di sperimentare nuovi servizi, testarli rapidamente e, se funzionano, monetizzarli e velocizzare in modo significativo il ciclo di innovazione. Sorge, quindi, l'esigenza di una traduzione efficiente dei *Business Model* e delle esigenze degli utenti in risorse di rete, molto dinamiche, in grado di essere combinate velocemente tra loro.

La seguente immagine illustra lo schema di un'architettura d'orchestrazione mirata alla traduzione dei business model nella rete di accesso, dove sono presenti diverse tecnologie di rete tradizionali e virtualizzate.

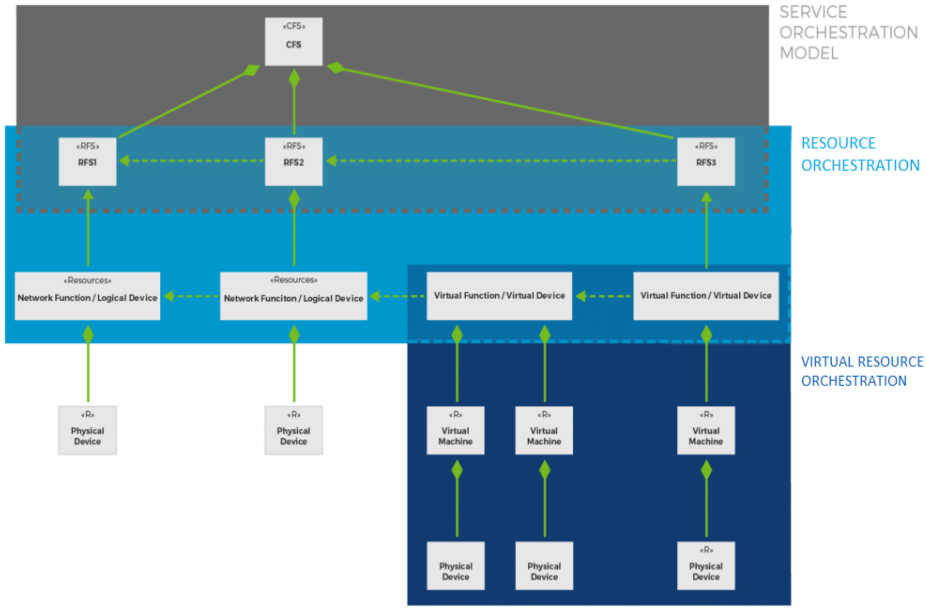

Figura 2.1. Architettura di orchestrazione per un Business Model.

L'immagine e le nozioni seguenti, sono tratte dall'articolo di Lukasz Mendyk intitolato:"What does orchestration really mean?" [2]. Il significato delle sigle CFS e RFS presenti nella figura 2.1 sono:

- **CFS, Customer Facing Services**, sono i requisiti definiti col cliente nel BSS per un servizio da erogare.
- **RFS, Resource Facing Services**, è una funzione di rete fornita da uno o più dispositivi di rete che possono essere fisici o virtualizzati.

Questa classificazione si ritrova anche nel personale lavorativo di un'azienda. I CFS sono di competenza dei product managers che lavorano a contatto con i clienti e attestano la validità di nuovi servizi. Diversamente, gli RFS sono di competenza degli ingegneri e del personale tecnico, i quali definiscono le funzioni di rete e le aggiungono ad un catalogo. Analizzando l'immagine notiamo come l'orchestrazione sia distribuita su più livelli:

1. Orchestrazione del servizio guidata alla scomposizione CFS-RFS.

2. Orchestrazione delle risorse guidata alla scomposzione delle RFS in funzioni di rete realizzate da dispositivi fisici o virtualizzati.

3. Orchestrazione delle risorse virtuali (in basso a destra).

La componente di orchestrazione delle risorse virtuali riveste un ruolo importante nella nuova rete 5G, in quanto fornisce molta flessibilità alla rete virtualizzata e introduce uno strato di orchestrazione specifica per essa.
Tuttavia si torna a sottolineare che, considerare l'orchestrazione come un unico strato è inesatto, è preferibile immaginare una topologia con più livelli di orchestrazione, che permetta nel complesso la traduzione da un business model desiderato a un set di risorse di rete fisiche e virtuali, capaci di erogare il servizio di alto livello con i requisiti prestazionali voluti.

2.2 Aree di orchestrazione

Sono diverse le aree dove l'orchestrazione trova impiego e si ritiene opportuno fornire una panoramica di quelle più importanti in vista anche dell'avvento delle tecnologie di rete di quinta generazione.

- L'orchestrazione è particolarmente importante in un ambiente **multicloud**, in cui le applicazioni e i servizi vengono forniti da più origini. OpenStack, un framework software open source per realizzare e gestire un'infrastruttura cloud pubblica e privata, fornisce anche un servizio di orchestrazione dell'infrastruttura costruita.

- I **Container** rappresentano la nuova area dove l'orchestrazione è applicata. I Containers sono la nuova via per fare *deploy e run* di applicazioni distribuite senza richiedere l'uso di una *virtual machine* dedicata. L'orchestrazione gestisce come questi cointainers sono creati, aggiornati e messi in funzione, gestisce anche come connetterli per realizzare applicazioni più complesse. Kubernetes è la piattaforma open source di orchestrazione che gestisce applicazioni multi-containers.

- L'orchestrazione ha un ruolo fondamentale nelle reti che supportano le nuove tecnologie note col nome SDN e NFV, infatti l'orchestrazione della rete consente a un controller SDN, tramite API, di eseguire il *provisioning*, l'aggiornamento e la gestione delle risorse di elaborazione, necessarie per fornire un'applicazione o un servizio.

2.3 Use case di orchestrazione di rete

In questa sezione, si citeranno gli use cases più importanti derivanti dall'impiego dell'orchestrazione di rete e dopo si procederà ad analizzare singolarmente alcuni di essi.

- Ottimizzare l'utilizzo delle risorse di rete.
- Automatizzare la configurazione delle risorse di rete.
- Monitoring e relative ottimizzazioni.
- Gestione del Network Slicing e miglioramento della qualità del servizio.
- Fault-tolerance capacità di rimediare ai guasti di vari servizi con particolari caratteristiche di QoS.
- Automatizzazione del processo di verifica del QoS.
- Potenziamento della sicurezza attraverso il blocco o il reindirizzamento del traffico secondo regole di software defined network.

Ottimizzare l'utilizzo delle risorse di rete. Data la complessità delle reti odierne e il crescere del volume e della varietà del traffico mobile, i tradizionali metodi umani di manutenzione e ottimizzazione della rete non sono più sufficienti. Le reti necessitano di un livello di orchestrazione tale che, tramite tecniche di autoapprendimento come Machine Learning e Artificial Intelligence, sia in grado di automatizzare la rete per renderla autonoma e ridurre l'OPEX. Si desidera ottimizzare l'utilizzo delle risorse per massimizzare il profitto derivante dall'acquisto delle risorse stesse. Pertanto è necessario un approccio "reattivo" basato sull'adeguamento della rete alle condizioni variabili di traffico. Inoltre è opportuno impiegare algoritmi di tipo "predittivo" in grado di stimare le variazioni di traffico, su periodi futuri e di abilitare la messa in campo di contromisure atte a prevenire degradi del funzionamento della rete. Questo use case è molto importante, lo dimostra il fatto di essere ampiamente trattato dall'O-RAN Alliance nel documento pubblicato [1]: "Towards an Open and Smart RAN". L'O-RAN Alliance riassume la necessità di avere l'orchestrazione di rete con la parola *Intelligence*. Inoltre l'O-RAN si pone come guida dell'industria per realizzare una scomposizione architetturale del RAN, all'interno della quale trova posto l'orchestrazione. In aggiunta l'O-RAN esplicita l'esigenza d'introdurre strumenti d'intelligenza artificiale in vari livelli dell'architettura per eseguire l'allocazione, la gestione e l'utilizzo delle risorse in maniera ottimale.

Automatizzare la configurazione delle risorse di rete. Questo use case consiste nell'usufruire di nuovi protocolli per introdurre una maggiore automazione della configurazione dei dispositivi di rete. Esempi sono la configurazione di un'interfaccia switch, del routing ip attraverso il protocollo OpenFlow o NetConf. OpenFlow è un protocollo che consente ad un sistema centralizzato di indicare agli switch di rete come fare forwarding dei pacchetti. In una rete convenzionale, ogni switch ha un software proprietario che gestisce il suo funzionamento. Con OpenFlow invece, le regole d'instradamento dei pacchetti sono decise dal un controllore centralizzato, in modo che la rete possa essere programmata indipendentemente dai singoli switch.

Monitoring e relative ottimizzazioni. In questo use case si riassume la volontà di voler raccogliere informazioni sullo stato della rete tramite le risorse che la compongono, analizzarle e impiegarle per decisioni di gestione future. Questo implica che la nostra infrastruttura di rete dovrà fornire il supporto per i Data Collector. I dati raccolti con questi potrebbero costituire un'ampia base di dati su cui poter addestrare tecnologie di AI e ML al fine di ottenere classificazioni e previsioni future. Un'altra tecnologia che potrebbe contribuire a questo use case è costituita dai Minimization of Drive Tests MDT. L'idea che si introduce con gli MDT, che è ampiamente discussa nella relazione pubblicata da Daniel Baumann [3], è quella di utilizzare un insieme di dispositivi mobili attivi per monitorare la rete. Ogni dispositivo diventa uno strumento di misura, in grado di inviare dati all'operatore della rete mobile. Al momento l'OAM, che è l'insieme dei processi delle attività e degli standard coinvolti nel funzionamento e nella gestione del sistema, è l'incaricato a lanciare una campagna MDT contattando i relativi RAN di rete. Risulta però facile immaginare un orchestratore di servizi di alto livello delegato a svolgere questo compito. Perciò dall'orchestrazione di servizi di rete che impieghi gli MDT per fare monitoraggio è possibile delineare ulteriori use case. Questi sono ampiamente documentati in un documento pubblicato dal 3GPP nel 2009 [4] intitolato: "Study on Minimization of drive-tests in Next Generation Networks".
Si procede ad elencare alcuni di questi:

- **Ottimizzazione della copertura**
  Un dato che può essere facilmente riscontrato dall'utente perchè incide sulla *user-experience* e sull'utilizzo del nostro dispositivo.
- **Ottimizzazione della mobilità**
  Consiste nell'ottimizzare quello che avviene quando i dispositivi mobili si muovono da una cella all'altra.
- **Ottimizzazione della capacità**
  Mira a rendere il traffico nella rete dell'operatore uniforme e identifica anche quali zone sono congestionate.

- **Parametrizzazione per i canali condivisi**
  Mira ad ottimizzare alcuni parametri dei canali condivisi, opera in primis su accessi random, paging e canali broadcast che tutt'ora non sono ottimizzati.

- **Verifica dei servizi in termini di QoS**
  Valori di *data rate, delay, interrupts*, pacchetti persi e tempo di risposta sono responsabili della QoS di un servizio. Di solito la QoS è misurata in KPI. In base alla lista di misurazioni configurabili per un MDT è possibile avere una stima dei valori di diverse grandezze che possono incidere in termini di QoS.

Gestione del Network Slicing e miglioramento della qualità del servizio. Qui si opera in termini di business models e servizi erogati richiesti dal cliente. Per introdurre il concetto di Network Slicing bisogna prima comprendere che l'infrastruttura di rete di ultima generazione è composta da molte componenti virtualizzate. Infatti, con l'avvento di SDN e NFV, che sono distribuiti commercialmente per offrire una maggiore flessibilità di rete, le tradizionali componenti di rete vengono sostituite con elementi virtuali che possono essere combinati tra loro (anche tramite software). Sulla base del documento pubblicato dall'IETF intitolato "Network Slicing Architecture" [5] si andranno a illustrare alcuni elementi fondamentali per comprendere questo use case. Dalla concatenazione di risorse di rete virtuali e fisiche (network, compute, storage) si crea una **Resource Slice**. Dall'insieme di Resource Slices, *virtual network functions* e *network functions* si ha una **Network Slice** che è una rete virtuale, che espone delle API e quindi è programmabile per i vari scopi. Ecco che entra in gioco l'orchestrazione che deve essere in grado di selezionare una Network Slice adeguata prestazionalmente al servizio da erogare. Una volta selezionata, questa prenderà il nome di **Network Slice Instance** e fornirà i requisiti di rete che sono richiesti dal servizio di orchestrazione. Una Network Slice Instance potrebbe essere condivisa fra più servizi. Detto ciò, ci si aspetta che i nuovi livelli di orchestrazione siano abilitati alla **Network Slicing**, cioè alla capacità di saper creare più reti virtuali (network slice) sulla stessa infrastruttura fisica. Si comprende che la creazione di una network slice non ha vincoli di requisiti fisici ma dipende dai requisiti logici e prestazionali di cui necessitano il servizio o le varie applicazioni di nuova generazione come: la casa intelligente, Internet of Things (IoT), l'automobile connessa o la rete energetica intelligente.

2.4 Orchestrazione nelle reti mobili

Anche nelle reti mobili viene introdotto il concetto di orchestrazione. Risulta opportuno notare come già nelle reti LTE è presente un concetto che fa da precursore a quello di orchestrazione tipico della futura infrastruttura di rete 5G. Questa peculiarità delle reti LTE prende il nome di **SON, Self-Organizing Network**. Esso

è stato introdotto nell'ambito della rete di accesso radiomobile sin dalle prime release LTE ed è una prima risposta alle esigenze degli operatori di far fronte alla complessità di gestione dei processi di configurazione, di ottimizzazione e di affidabilità delle reti.

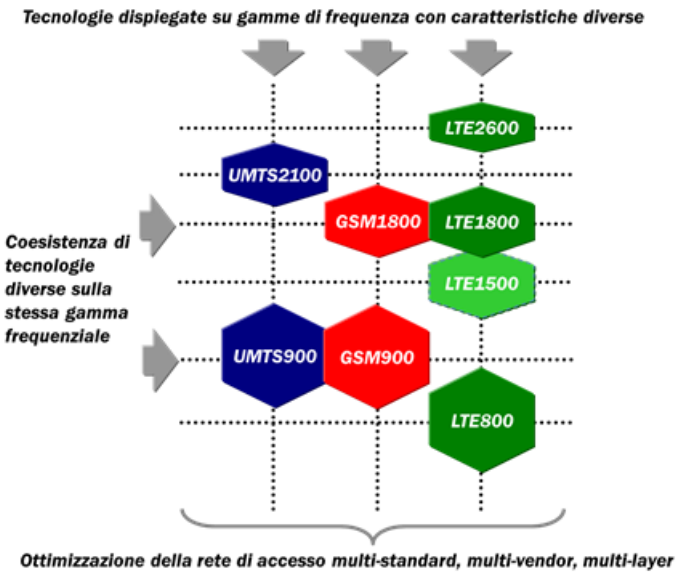

Figura 2.2. Varietà delle tecnologie di accesso radio

Come si può notare nell'immagine 2.2, l'impiego di diverse tecnologie come UMTS, GSM e LTE introduce un fattore di complessità nella rete mobile. Queste tecnologie differiscono ancora per aspetti implementativi e per le normative che il provider deve seguire per il loro impiego. In aggiunta la quantità dei dispositivi mobili connessi, la varietà del traffico e l'introduzione di nuove applicazioni con diversi requisiti di rete comporta una crescita della complessità nelle reti mobili. Con l'avvento del 4G, la tecnologia della rete di accesso radio mobile ha provato a dare una prima risposta alla crescente complessità della gestione delle reti, introducendo soluzioni in grado di agevolare i più comuni processi di configurazione e ottimizzazione. Alcuni esempi di tali processi sono: l'aggiornamento della topologia della rete, la riduzione dei problemi dovuti alla mobilità degli utenti, il bilanciamento del traffico e il miglioramento della copertura del segnale radio. Ciò si è tradotto nel rendere automatici alcuni processi, attraverso funzionalità distribuite nei nodi di rete, in grado di identificare una condizione critica del sistema e di adattare

autonomamente i parametri di controllo della rete stessa. Ma a questo approccio distribuito è seguito da un approccio centralizzato in grado di eseguire delle funzioni di automazione coordinate su più nodi di rete, anche di vendor diversi, però meno incisivo dal punto di vista delle funzionalità e delle tempistiche.

Un esempio importante che ci fa comprendere il percorso svolto nel campo delle tecnologie di accesso radio verso il concetto di orchestrazione attuale è il paradigma di TIM chiamato: "Open SON". Durante il 2015 TIM definì un'architettura di riferimento che fece un grosso passo avanti verso nuove prospettive di orchestrazione. L'evoluzione e i benefici prodotti da questa architettura sono trattai in un documento informativo della Tim [6] intitolato: "DigiRAN: il valore dell'automazione nell'accesso radio". L'architettura "Open SON" infatti prevedeva che le funzionalità di ottimizzazione e progettazione radio fossero strutturate in un'infrastruttura software che abilita due tipologie di processi *closed-loop*:

- **Automatic Closed Loop** per le funzionalità "basic" di ottimizzazione e configurazione di rete, completamente automatizzabili.
- **Human Closed Loop** per le funzionalità che richiedono un'analisi più avanzata, nelle quali l'azione degli specialisti radio è supportata da applicazioni sviluppate ad hoc, che garantiscono anche il controllo delle funzioni automatiche.

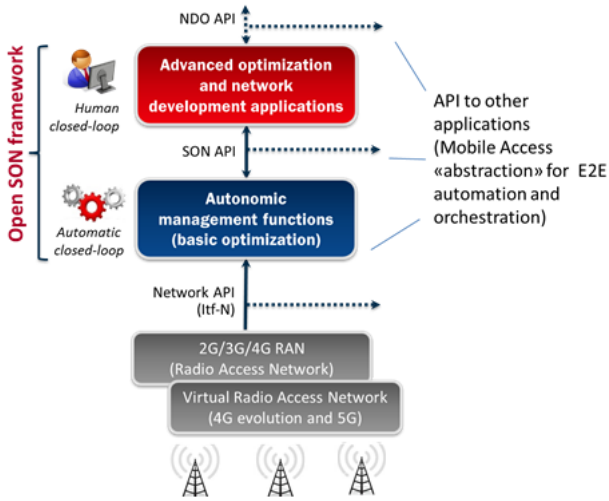

Figura 2.3. Architettura Open Son

Si può notare che in questa architettura è già presente un primo strato software di orchestrazione. Infatti i workgroups di TIM realizzarono questa infrastruttura col fine di raggiungere un maggior livello di automazione dove i processi lo permettessero e inoltre poter usufruire di applicazioni di monitoraggio e analisi della rete ad alto livello. Qui le componenti di accesso radio espongono delle API e queste sono fondamentali nell'ottica di voler raggiungere una maggiore flessibilità. Le API consentono a più sistemi SON di lavorare insieme e inoltre queste rappresentano la base per circuiti end-to-end programmabili.
Successivamente queste componenti di automazione, discusse precedentemente, sono state integrate come Virtual Network Functions all'interno delle infrastrutture NFV, questo comporta alcuni vantaggi come: assenza di attività dedicate di allocazione e configurazione dell'hardware, rapidità del processo di integrazione di nuove funzioni, abilitazione di paradigmi di automazione multi-dominio come il Network Slicing. La sfida successiva in cui Tim s'impegna prevede l'evoluzione verso la Virtual/Cloud RAN che consiste nel portare i paradigmi, appartenenti infrastruttura SON, di apertura, flessibilità, virtualizzazione e automazione a tutti i livelli della rete di accesso. La seguente immagine illustra l'evoluzione dell'infrastruttura OPEN SON:

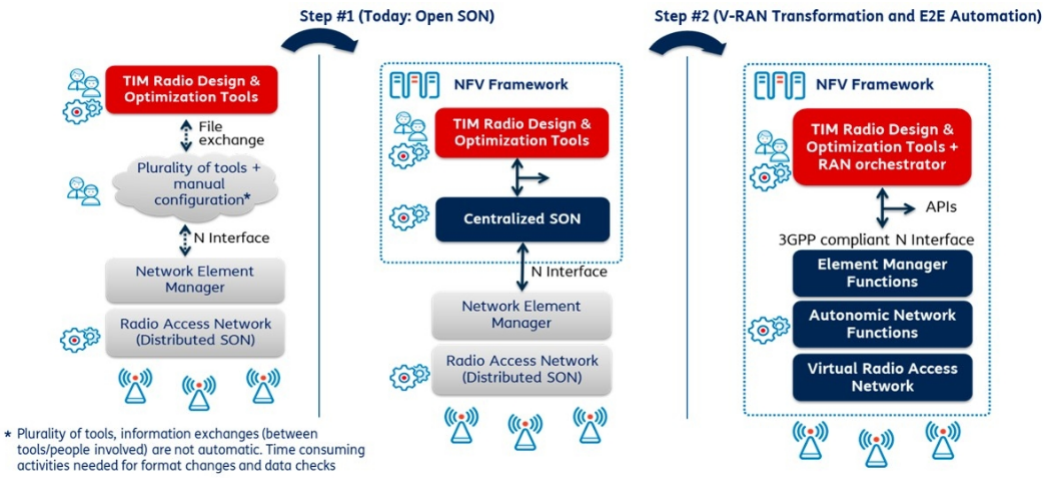

Figura 2.4. L'evoluzione dell'architettura Open Son

L'introduzione della virtual RAN (vRAN) prevede che parte delle componenti in banda base dei nodi BBU (Base Band Units) diventino funzioni virtualizzate integrabili su NFV, lasciando su HW fisico i moduli a radio frequenza, prossimi alle antenne RRU (Remote Radio Units) e una parte dei moduli che implementano i

layer protocollari più bassi (livello fisico). L'introduzione della virtual RAN accresce la flessibilità e l'efficienza operativa della rete di accesso. Nella rete di accesso tradizionale, infatti, le risorse elaborative sono legate ai singoli nodi, quindi ognuno di essi, deve essere dimensionato per fare fronte al proprio picco di traffico. Viceversa con l'introduzione della virtual RAN, si può ipotizzare che tutta la rete di accesso dell'operatore faccia parte di un unico *cluster* corrispondente ad un data center in cloud. Pertanto le risorse sono condivise tra tutti i nodi e, di conseguenza, possono essere allocate dinamicamente seguendo le variazioni di traffico, con un risparmio di risorse stimabile fino al 50%. La virtualizzazione, inoltre, consente di ottimizzare anche gli spazi e le risorse energetiche. Un'altra componente prevista, nell'evoluzione verso la completa digitalizzazione delle reti 5G, è il "SON-RAN-Orchestrator", che assicurerà il coordinamento tra le funzionalità interne al dominio (funzionalità SON e di design di rete), le altre funzionalità di O&M (CM, PM, TM, etc.) e gli orchestratori esterni al dominio (Service Orchestrator, NFV Orchestrator, etc.), astraendo tramite API di orchestrazione le risorse del dominio e le funzionalità di configurazione e ottimizzazione. L'evoluzione delle architetture di orchestrazione si svolgerà in un contesto nel qual gli elementi di rete accesso RAN dovranno evolvere in una logica "software-based" diventando maggiormente aperti e flessibili, superando i vincoli derivanti dal hardware e dalle archietture proprietarie e realizzando a pieno i paradigmi di apertura, flessibilità, virtualizzazione e automazione.

2.5 Orchestrazione nelle reti mobili 5G

Le nuove tecnologie di quinta generazione per le loro grandi potenzialità stanno suscitando l'interesse dei provider più importanti. La natura di queste potenzialità così elevate risiede nella capacità di sfruttare lo spettro di banda millimetrico, ovvero la porzione di frequenze radio corrispondenti alle onde elettromagnetiche con lunghezza d'onda dell'ordine del millimetro. Sorgono però altre problematiche, queste onde millimetriche così performanti hanno infatti una portata limitata e sono particolarmente soggette alle interferenze. Detto ciò, per garantire il trasferimento dei pacchetti dati tra due punti di una rete basata su banda millimetrica, sarà necessario che mittente e destinatario siano in linea di vista. Ossia, non devono esserci ostacoli fisici che possano schermare, anche parzialmente, le onde. Questa situazione porterà a procedure di **densificazione** della rete.
La densificazione corrisponderà a un numero molto elevato di cellule dispiegate: il 5G probabilmente avrà almeno cinque volte più cellule rispetto all'LTE e, nelle aree ad alto utilizzo, la cifra potrebbe essere 20 o più. Ciò solleva molte sfide nel modo in cui un numero così grande di elementi può essere costantemente ottimizzato per offrire le migliori prestazioni per qualsiasi applicazione. L'orchestrazione diventa

necessaria in una rete con così tanti elementi in gioco, sarà essenziale automatizzare la maggior parte dei processi di pianificazione e ottimizzazione di queste reti dense. Diventa difficile ipotizzare un dispiego di risorse umane per la gestione dei cambiamenti della rete o degli UE connessi, ma ancora più difficile risulta poter manipolare i servizi e le risorse al fine di fornire i diversi use case proposti nel 5G. Si comprende quindi che c'è la necessità di una modernizzazione nello strato di accesso radio, infatti a differenza della core network che ha conosciuto nuove tecnologie come SDN e NFV che hanno permesso di rendere la rete dinamica e virtualizzata, la parte di RAN è quasi sempre rimasta intatta. Mi sembra adeguato riportare e analizzare il lavoro che sta svolgendo l'O-Ran Alliance, in quanto ha molti workgroups aperti che stanno lavorando attivamente per formalizzare un RAN più autonomo e intelligente, introducendo anche questa volta un layer di orchestrazione distribuito su più livelli. Da questa analisi si spera di riuscire a delineare meglio l'evoluzione che subirà l'infrastruttura di accesso radio per raggiungere una maggiore automazione e poter così fornire il supporto adeguato alle applicazioni native del 5G.

2.5.1 O-RAN Alliance

Fondata nel febbraio 2018 da AT&T, China Mobile, Deutsche Telekom, NTT DOCOMO e Orange, l'O-RAN Alliance è stata costituita come entità tedesca nell'agosto 2018. L'O-Ran Alliance si impegna a sviluppare reti di accesso radio in tutto il mondo. Si pone l'obiettivo di modernizzare il RAN e si pone come guida delle industrie per realizzare un ambiente di accesso radio abilitato al machine learning e AI, definendo una prima architettura e i suoi prerequisiti. Questa nuova architettura, che contiene una parte orchestrante distribuita chiamata "Ran Intelligent Controller", si mostra come l'evoluzione intelligente del RAN e dell'RNC. Per analizzare questa nuova architettura di rete necessitiamo di ricollegarci alla figura 1.2 del paragrafo 1.3. Notiamo la presenza di un layer superiore delegato a svolge molteplici funzioni di orchestrazione e automazione che rientrano in quelle precedentemente trattate in questo lavoro di tesi. Tra cui si ricorda: creazione di slicing, monitoring, configurazione e gestione delle risorse di rete. Una caratteristica importante da notare è la distribuzione in due parti del "Ran Intelligent Controller". La prima è "RAN Intelligent Controller(RIC) non-RT" ed è situata all'interno nel layer di più alto livello. Essa lavora con tempi superiori al secondo, infatti la sigla "non-RT" significa no *real-time*. La seconda è "RAN Intelligent Controller(RIC) near-RT" ed è situata nel layer sottostante e lavora nel range del secondo. Queste due parti sono collegate da un'interfaccia chiamata A1. Attraverso questa interfaccia, la componente non-RT invia e riceve dati aggiornati dal resto dell'infrastruttura sottostante. La stessa componente é delegata anche ad addestrare, sulla base dei dati ricevuti, i modelli per le istanze degli algoritmi di AI

che risiedono nella parte near-RT. Successivamente i modelli addestrati sono consegnati alla componente near-RT che li esegue a runtime.L'O-RAN Alliance desidera definire una nuova scomposizione del RAN, contenente più livelli di orchestrazione delegati a coordinare vari processi su risorse differenti. Inoltre la figura 1.2 può essere analizzata come una struttura piramidale nella quale scendendo aumenta il numero di tecnologie schierate e impiegate. Conseguentemente risulta valida la scelta di traslare i livelli di orchestrazione verso le parti superiori dell'architettura perché si riduce il numero di apparati da gestire e si pone l'orchestrazione a un più alto livello di astrazione dall'infrastruttura sottostante. In aggiunta questi layer di orchestrazione devono comunicare e interagire con le altre entità di orchestrazione presenti nella rete di accesso o in ambienti differenti come: la core network, la rete virtuale NFV, la gestione dei servizi ed etc. Concludendo il core dgli algoritmi di orchestrazione è sviluppato e gestito dagli operatori. Questi algoritmi danno la possibilità ai providers di modificare il comportamento del RAN a loro piacimento, utilizzando diversi Business Model, al fine di ottimizzare l'utilizzo della rete per i diversi use cases.
Maggiori informazioni e dettagli su quest'architettura sono stati divulgati dall'O-Ran Alliance nel white paper, pubblicato nell'ottobre del 2018 [1].
Oltre alla necessità di automazione e orchestrazione, l'O-RAN Alliance sottolinea l'esigenza di una **migrazione verso il Cloud** dell'infrastruttura di RAN. Ciò renderebbe possibile creare un'architettura più scalabile e resistente al volume di traffico in crescita e nello stesso tempo di contenere i costi economici. Ritenuta cruciale questa migrazione verso il cloud, l'O-RAN Alliance ha istituito un workgroups chiamato "The Cloudification and Orchestration Workgroup". Questo gruppo di lavoro cerca di guidare il disaccoppiamento del software RAN dalla piattaforma hardware sottostante. L'obiettivo è quello di creare una soluzione senza soluzione di continuità che miri ad aggiornamenti continui con una mescolanza flessibile di CU/DU di diversi fornitori per consentire l'innovazione nei livelli PHY e MAC. Inizialmente lavorerà per definire una NFVI e un'architettura di orchestrazione. Valuterà diversi modelli di implementazione, in base a come le parti costituenti l'architettura (RIC vicino-RT, RIC RT, CU-UP e DU) potranno essere distribuite in un'infrastruttura cloud. Inoltre saranno definiti i requisiti di CPU, memory, storage e rete che l'infrastruttura cloud dovrà supportare. Si prevedono collaborazioni con comunità open source e la possibilità di introdurre codice open source, come Openstack, Kubernetes e ONAP, dove possibile.

Capitolo 3

Architettura generale

3.1 Premessa

In questo capitolo si illustrerà la propria implementazione di un orchestratore di accesso radio realizzato per svolgere un use case di monitoraggio della rete, tramite campagne MDT attivate sugli UEs presenti nella rete. Questa componente di orchestrazione è prevista per lavorare insieme con un controllore di rete di più basso livello. Le due componenti espongono delle API con cui è possibile interagire e cooperano insieme al fine di fornire un primo livello di automazione della rete di accesso. Per costruire il livello di orchestrazione adeguato agli obiettivi, si è sviluppata un'architettura a microservizi di back-end implementata in Java. Per questa prima parte si è scelto di usufruire del supporto del framework Spring. Successivamente, invece, si è realizzata un'interfaccia web che rappresenta una console di controllo per un utente addetto alla rete. Questa è stata realizzata in Python con il supporto del framework Django. Tramite l'interfaccia è possibile iniziare una nuova campagna MDT, su un insieme di celle o su degli UE specifici e visualizzare le misure effettuate con i vari MDT raccolti. I parametri di misurazione scelti sono stati RSRP e RSRQ.

Figura 3.1. Componenti dell'architettura in generale

3.2 Architettura a microservizi

Lo stile architetturale a microservizi è un approccio allo sviluppo di una singola applicazione come insieme di piccoli servizi, ciascuno dei quali viene realizzato da un processo indipendente e può essere eseguito su una macchina fisicamente differente. Questi servizi comunicano tra loro con un meccanismo snello, spesso una HTTP API. L'avvento e la diffusione dell'infrastuttura Cloud, le pratiche di *continuous delivery*, l'organizzazione agile delle aziende in team di sviluppo piccoli ed autonomi (3-7 persone), sono i contesti in cui il modello dell'architettura a microservizi trova il suo largo impiego. Ogni microservizio si propone all'esterno come una *black-box*, infatti espone solo un Application Programming Interface (API), astraendo rispetto al dettaglio di come le funzionalità sono implementate e dallo specifico linguaggio o tecnologia utilizzati. Ciò mira a far sì che il cambiamento di ciascun microservizio non abbia impatto sugli altri. Questa scelta è in linea con le attività che sta svolgendo il gruppo di lavoro "The Cloudification and Orchestration Workgroup" dell'O-Ran Alliance discusse nel capitolo 2. Infatti in questa architettura ogni servizio è isolato, riducendo i tempi di *deployment* e rendendo il processo di *update* e *rollback* più snello e facile. Inoltre questa architettura ha una buona resilienza infatti quando un servizio non funziona non è automatico che tutto il sistema software smetta di funzionare. In molti casi è possibile isolare il problema ed intervenire mentre il resto del sistema continua a funzionare. Si può quindi comprendere come un'architettura basata sul modello dei microservizi sia la scelta più adatta per realizzare un sistema resiliente, dinamico e con tempi di deployment contenuti.
Si è scelto di utilizzare un tale modello di architettura per i molteplici vantaggi citati precedentemente ma anche perchè coincide con le linee guida indicate dalle prime organizzazioni incaricate di definire le nuove tecnologie di accesso radio, esempio è l'O-RAN Alliance. Questa sottolinea, nel white paper precedentemente citato [1], l'importanza di un sistema più dinamico, intelligente e migrabile sul cloud. Infatti di ogni singolo microservizio è possibile fare il deployment su una piattaforma cloud e questa migrazione è oscura al resto dell'architettura a microservizi. Inoltre questa architettura è predisposta a futuri aggiornamenti e miglioramenti, come può subirli un lavoro di tesi, anche nell'ottica di un lavoro di sviluppo futuro più ampio e completo. Il protocollo scelto per la comunicazione tra i vari microservizi è il protocollo Http, standard da molti anni per la comunicazione di applicazioni Web. Http è un protocollo sincrono e senza stato. Inoltre è il protocollo base nello sviluppo delle API RESTful ampiamente utilizzate in ambiente web e da grandi operatori come: Amazon, Google e molti altri.

Di seguito è riportata l'immagine dell'architettura a microservizi realizzata durante questi mesi di lavoro e in seguito si procederà ad analizzare ogni singolo servizio, nell'ordine cronologico con cui è entrato a far parte dell'architettura.

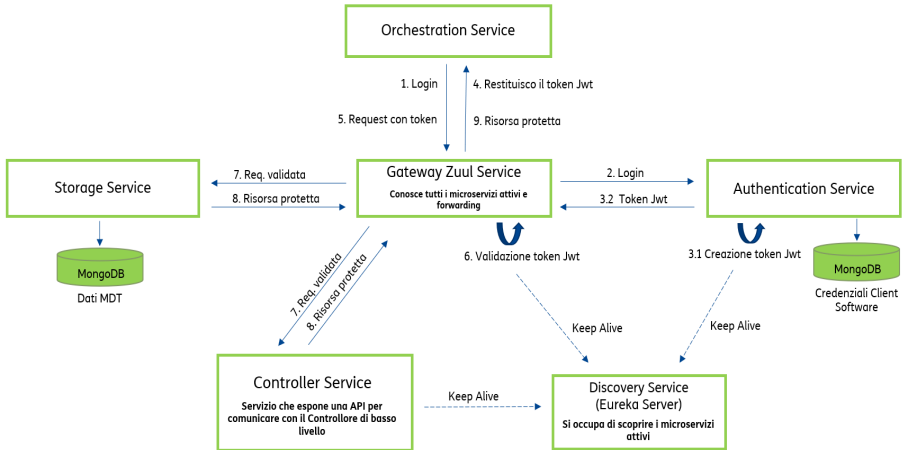

Figura 3.2. Architettura a microservizi realizzata con il framework Spring

3.2.1 Spring Framework

Nell'architettura sviluppata è stato utilizzato il framework Spring. La definizione di Spring più calzante è forse quella data dagli stessi autori, che lo definiscono: "un framework open source nato con l'intento di gestire la complessità nello sviluppo di applicazioni enterprise". Spring è un framework "leggero" e grazie alla sua architettura estremamente modulare è possibile utilizzarlo nella sua interezza o solo in parte. Ci sono diversi moduli che possono essere integrati nei progetti, la lista di questi è presente al seguente link: `https://spring.io/projects`. L'adozione di Spring in un progetto è molto semplice, può avvenire in maniera incrementale e senza sconvolgere l'architettura esistente. Questa sua peculiarità ne permette anche una facile integrazione con altri framework esistenti. Tutto ciò ha permesso di aggiungere più moduli nell'architettura durante le varie fasi di sviluppo. I moduli di Spring che sono stati utilizzati sono elencanti di seguito:

- **Spring Boot**, progettato per ridurre il tempo necessario per integrare il framework Spring in un progetto, fornisce una configurazione iniziale e automatica del framework e inoltre accende un'istanza del server Tomcat che funge da container per l'applicazione che si sta sviluppando, riducendo i tempi per creare una build e farne il deploy su un server esterno.

- **Spring Framework**, questo modulo, il cui nome potrebbe generare confusione, consiste nel core nel framework. Infatti questo consente alcune funzionalità base di Spring come: "dependency injection", "events", "resources", "validation", "data binding" e ecc.
- **Spring Cloud**, questo modulo semplifica lo sviluppo di un'architettura distribuita in stile microservizi implementando modelli che producono resilienza, affidabilità e coordinamento tra i vari servizi.
- **Spring Security**, questo modulo fornisce un potente sistema di autenticazione e autorizzazione, altamente personalizzabile. Come tutti gli altri moduli, Spring Security è davvero facile da utilizzare ed estendere per soddisfare requisiti personalizzati.

In tutti i servizi sviluppati sono stati inseriti i primi tre moduli, al fine di sfruttare al meglio il framework e iniziarlo ad usare nel minor tempo possibile. Il modulo "Spring Security" è stato integrato soltanto nell'"Authentication Service" e nel "Gatewway Zuul Service".
In particolare il modulo Spring Cloud realizza molteplici funzionalità che sono molto interessanti per un funzionamento efficiente e dinamico di un'architettura a microservizi. Le più importanti sono:

- Configurazione distribuita dei servizi
- Registrazione e scoperta dei servizi
- Routing
- Bilanciamento del carico
- Circuito Breakers
- Sistema di messaggi distribuiti
- Lock globali
- Chiamate service to service

Per realizzare al meglio alcune di queste funzionalità e rendere il sistema resiliente ai guasti, si è deciso di sviluppare due servizi separati nell'architettura a microservizi realizzta. Questi sono delegati a svolgere le funzioni di "Routing" e "Registrazione e scoperta dei servizi". Si sottolinea che le altre funzionalità precedentemente citate, incluse nel modulo Cloud, non sono state sviluppate e utilizzate in quanto ritenute eccessive per un'architettura di ridotte dimensioni come questa sviluppata in questo lavoro di tesi.

3.2.2 Storage Service

Il primo servizio realizzato è stato quello di Storage. Era necessaria una base di dati su cui l'architettura, tramite uno strato di API esposte dal servizio, potesse scrivere e leggere i dati. Il primo passo era valutare un database adeguato al nostro obiettivo. La prima domanda a cui bisognava rispondere era: "Database Sql o NoSql?"
Si comprese velocemente che un Database NoSql era la scelta migliore, per i seguenti requisiti:

- La struttura dei dati non era definita a priori, soltanto l'ordine di grandezza di questa poteva essere ipotizzato, circa un GigaByte.
- Una struttura dati vicina al software e di conseguenza alla programmazione orientata agli oggetti di Java.
- Alte prestazioni, questo è un requisito fondamentale per ogni applicazione di rete.
- Sistema key-value, vicino al formato JSON utilizzato nella maggior parte delle applicazioni client/server web per trasporto e store di dati.

Fatta questa scelta a discapito dei tradizionali database relazionali, bisognava valutare quale fosse il prodotto open-source più valido e aggiornato tra i database NoSql odierni. A tal fine si è deciso di considerare un articolo scritto dall'azienda produttrice di MongoDB e pubblicato nel 2015 [7] intitolato: "Top 5 Considerations When Evaluating NoSQL Databases". Questo documento chiarisce quali aspetti valutare nella scelta di un database NoSql. In primis bisogna comprendere che questa tipologia di database differisce dai databse Sql per il *data-model* e in base a questo possono essere raggruppati in 3 gruppi:

1. Document Database
2. Graph Database
3. Key-Value and Wide Column Database

Si è scelta la prima tipologia e cioè i "Document Database". Essi hanno questo nome perché a differenza delle *tuple* dei database relazionali, qui i dati sono salvati in strutture dati che prendono il nome di Document. Questi Document, in genere, utilizzano una struttura simile al JSON, un formato diffuso tra i sviluppatori software e ogni Document può contenere diversi campi. Queste caratteristiche implicano una flessibilità che può essere particolarmente utile per i dati non strutturati a priori e rende anche più facile l'evoluzione di un'applicazione durante lo

sviluppo, ad esempio aggiungendo o rimuovendo nuovi campi con molta semplicità. I Document Database sono di uso generale, utili per un'ampia varietà di applicazioni, grazie alla flessibilità del modello dati, alla possibilità di eseguire *query* su qualsiasi campo dei Document e per la mappatura naturale tra i Document e gli oggetti dei moderni linguaggi di programmazione orientata agli oggetti. Pertanto questa tipologia di database aderisce bene con l'architettura a microservizi, supporta i dati in formato JSON (formato dati più utilizzato nel web) e si integra nativamente con lo sviluppo in linguaggio Java.

La seconda topologia illustrata era Graph Database, in questo caso il data model è una struttura a grafo con nodi, archi e proprietà rappresentanti i dati. Questa tipologia è stata scartata in quanto utilizzata in applicazioni che hanno una topologia da rappresentare o che hanno dei dati costituiti principalmente da relazioni, caratteristiche che non ritroviamo però nei nostri dati prettamente numerici. Esempi di questi database sono: Neo4j e HyperGraphDB.

La terza e ultima topologia Key-Value and Wide Column Database si basa su un data model di tipo key-value che è il tipo base di tutti i database NoSql. Ogni elemento salvato nel database è costituito da un nome/chiave a cui è associato un valore. Con il termine Wide Column Database si intende invece un database che memorizza i dati con una mappa ordinata, sparsa, distribuita e multi dominio. Anche questa tipologia è stata esclusa in quanto utilizzata solo per una serie ristretta di applicazioni che eseguono unicamente query sui dati con una singola coppia chiave valore. Le caratteristiche rilevanti di questa topologia sono le prestazioni e la scalabilità, che possono essere altamente ottimizzate grazie alla semplicità dei modelli di accesso ai dati utilizzati. Esempi di questi database sono: Riak, Redis e Cassandra.

Un altro aspetto valutato riguarda la maturità delle API esposte dal database. Infatti non esiste uno standard per interfacciarsi con i sistemi NoSQL. Ogni database presenta un diverso design e diverse funzionalità. La maturità delle API con cui si interagisce con il database NoSql può avere notevoli ripercussioni nel tempo e nel costo di sviluppo ma anche nel mantenimento del sistema NoSQL sottostante. Alcuni sistemi forniscono anche interfacce RESTful. Questo approccio ha il fascino della semplicità e della familiarità, ma soffre delle latenze relative al trasporto in rete e del protocollo Http.

Per ultimare la scelta del "Document Database" più adatto al progetto occorreva valutare le prestazioni dei singoli prodotti presenti nel mondo open source. È risultata interessante l'analisi svolta in un articolo pubblicato dall'università di Auckland in Nuova Zelanda nel 2013 [8] intitolato: "A performance comparison of Sql and NoSql database". In questo viene fatto un confronto prestazionale tra un

database Sql e vari database NoSql. In particolare vengono confrontati: Microsoft SQL come database Sql, MongoDB, CouchDB, Cassandra e HyperTable come database NoSql e vengono analizzate le operazioni di scrittura, lettura, rimozione e inserimento di un insieme di coppie chiave-valore.
Di seguito sono riportati i risultati raccolti in questo articolo scientifico e ritenuti più importanti per la nostra valutazione. Le operazioni considerate sono quelle di scrittura e lettura, infatti si ipotizzava correttamente che queste sarebbero state le operazioni più comuni del funzionamento dell'architettura da sviluppare.

Database	Number of operations					
	10	50	100	1000	10000	100000
MongoDB	61	75	84	387	2693	23354
RavenDB	570	898	1213	6939	71343	740450
CouchDB	90	374	616	6211	67216	932038
Cassandra	117	160	212	1200	9801	88197
Hypertable	55	90	184	1035	10938	114872
Couchbase	60	76	63	142	936	8492
MS SQL Express	30	94	129	1790	15588	216479

TABLE III
TIME FOR WRITING (MS)

Figura 3.3. Tempi per la scrittura

Database	Number of operations					
	10	50	100	1000	10000	100000
MongoDB	8	14	23	138	1085	10201
RavenDB	140	351	539	4730	47459	426505
CouchDB	23	101	196	1819	19508	176098
Cassandra	115	230	354	2385	19758	228096
Hypertable	60	83	103	420	3427	63036
Couchbase	15	22	23	86	811	7244
MS SQL Express	13	23	46	277	1968	17214

TABLE II
TIME FOR READING (MS)

Figura 3.4. Tempi per la lettura

Questi risultati risultano poco differenti da quelli registrati anche nelle altre operazioni. Si nota che solo alcuni database NoSql eseguono le operazioni egregiamente, infatti solo MongoDB e CouchBase risultano estremamente rapidi in queste operazioni. CouchBase è leggermente più veloce, ma non supporta operazioni più complesse come restituire l'insieme completo delle chiavi. Un ultimo elemento su cui ci si è documentati è stato la condizione in cui vive l'azienda e il prodotto che offre. Infatti, bisogna analizzare lo stato del progetto, la documentazione fornita, se sono pianificati altri aggiornamenti e se c'è una comunità attiva che utilizza tale prodotto.
Detto ciò, si è scelto MongoDB perchè è un "Document Database", risulta il più veloce, efficiente e completo dall'analisi prestazionale trovata nell'articolo precedente. Inoltre MongoDB è anche un prodotto molto popolare in ambiente aziendale, tra i suoi più importanti clienti troviamo: Google, Facebook, Amadeus, Cisco, EA-sports e tanti altri. Questi dichiarano che i tempi necessari per impiegare questo prodotto, in nuovi progetti o già esistenti, sono davvero ridotti . MongoDB fornisce anche il supporto agli "Idiomatic Drivers", che consistono in un livello software, in uno specifico linguaggio di programmazione, che aiuta i programmatori ad interagire con la base di dati sottostante. Ad esempio forniscono metodi get and set di Document o campi di questi nei Document Database. MongoDB supporta gli Idiomatic Drivers per diversi linguaggi di programmazione tra cui: Java, Python, C++ ed altri. Esso è anche supportato da una documentazione online che io ritengo chiara, ampia e aggiornata.

Successivamente si è proseguito ad analizzare le minacce di sicurezza di cui soffrono i database NoSql. Le più comuni sono elencate di seguito:

- Utilizzo improprio dell'operatore $gt nelle query javascript;
- Caratteri speciali(' , " , \, ; , $, < >) utilizzati nelle query javascript;
- Tutte le variabili $*nomevariabile* tra apici singoli;

Per rimediare a queste vulnerabilità di sicurezza sono state svolte due operazioni:

1. **Disabilitare l'esecuzione del Javascript**, utilizzato per effettuare query sulla base di dati.
2. **Abilitare l'autorizzazione degli accessi** alla base di dati.

Queste operazioni sono state possibili con l'aggiunta di alcune righe specifiche, riportate di seguito, nel file di configurazione di MongoDB.

```
security:
 javascriptEnabled: false
 authorization: enabled
```

Figura 3.5. Frammento file mongod.conf

Successivamente sono stati creati degli utenti per l'accesso alla base di dati. È stato creato un utente amministratore con privilegi di scrittura e lettura su ogni database e un altro anche di amministrazione capace di scrivere e leggere soltanto sul database che si era creato per i primi test. Si riconosce la mancanza di altri utenti con privilegi più ristretti, ma in fase di sviluppo questi non erano necessari.

Fatto ciò è stata implementata una API RESTful che espone degli endpoints da contattare per svolgere operazioni di scrittura e di lettura sul database di MongoDB. Si procede ad indicare gli endpoint più utilizzati per erogare il servizio di monitoraggio di alto livello:

1. **GET** `/mdts/header-mdt?user=email_user`
2. **GET** `/mdts/body-mdt?_id=id_univoco`

Entrambi gli endpoint operano su uno stesso database di dati, ma su *colletion* differenti che sono: "mdt_header" e "mdt_body". Il primo endpoint riceve una *request-param* settata con l'email di un utente e restituisce una risposta Http contenente

un JSONArray. Questa risposta contiene la lista degli header delle campagne MDT eseguite dall'utente identificato dall'email precedente. Questo set d'informazioni serve nell'interfaccia grafica per mostrare lo stato corrente delle campagne MDT in corso e lo storico di quelle concluse. Il secondo endpoint riceve come request-param l'_id che identifica in maniera univoca la campagna MDT, questo id è restituito dal database MongoDB nel momento in cui si effettua un'operazione d'inserimento.

La seguente immagine illustra, in alto a destra, la struttura dei *document* contenuti nella collection "mdt_header". Questi document contengono le informazioni di contesto di ciascun MDT. Nella restante parte è illustrata la struttura dei document contenuti nella collection "mdt_body", essi contengono i dati prestazionali del MDT cioè le misure di RSRP e RSRQ registrate da celle differenti.

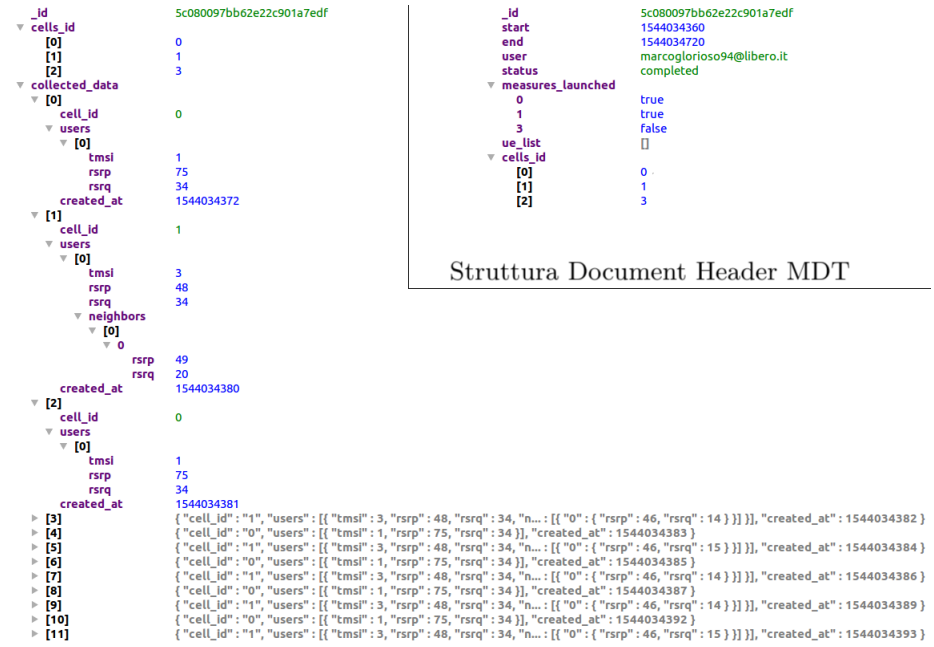

Figura 3.6. Struttura Document Body MDT

Nella struttura del document contenuto nella collection "mdt_header" sono presenti i campi "start" ed "end" che assumono come valore due timestamp indicanti ora e data d'inizio e fine della campagna MDT. Il campo "status" indica lo stato attuale della campagna MDT, in questo caso "completed" indica che la campagna è conclusa. Successivamente è interessante analizzare il vettore "cells_id", questo

contiene gli id delle celle su cui l'utente ha lanciato la campagna MDT, nell'immagine è stata lanciata sulle celle con id 0, 1 e 3. Rimane il campo "ue_list" predisposto ad accogliere una lista di "id" di UE per poter lanciare una campagna MDT su specifici device conosciuti dall'amministratore di rete.

Nella seconda struttura rappresentante il body dell'MDT, il campo "collected_data" è un vettore contenente le misurazioni periodiche di RSRP e RSRQ provenienti da celle differenti. Ogni elemento di questo vettore rappresenta una misurazione e contiene i seguenti campi:

- "cell_id", rappresenta l'id della cella sorgente che ha registrato la misurazione.
- "users" è un vettore contenente più campi associati all'UE che ha inviato la misura. Contiene il campo "tmsi" che identifica il maniera univoca l'UE che ha inviato la misura, i campi "rsrp" e "rsrq" settati con valori misurati e un ulteriore campo chiamato "neighbors" che è un vettore contenente le misure inviate, dallo stesso UE, a celle vicine ma diverse dalla sorgente.
- "created_at" assume come valore un timestamp, che indica la data e ora di quando la misura è stata registrata.

Nello sviluppo del servizio di storage, riveste un ruolo importante la realizzazione di un sistema di validazione dei *request-param* ricevuti sugli endpoint. La validazione è stata realizzata attraverso la classe java "MapValidator" implementata per raggiungere questo scopo. Il servizio supporta anche la scrittura sui log, in particolare si è scelto di utilizzare la libreria Log4j di Apache. Questa permette di realizzare un ottimo sistema di logging per tenere sotto controllo il comportamento del sistema sviluppato. Tramite un opportuno file di configurazione è possibile scegliere diversi livelli di log (error, info, debug, warnig, fatal) ed è possibile usufruire di diverse tecnologie di scrittura, denominate *Appender* all'interno della libreria, tra queste rientrano: file, socket, coda JMS e l'invio di email utilizzando il protocollo SMTP o JavaMail.

In conclusione nel file "application.properties" sono state settate alcune proprietà importanti per il funzionamento del servizio nell'infrastruttura a microservizi. Tra queste troviamo:

- **spring.application.name**, rappresenta il nome con cui il servizio si presenta al resto dell'infrastruttura, è stato scelto il nome di "mongodb-service".
- **server.port**, settata al valore 7000 è la proprietà che indica la porta su cui è in ascolto il servizio.
- **server.address**, indica l'indirizzo ip su cui è in ascolto il servizio.

- **management.endpoints.web.exposure.include**, permette al servizio di scegliere quali endpoint esporre, nel nostro caso settato ad "*" per esporre tutti gli endpoint.

3.2.3 Authentication Service

Lo sviluppo del servizio di autenticazione deriva dalla volontà di fornire un primo livello di sicurezza nell'infrastruttura a microservizi sviluppata. Si è scelto di utilizzare il protocollo OAuth 2.0 per realizzare un sistema di autorizzazione basato su token di accesso che è una stringa di caratteri alfanumerici. Questo protocollo è ampiamente utilizzato da grossi fornitori di servizi come Google, Facebook, GitHub per permettere a un client con un singolo token di accedere ad API su diversi server di risorse. Il protocollo OAuth 2.0 definisce i seguenti ruoli:

- Server di Autorizzazione.
- Server di Risorse.
- Client Software.
- Utente (si ipotizza uno schema confidenziale tra utente e client software).

Tra questi ruoli si svolgono delle interazioni che sono definite dal protocollo. La seguente immagine illustra soltanto alcune di queste iterazioni.

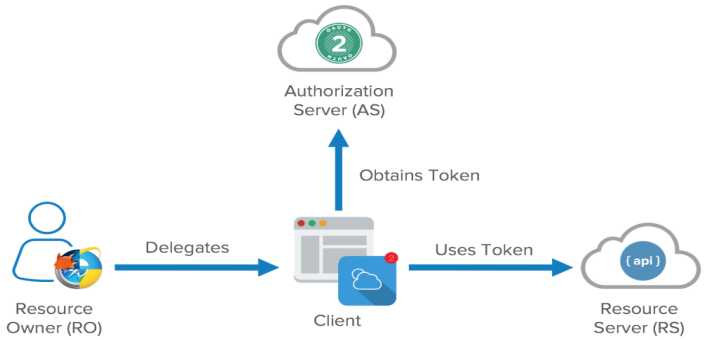

Figura 3.7. Interazioni base del protocollo OAuth 2.0

Il Client Software contatta il Server di Autorizzazione con una richiesta POST Http, con la quale le credenziali del Client Software sono inviate al Server di Autorizzazione. Il server di autorizzazione verifica l'identità del Client e se l'autenticazione è avvenuta con successo risponde con un messaggio 200 "OK". La risposta del server contiene nel body alcune proprietà e le più importanti sono: "access_token" contenente il token creato per il client, "token_type" che indica il tipo del token ed "expires_in" che indica la durata temporale in cui il token è valido. Si intuisce che per lo scambio d'informazioni avvenuto nel precedente passaggio, il protocollo OAuth 2.0 preveda una comunicazione di rete protetta, probabilmente con il protocollo TLS. Successivamente il client, ricevuto un token di accesso, contatta il Server di Risorse con un richiesta Http che contiene nell'header il campo "Authorization" dove sarà contenuto il token. Il Server di Risorse deve validare il token e se questo è autentico e non è scaduto, deve permettere al Client l'accesso alle risorse.
Quanto detto sin qua è un breve riassunto di cosa stabilisce il protocollo OAuth 2.0. Tuttavia il protocollo non definisce i seguenti aspetti:

- Il processo di autenticazione del Client Software nei confronti del Server di Autorizzazione.
- Il formato, la struttura e i metodi di utilizzo del token, basato sui requisiti di sicurezza del Server di Risorse.
- Il processo di validazione del token eseguito sul Server di Risorse.

Il primo aspetto deriva dal fatto che OAuth 2.0 è un protocollo di autorizzazione e non di autenticazione, quindi questa fase è necessaria, ma non viene specificato lo standard o il protocollo con cui la si debba eseguire. Questo è ampiamente chiarito in un articolo della comunità "OAuth2.0.net" redatto da Justin Richer intitolato "End User Authentication with OAuth 2.0" e presente al link `https://oauth.net/articles/authentication/`. Utilizzando OAuth 2.0 è probabile che ogni provider sviluppi le proprie API di autenticazione, ad esempio l'identificativo di un utente potrebbe essere trovato nel campo "user_id" in un provider ma nel campo "subject" in un altro provider. Anche se questi sono equivalenti, richiederebbero lo sviluppo di sezioni di codice differenti cioè mentre l'autorizzazione può avvenire allo stesso modo su ciascun fornitore, la trasmissione delle informazioni di autenticazione potrebbe essere diversa. Questo problema può essere attenuato utilizzando un protocollo di autenticazione standard costruito su OAuth 2.0 che trasmette allo stesso modo le informazioni sull'identità indipendentemente da dove esse provengono, esempio è: OpenID Connect. Il secondo aspetto, deriva dal fatto che il protocollo OAuth 2.0 non impone un formato di token fisso e per conoscere le specifiche su come utilizzare i token e quelli "Self-encoded" bisogna documentarsi sull'RFC 6750 che è immediatamente successivo a quello del protocollo OAuth 2.0 (RFC 6749). L'ultimo aspetto riguarda la validazione del token,

una fase fondamentale per l'autorizzazione, dove il Server di Risorse deve validare che il token è associato al client, che il token non è scaduto e che è stato erogato da un Server di Autorizzazione di cui si fida. Le specifiche del protocollo OAuth 2.0 non definiscono come il Server di Risorse dovrebbe verificare i token di accesso, ma specificano la necessità di un coordinamento tra i Server di Risorse e di Autorizzazione. Nei sistemi di maggior dimensione, ad esempio in un modello a microservizi, l'endpoint delegato all'emissione del token e quello dove viene esercitato si trovano su server diversi, ciò ha portato a protocolli proprietari per la comunicazione tra i due server citati. L'estensione "OAuth 2.0 Token Introspection" definita nell'RFC 7662, specifica un endpoint che restituisce informazioni su un token di accesso, destinato ad essere utilizzato dai Server di Risorse interni al sistema. L'endpoint d'introspezione deve essere in grado di restituire informazioni su un token, pertanto è probabile che lo si crei nello stesso server in cui si trova l'endpoint che eroga i token, quest'ultimo e quello d'introspezione devono condividere un database o, se sono stati implementati token "Self-encoded", devono condividere il segreto per la codifica.
Sulla base di quanto riportato è stato necessario fare delle scelte per risolvere gli aspetti del protocollo OAuth 2.0 lasciati senza una specifica implementativa.
Si è scelto di utilizzare un token "Self-encoded", precisamente il **JSON Web Token** (JWT) che è uno standard (RFC 7519) che definisce un modo compatto e autonomo per trasmettere informazioni tra le parti come oggetto JSON in modo sicuro. Questa informazione può essere validata perché è firmata digitalmente. Il token JWT può essere firmato usando un segreto condiviso con l' algoritmo HMAC o una coppia di chiavi pubblica/privata usando RSA o ECDSA. Nella sua forma compatta, i token JWT sono composti da tre parti separate da un punto (.) che sono: Header, Payload, Signature. Si procederà ad analizzare i campi più importanti di queste tre sezioni e successivamente sarà riportata un'immagine del token realizzato nel lavoro di tesi.
Nella sezione Header troviamo i seguenti campi:

- **"typ"**, indica il tipo di token utilizzato.
- **"alg"**, indica l'algoritmo di firma utilizzato.

Questo sezione viene codificata in Base64Url.
Nel Payload troviamo i seguenti campi:

- **"sub"**, indica il soggetto a cui è attribuito il token.
- **"iss"**, indica l'emittente che ha erogato il token.
- **"iat"**, indica il momento in cui è stato erogato il token(timestamp).
- **"exp"**, indica il momento in cui il token termina di validità(timestamp).

Anche questa sezione viene codificata in Base64Url.
Per creare la parte Signature occorre legare insieme l'Header codificato con il Payload codificato separati da un punto, aggiungere il segreto e utilizzare l'algoritmo specificato nel campo "alg" per ottenere la firma.
Ad esempio, se si desidera utilizzare l'algoritmo HMAC SHA256, la firma verrà creata nel modo seguente:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Figura 3.8. Firma Token JWT

L'output è composto da tre stringhe Base64-URL separate da punti che possono essere facilmente trasmesse negli ambienti HTML e Http.
La seguente immagine illustra il token JWT che si è sviluppato ed utilizzato nell'infrastruttura a microservizi:

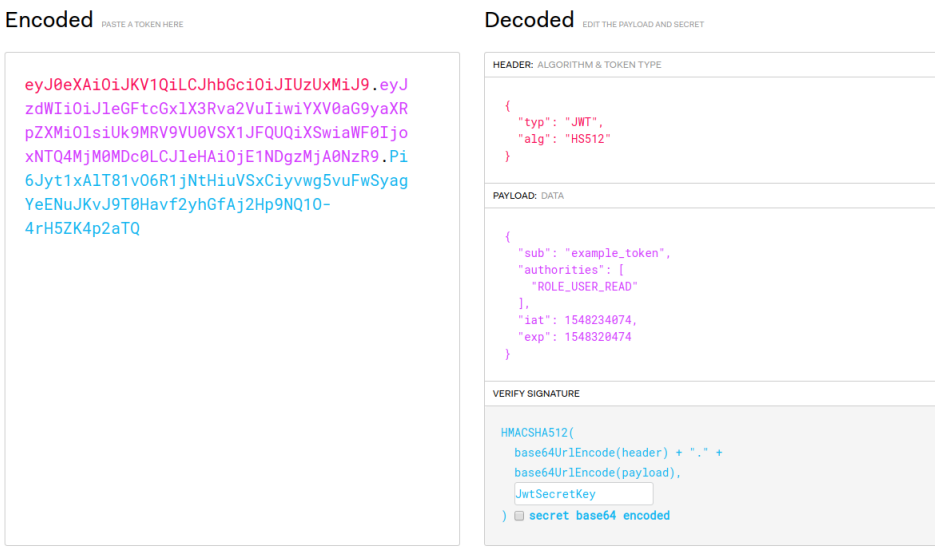

Figura 3.9. Token JWT

Nell'immagine sono presenti i due campi "iat" e "exp" che indicano l'inizio e la fine della validità del token erogato. In particolare si è stabilita una validità temporale uguale a 24 ore, tale validità è stata considerata accettabile per un funzionamento efficiente e anche sicura, in quanto un token JWT non può essere revocato fino alla sua scadenza. Si è scelto di utilizzare un algoritmo di firma HS512, cioè HMAC usando l'algoritmo di hash SHA-512. Si è aggiunto un campo non convenzionale chiamato "authorities", questo viene settato con un array che contiene delle stringhe che definisco dei ruoli con accessi differenti. È stato definito il ruolo "ROLE_USER_READ" per un accesso alle risorse soltanto in modalità lettura, il ruolo "ROLE_USER_WRITE" per un accesso soltanto in modalità scrittura e ultimo ruolo "ROLE_ADMIN" per un accesso alle risorse in modalità amministratore. A un Client possono essere associati uno o più ruoli tramite l'endpoint d'iscrizione sull'Authentication Service sviluppato. Il controllo degli accessi viene effettuato sui servizi di risorse che controllano se la risorsa è accessibile al Client Software con uno o più ruoli. Ogni volta che il Client desidera accedere a una risorsa protetta, deve inviare il token JWT, nel campo Authorization dell'header Http preceduto dalla parola chiave "Bearer". La seguente immagine illustra le interazioni del protocollo OAuth 2.0 con l'utilizzo del token JWT.

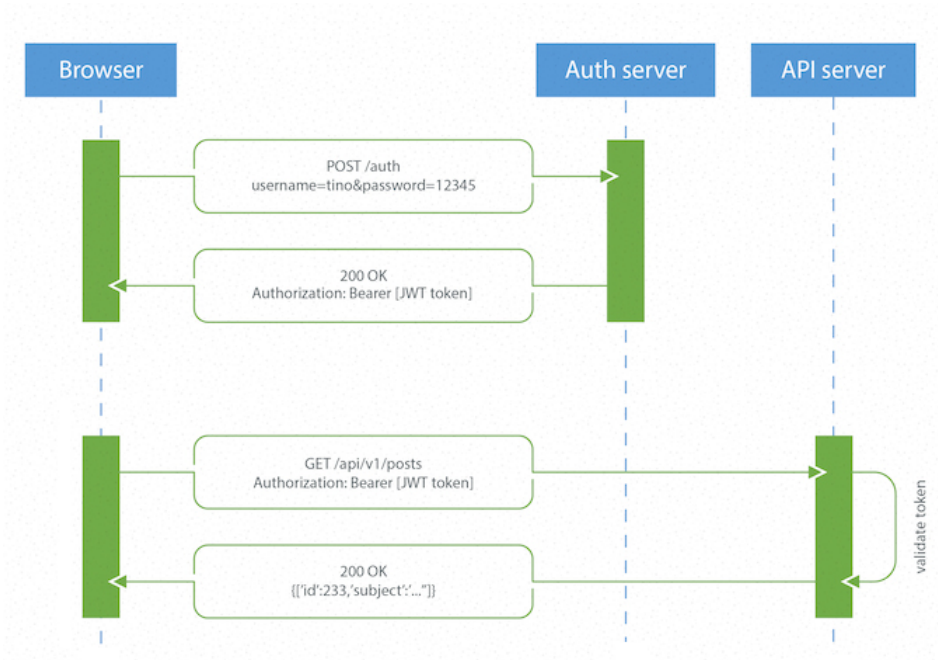

Figura 3.10. Interazioni OAuth 2.0 con token JWT

La scelta di utilizzare un token JWT "Self-encoded" pone chiarezza sul token da utilizzare e sul suo formato, aspetto che non era specificato nello standard del protocollo OAuth 2.0 tuttavia si è scelto un prodotto sviluppato appositamente per cooperare con il protocollo di autorizzazione in questione. Il token JWT, realizzato in formato JSON, può essere facilmente mappato in oggetti attraverso i parser presenti nella maggior parte dei linguaggi di programmazione OOP. In aggiunta la validazione del token sui Server di Risorse è possibile grazie alla natura "Self-encoded" intrinseca del token JWT. Il token, infatti, è una stringa traducibile in un JSON file che contiene tutte le informazioni necessarie alla validazione del token, pertanto non è più necessario contattare il Server di Autorizzazione per ottenere tali informazioni. Inoltre è stato possibile eliminare un database specifico, in cui salvare le informazioni associate a ciascun token erogato dal Server di Autorizzazione. Attraverso la verifica della firma contenuta nel token JWT, è possibile verificare l'entità che ha erogato il token e garantire l'integrità del token attraverso la rete. In conclusione la validazione del token JWT viene eseguita sui Server di Risorse, escludendo la necessità di sviluppare un ulteriore endpoint sul Server di Autorizzazione per ottenere le informazioni necessarie per la validazione, come invece è suggerito nell'estensione del protocollo OAuth 2.0, chiamata "OAuth 2.0 Token Introspection".

Successivamente alla scelta del token da utilizzare, è stato necessario sviluppare la fase di autenticazione del Client Software nei confronti del Server di Autorizzazione, ulteriore aspetto non definito nello standard del protocollo OAuth 2.0. Per fare ciò è stato sviluppato un endpoint di "sign-up", cioè un endpoint di registrazione per un nuovo Client nei confronti dell'Authentication Service. La seguente immagine illustra la richiesta POST Http necessaria per registrare un nuovo Client:

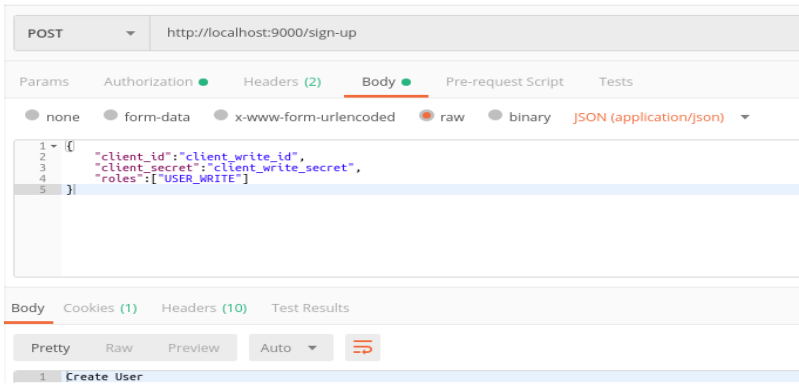

Figura 3.11. Richiesta POST per registrare un nuovo Client

Nella richiesta sono inviati un "client_secret", un "client_id" e l'array "roles" che contiene i privilegi di accesso. Il servizio di autenticazione risponde con 201 "CREATED". È possibile inviare in chiaro nel body le credenziali perché s'ipotizza la presenza di un canale sicuro di rete, realizzato probabilmente con il protocollo TLS, ricordo che questo è un requisito necessario per il funzionamento sicuro del protocollo OAuth 2.0. Il body della richiesta POST viene mappato in un oggetto della classe "ApplicationUser". In seguito il contenuto del campo "client_secret" viene processato da una classe di criptografia fornita dal framework Spring. Quest'ultima è la classe "BCryptPasswordEncoder" che utilizza la potente funzione di hash BCrypt, che contiene al suo interno un sale per proteggere il segreto contro attacchi "Rainbow Table" e permette di aumentare il numero d'iterazioni per renderla più lenta, in modo da essere resistente ad attacchi a forza bruta. Infine l'oggetto della classe ApplicationUser è pronto per essere salvato in un database MongoDB. Questo database si chiama "DATABASE_USERS" e contiene la collezione "users", in cui sono memorizzati i Client registrati tramite l'endpoint di "sign-up". In questa collezione il "client_secret" è salvato criptografato e non in chiaro.
Il servizio di autenticazione espone un ulteriore endpoint sul quale è possibile effettuare il login e ricevere un token JWT. La seguente immagine illustra la richiesta POST Http sull'endpoint di login.

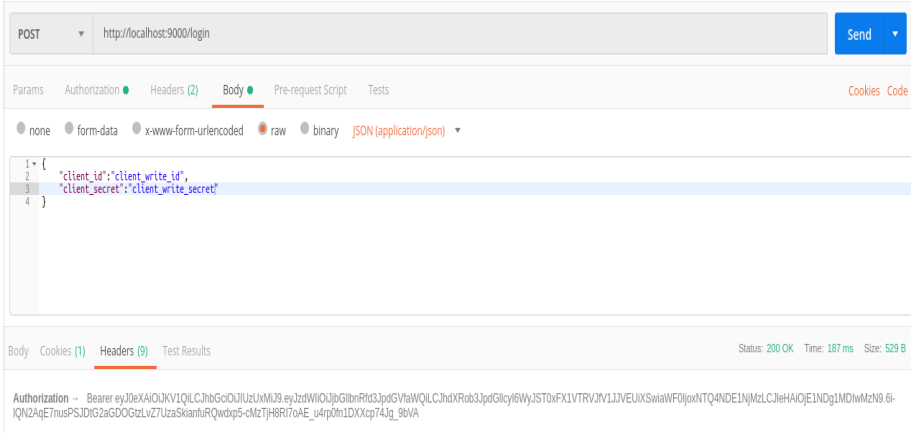

Figura 3.12. Richiesta POST per il login e ricezione del token JWT

Nell'immagine è possibile osservare la risposta 200 "OK" del servizio di autenticazione che contiene nell'header Http il token JWT. Nel servizio di autenticazione, alla ricezione di tale richiesta, si attiva un filtro di sicurezza supportato dal modulo Spring Security, chiamato "UsernamePasswordAuthenticationFilter" che è stato esteso nella classe chiamata "JWTAuthenticationFilter" e questa è composta da due metodi:

1. **public Authentication attemptAuthentication(...)**, questo metodo recupera le credenziali dalla richiesta Http e crea un oggetto della classe "AuthenticationManager". Esso è il delegato a svolgere le funzioni di autenticazione all'interno del modulo Security Spring e invoca il metodo "loadUserByUsername(String client_id)" (implementato nella classe "UserDetailsServiceImpl") che accede al database MongoDB e prova a restituire un oggetto "User" corrispondente alle credenziali ricevute. Se quest'ultimo metodo restituisce il risultato atteso, l'AuthenticationManager autentica l'oggetto "User" e restituisce un oggetto "Authentication". Viceversa ritorna una risposta 401 "UNAUTHORIZED".

2. **protected void successfulAuthentication(...)**, questo secondo metodo viene chiamato nel caso in cui l'autenticazione avvenuta nel metodo precedente ha avuto successo. L'oggetto "Authentication" precedentemente restituito diviene uno dei parametri di questo metodo. Detto ciò, questo metodo costruisce il token JWT corrispondente all'autenticazione, lo lega alla stringa "Bearer" e lo colloca nel campo "Authorization" dell'header Http. Infine restituisce la risposta Http al Client che aveva effettuato la richiesta POST sull'endpoint di login.

Per completare l'analisi della fase di autenticazione e per chiarire le azioni svolte da questi due metodi, si riporta il codice di cui questi sono costituiti.

```
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
    HttpServletResponse response)throws AuthenticationException{

    try {
        // 1. Get credentials from request
        UserCredentials creds = new
            ObjectMapper().readValue(request.getInputStream(),
            UserCredentials.class);

        // 2. Create auth object (contains credentials) which will be used
            by auth manager
        // 3. Authentication manager authenticate the user, and use
            UserDetialsServiceImpl::loadUserByUsername() method to load the
            user.
        return authManager.authenticate(
               new UsernamePasswordAuthenticationToken(
                       creds.getClient_id( ),
                       creds.getClient_secret( ),
                       new ArrayList <>()));

    } catch (IOException e) {
        AuthenticationApplication.logger.error(e.getMessage());
        throw new RuntimeException(e);
    }
}
```

```
// Upon successful authentication, generate a token.
// The 'auth' passed to successfulAuthentication() is the current
    authenticated user.
@Override
protected void successfulAuthentication(HttpServletRequest request,
    HttpServletResponse response, FilterChain chain,Authentication auth)
    throws IOException, ServletException {

    Long now = System.currentTimeMillis();
    // Create the token
    String token = Jwts.builder()
            .setHeaderParam("typ", "JWT")
            .setSubject(auth.getName())
            .claim("authorities", auth.getAuthorities().stream()
                    .map(GrantedAuthority::getAuthority).collect(Collectors.toList()))
            .setIssuedAt(new Date(now))
            .setExpiration(new Date(now + jwtConfig.getExpiration() *
                1000)) // in milliseconds
            .signWith(SignatureAlgorithm.HS512,
                jwtConfig.getSecret().getBytes())
            .compact();

    // Add token to header
    response.addHeader(jwtConfig.getHeader(), jwtConfig.getPrefix() + " " +
        token);
    AuthenticationApplication.logger.info("Token created for the user: " +
        auth.getName());


}
```

Questo servizio espone un altro endpoint per eseguire un'operazione di rimozione di un Client già registrato risalendo a questo tramite il suo "client_id". L'endpoint è il seguente:

1. **DELETE** `/users?client_id=client_id_to_delete`

Questo endpoint come anche quello di sign-up sono accessibili soltanto ad un utente con un accesso "ADMIN".

Anche in questo servizio troviamo il supporto al sistema di logging realizzato con la libreria Log4j di Apache.

In questo servizio troviamo più file di configurazione. Il primo file è "application.properties" dove sono presenti le stesse proprietà analizzate nel primo servizio. In particolare la proprietà **spring.application.name** è settata con la stringa "auth-service" e la **server.port** è settata con il valore 9000.
Il secondo è "config.properties", in questo file troviamo molteplici proprietà di sicurezza settate per il corretto funzionamento del servizio. Quelle ritenute più importanti sono elencate di seguito:

- **security.jwt.header**, proprietà che indica il nome del campo header Http dove deve essere inserito il token, il valore settato è "Authorization".
- **security.jwt.prefix**, proprietà che indica la stringa che deve precedere il token, il valore settato è "Bearer".
- **security.jwt.expiration**, proprietà che indica la durata del token, questa è un intero che indica il numero di secondi, il valore settato è 86400.
  (24*60*60 = 1 giorno)

3.2.4 Gateway Zuul Service

Gateway Zuul Service è stato realizzato per svolgere le funzioni di "routing" incluse nel modulo Spring Clound. Questo modulo permette d'introdurre nell'architettura un *reverse proxy* che semplifica lo sviluppo di molti casi d'uso in cui un'applicazione UI effettua richieste per contattare uno o più servizi di *back-end* e permette di gestire i problemi CORS, l'autenticazione e l'autorizzazione in modo indipendente per tutti i servizi di *back-end*. Un *reverse proxy* di solito svolge anche le funzioni di controllo degli accessi alla rete privata, di bilanciamento del carico e di caching. Il Gateway Zuul Service, sviluppato in questo lavoro di tesi, gestisce il flusso di traffico in ingresso e in uscita dall'architettura a microservizi. Questo servizio sviluppato potrebbe essere affiancato o inglobare al suo interno un firewall al fine di creare una DMZ che contenga i servizi dell'architettura che vogliamo esporre. Pertanto un utente esterno può accedere soltanto ai servizi messi a disposizione, senza mettere a rischio la sicurezza dell'intera architettura. Un esempio sarebbe mettere in DMZ i server di front-end e lasciare i server di back-end in LAN. In questo modo limitiamo il traffico che dall'esterno può entrare nella nostra architettura e nel momento in cui un servizio che abbiamo esposto in DMZ è sotto attacco, questo difficilmente si propaga nelle altri parti dell'architettura perché il traffico è fortemente limitato dal firewall sia nella direzione d'ingresso sia in quella di uscita. Tuttavia questo servizio diventa un "single point of failure", una componente che, in caso di non funzionamento, ad esempio per un guasto o per un attacco, interrompe l'erogazione dei servizi da parte dell'architettura a microservizi. Si ritiene

opportuno sottolineare che le precedenti sono delle valutazioni di carattere generale, in quanto l'architettura realizzata è destinata a un deploy in laboratorio, ma è valido ipotizzare che l'architettura sia distribuita su reti differenti con annessa DMZ nelle sue successive evoluzioni.
Per realizzare queste funzionalità d'instradamento è stata usata la componente "Zuul" contenuta in uno dei progetti più importanti di Sping Cloud chiamato "Spring Cloud Netflix". Zuul viene integrato nel nostro servizio aggiungendo una dipendenza nel file "pom.xml" e viene attivato aggiungendo l'annotazione "@EnableZuulProxy" sopra la definizione della classe che contiene il metodo "main" che lancia in esecuzione l'applicazione.
L'introduzione di un reverse proxy nell'architettura sviluppata determina le proprietà precedentemente discusse, ma semplifica anche la validazione del token JWT. Il reverse proxy diventa l'unico servizio a eseguire la validazione del token JWT, essendo il primo ad essere contattato dall'esterno è possibile traslare la fase di validazione da ogni Server di Risorse al solo Gateway Zuul Service. Questa scelta architetturale determina che i Servizi di Risorse siano ignari della presenza del protocollo OAuth 2.0, siano più semplici in quanto privi del modulo Spring Security e infine non abbiano conoscenza delle informazioni utilizzate per firmare e per verificare la firma. Il servizio implementato si compone di due classi fondamentali:

1. **WebSecurity**
2. **JWTAuthorizationFilter**

La prima classe presenta l'annotazione "@EnableWebSecurity", estende la classe "WebSecurityConfigurerAdapter" del modulo Spring Security ed implementa un metodo molto importante chiamato "configure". Questo metodo permette di configurare un filtro di Spring Security che si attiva ogni qualvolta il servizio riceve una richiesta Http. Questo filtro utilizza un metodo della classe "JWTAuthorizationFilter" che estende uno dei filtri base di Spring Security. La funzione di questo filtro è svolgere la validazione del token JWT. Il metodo "configure" definisce anche i ruoli necessari per accedere a ciascun endpoint definito. Di seguito è riportata l'implementazione del metodo "configure".

```
@Override
protected void configure(HttpSecurity http) throws Exception
{   //use stateless session and no csrf token
    http.csrf().disable()
    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
    //handle an authorized attempts
    .exceptionHandling().authenticationEntryPoint((req, rsp, e) ->
        rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED)).and()
    //Add a filter to validate the tokens with every request
    .addFilterAfter(new JWTAuthorizationFilter(jwtConfig),
        UsernamePasswordAuthenticationFilter.class)
    //authorization requests config
    .authorizeRequests()
    //formard rules to authorization service
    .antMatchers(HttpMethod.POST, jwtConfig.getUriLogin()).permitAll()
    .antMatchers(HttpMethod.POST, jwtConfig.getUriSign()).hasRole("ADMIN")
    .antMatchers(HttpMethod.DELETE,
        jwtConfig.getUriUsers()).hasRole("ADMIN")
    .antMatchers(HttpMethod.GET,
        "/mongodb-service/**").hasAnyRole("USER_READ", "ADMIN")
    .antMatchers(HttpMethod.POST,
        "/mongodb-service/**").hasAnyRole("USER_WRITE", "ADMIN")
    .antMatchers(HttpMethod.GET,
        "/controller-service/**").hasAnyRole("USER_READ", "ADMIN")
    .antMatchers(HttpMethod.POST,
        "/controller-service/**").hasAnyRole("USER_WRITE", "ADMIN")
    // Any other request must be authenticated
    .anyRequest().authenticated();
}
```

Dal frammento di codice precedente si nota che soltanto un utente con ruolo "ADMIN" può registrare un nuovo utente o eliminarne uno già registrato. Viceversa l'endpoint di login è volutamente lasciato accessibile a tutti gli utenti, invece tutte le altre richieste Http sono elaborate dal filtro di autorizzazione configurato.

La classe "JWTAuthorizationFilter" estende la classe "OncePerRequestFilter" ed implementa il filtro precedentemente accennato. Pertanto la classe implementa il metodo "doFilterInternal" che, ricevuta una richiesta Http (con o senza il token JWT), legge il campo Authorization dell'header, verifica che il contenuto non sia nullo ed inizi con la stringa "Bearer". Fatte queste prime verifiche il token viene estratto dall'header Http e si procede ad eseguirne la sua validazione. Questa consiste nel verificare la firma utilizzando il segreto condiviso tra il Gateway Zuul Service e l'Authentication Service e nel verificare la validità temporale del token.

Validato il token con successo, si estrae da questo il vettore "authorities" per verificare che il Client abbia adeguati permessi di accesso per l'endpoint richiesto. Una richiesta con permessi adeguati viene inoltrata al servizio di risorse, viceversa una con permessi non sufficienti riceverà una risposta 401 "UNAUTHORIZED".

Il Gateway Zuul Service contiene diversi file di configurazione che si procede ad elencare:

- **application.yml**, configura le proprietà generiche del servizio come, l'indirizzo, la porta, il nome dell'applicativo Spring, ecc.
- **config.properties**, configura le proprietà di sicurezza, come il campo dell'header Http dove leggere il token, il prefisso del token, la durata del token ed ecc.
- **lo4j.properties**, abilita il supporto per il sistema di logging.

Nel primo file di configurazione troviamo le proprietà "spring.application.name" settata con la stringa "zuul-service", la "server.port" settata con il valore 8762 e la **"zuul.sensitive-headers"**, che permette il trasporto dal Authentication Service al Client del token JWT nell'header Http, dopo una richiesta di login avvenuta con successo. In aggiunta la proprietà **"eureka.client.fetch-registry"**, contenuta anch'essa nel file application.yml, abilita il servizio a scaricare il registro dei microservizi attivi dal servizio di scoperta e registrazione (Eureka Service).

3.2.5 Discovery Service

Il Discovery Service è stato realizzato per assolvere unicamente le funzioni di "registrazione e scoperta dei microservizi" incluse nel modulo Spring Clound. Queste funzionalità sono importanti perché permettono di avere un primo livello di monitoraggio dell'architettura e di riconoscere velocemente se un servizio ha smesso di funzionare impiegando dei segnali periodici come *l'heartbeat*, che tradotto significa "battito cardiaco". Il servizio in esame è stato dotato della componente Eureka contenuta nel progetto "Spring Cloud Netflix" del modulo Spring Cloud. Pertanto è stato necessario aggiungere la dipendenza "Eureka Server" nel file "pom.xml" e l'annotazione "@EnableEurekaServer" sopra la definizione della classe contenente il metodo "main". In questo modo il "Discovery Service" diviene il server per la registrazione e la scoperta degli altri servizi attraverso la ricezione di segnali periodici. Analogamente negli altri servizi dell'architettura, analizzati precedentemente, è stata inserita la dipendenza "Eureka Client" nel file "pom.xml" ed è stata aggiunta l'annotazione "@EnableEurekaClient" nella classe contenente il metodo "main". I servizi, etichettati come "client" inviano alcuni metadati, come: nome dell'applicativo, indirizzo, porta e stato per registrarsi con il Discovery Service. Quest'ultimo

ricevendo i metadati registra il servizio mittente in un registro interno. I servizi "client" potrebbero anche richiedere il registro interno al Discovery Service per conoscere gli altri servizi attivi nell'architettura. L'annotazione "@EnableEureka-Client", secondo la documentazione di Spring, rende il servizio *instance* di Eureka in quanto si registra al server che contiene il registro interno ma potrebbe anche essere *client* che inoltra una richiesta al server per ottenere il registro dei servizi attivi. Le proprietà di configurazione ed i valori ad esse assegnati rivestono un ruolo chiave nella realizzazione del processo di scoperta e registrazione dei microservizi all'interno dell'architettura. Le proprietà utilizzate sono elencate di seguito:

- **eureka.client.register-with-eureka**, indica se l'applicativo deve registrasi o meno al server contenente il registro, ammette valore *true* o *false (default=true)*.

- **eureka.client.fetch-registry**, indica se l'applicativo deve richiedere al server il registro dei servizi registrati, ammette valore *true* o *false (default=false)*.

- **eureka.instance.prefer-ip-address**, indica la volontà dell'applicativo di registrarsi al server anche con il suo indirizzo ip oltre al nome dell'applicativo, ammette valore *true* o *false (default=false)*.

- **eureka.instance.ip-address**, indica l'indirizzo ip che l'applicativo comunica al server in fase di registrazione, necessario per effettuare l'instradamento in presenza di un proxy, ammette un indirizzo ip valido.

- **spring.application.name**, indica il nome dell'applicativo con cui questo si registra al server, non è una proprietà specifica della componente Eureka, ma viene utilizzata da essa, ammette una stringa rappresentante il nome dell'applicativo.

Si ritiene opportuno analizzare i valori attribuiti alle precedenti proprietà in alcuni servizi dell'architettura sviluppata. Il primo servizio è il "Discovery Service" che presenta tale configurazione:

- **eureka.client.register-with-eureka=false**, con il valore *false* si disattiva il processo al server di registrazione, questo valore è dovuto al fatto che il servizio in esame è il server Eureka stesso.

- **eureka.client.fetch-registry=false**, con il valore *false* si disattiva il processo che richiede al server l'elenco completo dei servizi registrati. La motivazione del valore *false* è la medesima della proprietà precedente.

Il Gateway Zuul Service che svolge le operazioni d'instradamento ed è posto all'ingresso della nostra architettura presenta la seguente configurazione:

- **eureka.client.register-with-eureka=true**, oltre a svolgere la funzione d'instradamento, riveste il ruolo di client nel servizio Eureka. Il Gateway Zuul Service deve registrarsi al server Eureka in quanto esso è un servizio dell'architettura e inoltre si desidera monitorare il suo funzionamento.

- **eureka.client.fetch-registry=true**, il valore *true*, indica che il servizio richiede periodicamente il registro dei servizi registrati al server Eureka. Tramite il registro scaricato, contenente informazioni come l'indirizzo ip, la porta e il nome del servizio, il Gateway Zuul Service è in grado di instradare le richieste ricevute a servizi interessati.

- **eureka.instance.prefer-ip-address=true**, il servizio annuncia il suo indirizzo ip al server Eureka.

- **eureka.instance.ip-address=proxy_address**, l'indirizzo ip annunciato.

Per l'Authentication Service, lo Storage Service e il Controller Service, che sarà analizzato del paragrafo successivo, la configurazione adeguata è la seguente:

- **eureka.client.register-with-eureka=true**, la registrazione al server Eureka permette di monitorare lo stato dei servizi e con i metadati inviati al server è possibile popolare il registro interno, indispensabile per l'instradamento sul gateway service.

- **eureka.instance.prefer-ip-address=true**, i servizi annunciano i loro indirizzi ip al server Eureka.

- **eureka.instance.ip-address=ip_address**, l'indirizzo ip annunciato.

In conclusione la componente Eureka abilita anche un'url, navigabile dal browser, che mostra l'"Eureka Dashboard", cioè una pagina html dove è possibile osservare lo stato, l'indirizzo e la porta di ciascun servizio attivo. Di seguito è riportato lo screenshot dell'Eureka Dashboard.

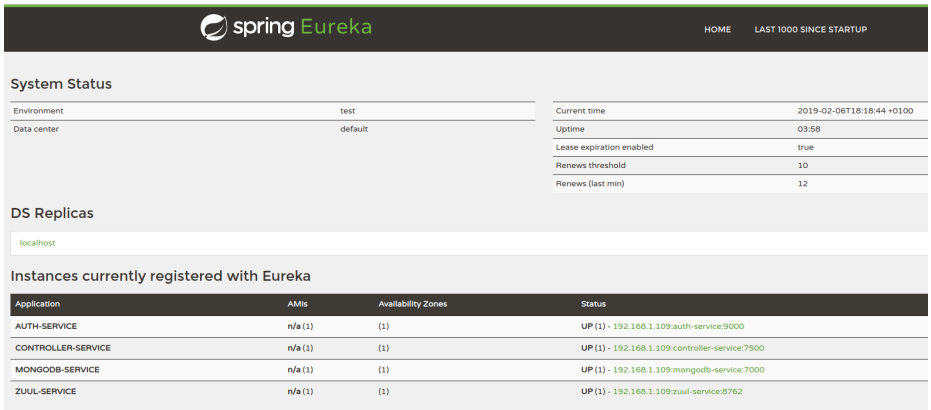

Figura 3.13. Eureka Dashboard

3.3 Controller Service

Il Controller Service favorire la comunicazione con il controllore RAN. Tuttavia è doveroso chiarire che, nonostante il lavoro di tesi interessi il 5G, il sistema sviluppato opera su una rete LTE essendo l'unica tecnologia disponibile nel laboratorio TIM presso cui si è svolto il lavoro di tesi. Il controllore RAN è delegato a coordinare i processi sulle tecnologie di rete sottostanti note col nome di Evolved Node B, abbreviato eNodeB. Quest'ultimi (eNodeBs), nelle reti LTE, devono gestire le risorse radio e la mobilità delle celle per ottimizzare tutte le comunicazioni con gli UE. Pertanto, le prestazioni di un eNodeB LTE dipendono dal suo algoritmo di gestione delle risorse radio e dalla sua implementazione. Il Controller Service espone un API delegata a realizzare la comunicazione con il controllore RAN. Gli endpoint sviluppati sono molteplici e sono utilizzati per diversi obiettivi che sono elencati di seguito:

- Ottenere le informazioni sugli eNodeB a partire da una specifica area geografica.
- Ottenere le informazioni sulle celle collegate ad un eNodeB identificato da un id univoco.
- Creare una campagna MDT.

Per ottenere le informazioni sugli eNodeB è stato svilupatto il seguente endpoint:

1. **GET** `/enodeb-functions/`

Questo endpoint riceve due *request-param* che sono "coordinates" e "technologies". Il primo rappresenta le coordinate di un poligono che viene disegnato sull'interfaccia web per indicare l'area geografica dove si desidera cercare gli eNodeB. Il parametro "technologies" invece è una lista contenente i tipi di tecnologie di rete che si desidera analizzare. I tipi di tecnologie prese in esame sono: 2G, 3G, 4G ed LTE con diverse frequenze. Queste informazioni ricevute sull'endpoint sono reperite sull'interfaccia web dall'iterazione con l'utente. Nel servizio in esame è stata sviluppata una classe chiamata "MapValidatorENB" delegata ad effettuare la validazione dei valori ricevuti per questi request-param. Validati i parametri il servizio alloca un client Http per eseguire una richiesta al controllore di basso livello. Con questa richiesta sono inoltrati i parametri ricevuti al controllore e possiamo affermare che il client Http svolge il ruolo di passacarte delle informazioni sull'area geografica e sulle tecnologie ricevute al controllore RAN che è esterno all'architettura di back-end. Il controllore risponde con una response Http con status code 200 "OK" se la richiesta è andata a buon fine. Nel body della response è contenuto un "JSONArray" composto da oggetti rappresentanti i singoli eNodeB che avranno soddisfatto i requisiti di area geografica e di tecnologia selezionati. Infine il Controller Service esegue una validazione dei dati, rappresentanti gli eNodeB, ricevuti dal controllore sottostante. Infine questi ritornano, attraverso una response Http, al web server che si occuperà del *rendering* sull'interfaccia web.

Il controllore RAN è abilitato ad inviare ulteriori informazioni sui dispositivi impiegati nella rete, infatti è possibile ricevere le informazioni sulle celle sotto il dominio di un eNodeB. Per ottenere queste informazioni è stato sviluppato il seguente endpoint sul nostro servizio:

1. **GET** `/eutran-gnrc-cells?id_enodeb=unique_id`

Questo endpoint riceve l'identificativo univoco di un eNodeB attraverso un *request-param*. Dopo la validazione del request-param il servizio alloca un client Http che inoltra la richiesta al controllore RAN. Questi restituisce una response con status code 200 "OK" contenente nel body un JSONArray di oggetti associati a ogni cella sotto il controllo dell'eNodeB indicato. Il body della response Http viene infine validato dal Controller Service e se quest'ultima operazione si conclude con successo, i dati ricevuti dal controllore RAN vengono inoltrati al web server nuovamente per il *rendering* sull'interfaccia web.

L'ultima funzionalità, precedentemente elencata, consiste nel poter creare una campagna MDT su un insieme di celle coordinate da un eNodeB. L'insieme delle celle

può comprendere tutte le celle controllate dall'eNodeB o un sottoinsieme di esse. L'endpoint, delegato a svolgere questa funzionalità, riceve due timestamp che indicano la data e l'ora in cui la campagna MDT dovrebbe iniziare e concludersi. L'ultimo parametro ricevuto è un identificativo dell'utente che sta utilizzando l'interfaccia web, nello specifico si tratta dell'email di registrazione sul web server. Si intuisce che l'insieme delle celle, i timestamp e anche l'identificativo sono dei dati che sono reperiti dall'interfaccia web attraverso l'iterazione con l'utente che ha effettuato il login sul web server.
Per creare una campagna MDT è necessario contattare il seguente endpoint esposto dal Controller Service:

1. **POST** `/mdts`

Questo endpoint riceve le precedenti informazioni e alloca un client Http che esegue una richiesta al controllore RAN. Se tutto va a buon fine ed è possibile registrare la campagna MDT per l'utente indicato, il controllore risponde con una response con status code 201 "CREATED".

Per ultimare la presentazione del Controller Service si procede ad analizzare i file di configurazione di questo servizio. Il primo file di configurazione è "application.properties", le cui proprietà più importanti sono: **spring.application.name** settata con la stringa "controller-service", **server.port** settata con l'intero 7500 e **management.endpoints.web.exposure.include=*** che permette al servizio di esporre tutti gli endpoint sviluppati.
Il successivo file di configurazione è "log4j.properties" che permette di realizzare un sistema di logging utilizzando la libreria Log4j di Apache.

La sezione successiva ha l'obiettivo di chiarire il processo di validazione dei dati provenienti dal controllore RAN e di valutare gli strumenti e le tecnologie impiegate per realizzare questo processo di validazione.

3.3.1 Validazione

La validazione o convalida dei dati è un processo che verifica l'integrità e la validità dei dati immessi nel software e nelle singole componenti. La convalida dei dati garantisce che i dati siano conformi ai requisiti e ai parametri attesi. Questo processo prova a garantire che i dati inviati alle applicazioni siano completi, sicuri e coerenti impiegando regole di validazione dei dati che controllano regolarmente i dati processati. Queste regole sono generalmente definite in un schema, in un dizionario o implementate tramite software delegato alla validazione dei dati. Il Controller Service, discusso precedentemente, instaura una comunicazione con il controllore RAN e attraverso questa vengono trasportati dati da una componente all'altra. Poiché il Controller Service riceve dei dati dall'esterno dell'architettura a

microservizi, sorge la necessità di definire un data model e di effettuare la validazione dei dati ricevuti dal controllore RAN.

Per definire il data model e abilitare la validazione di questo, si sono esaminati diversi linguaggi di modellazione dei dati. Il primo è stato il linguaggio YANG, documentato nell'RFC 6020. Questo è utilizzato per modellare la configurazione di apparati di rete gestiti dal protocollo di configurazione di rete NETCONF. Questo linguaggio è stato presto abbandonato per diversi aspetti negativi elencati di seguito:

- Necessità di introdurre i software ONOS o OpenDaylight. Questi sono due software *open source* ampiamente utilizzati per realizzare un sistema SDN. Si sottolinea che sono dei progetti molto vasti e dalle molteplici funzionalità. Tuttavia questi progetti sono risultati gli unici a fornire dei parser per la validazione del data model YANG, ma gli esempi e la documentazione a disposizione per il loro utilizzo è quasi inesistente. Utilizzare questi parser implicava introdurre una grossa complessità nel progetto, complessità derivata dall'utilizzo di software molto vasto e inoltre poco documentato per il nostro specifico uso. Ciò avrebbe inciso negativamente sulle tempistiche per realizzazione uno strumento efficiente di validazione dei dati.

- L'architettura sviluppata utilizza per i dati un formato JSON, infatti i Document contenuti nello Storage Service, sono dei JSON file. Pertanto utilizzare il YANG significava introdurre un nuovo data model nell'architettura, differente da quello utilizzato nel servizio di storage e destinato unicamente alla comunicazione con il controllore di basso livello.

- Il data model YANG viene ampiamente utilizzato per la configurazione degli apparati di rete con i protocolli di trasporto NETCONF e OpenDaylight. Nella nostra architettura a microservizi e nel livello sottostante del controllore non sono impiegati questi protocolli di trasporto.

- Il framework Spring, impiegato per lo sviluppo agile dell'architettura a microservizi, utilizza un sistema di annotazioni e di validatori per eseguire una validazione dei dati in fase di *binding*. Con l'utilizzo del data model YANG e di parser specifici per la validazione, questo sistema ottimizzato del framework Spring sarebbe poco utilizzato o addirittura inutilizzato.

Conclusa questa valutazione del data model YANG con esito negativo, si è valutato il sistema di annotazioni e validatori fornito dal framework Spring. Questo sistema è molto efficiente quando si desidera validare i request-param o quando si effettua il data binding nell'oggetto di una classe. In quest'ultimo caso si introducono delle annotazioni specifiche sopra gli attributi nella definizione della classe. Queste annotazioni esprimono delle regole di validazione che vengono verificate nella fase

di binding. Si intuisce che le annotazioni sono delle regole molto limitate, esempio: valore non nullo per un attributo, la lunghezza di una stringa e il range di un intero. I validatori invece consistono in metodi chiamati automaticamente dal framework sempre nella fase di binding o manualmente per validare il risultato di un'operazione. Questi permettono di verificare che un oggetto è un'istanza di una classe e fare degli ulteriori controlli sugli attributi. I validatori forniscono una validazione più completa rispetto alle annotazioni, pertanto questi sono stati utilizzati per validare i request-param nei vari endpoint esposti nell'architettura a microservizi. Tuttavia risultano uno strumento di validazione di base e lungo da sviluppare per dei controlli precisi e specifici. Dettò ciò, si è ritenuto opportuno valutare un ulteriore strumento di validazione che fosse più rapido da sviluppare rispetto ai validatori e che evitasse la complessità del YANG e di un altro data model. La scelta è stata il **JSON Schema**, questo sistema di validazione è molto recente, è documentato in vari draft IETF e se ne attende un successivo nei primi mesi del 2019. Pertanto si tratta di una nuova tecnologia che si sta evolvendo per diventare uno standard. Consiste in un dizionario di regole e parole chiavi che consentono di validare i documenti JSON. Si riporta di seguito il JSON Schema file utilizzato per la validazione dell'elenco degli eNodeB.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": " Array of ENBFunction",
  "description": "JSON schema representing an Array of ENBFunction object",
  "definitions": {
    "ENBFunction": {
      "description": "JSON schema representing an ENBFunction object",
      "type": "object",
      "properties": {
            "id": {
              "description": "The unique identifier for a eNodeB function
                  (es. AL01)",
              "type": "string",
              "minLength": 1
            },
            "cells": {
              "description": "list of Cells id",
              "type": "array",
              "items": {
                "type": "string",
                "minLength": 1
              },
              "minItems": 0,
              "uniqueItems": true
            },
```

```
      },
      "required": ["id","eNBID","cells","f_type","status"]
    }
  },
  "type": "array",
  "items": {
    "$ref": "#/definitions/ENBFunction"
  },
  "minItems": 0,
  "uniqueItems":true
}
```

Per semplicità sono state omesse alcune definizioni all'interno della sezione "properties", al fine di limitare il frammento di codice e focalizzare l'attenzione sulle sezioni più importanti. Nella sezione "definitions" viene presentato un oggetto eNodeB, segue la sezione "ENBFunction" contenente la definizione di ogni singolo attributo dell'oggetto eNodeB. Per ogni attributo, come "id" o "cells", viene definito un tipo e delle regole di validazione come: lunghezza minima di una stringa, numero minimo di elementi di un array e univocità di ogni elemento. Conclusa la sezione "properties", segue la sezione "required", un array dove sono specificati gli attributi dell'oggetto eNodeB che sono obbligatori e non possono mancare. Infine conclusa la definizione dell'oggetto eNodeB, si prosegue a definire un array composto di eNodeB, con numero minimo di elementi uguale a zero e specificando che ogni elemento ha un id univoco.

Un ulteriore JSON Schema è stato definito per effettuare la validazione dell'array contenente le celle sotto un eNodeB, restituito all'interno di una response Http dal controllore RAN alla ricezione di un id univoco per un eNodeB.

Il JSON Schema è risultato un ottimo strumento per la validazione dei JSON file, è rapido da impiegare perché la sua sintassi è molto simile a quella dei JSON file e per questo motivo si riducono drasticamente i tempi di apprendimento che una nuova sintassi richiederebbe come nel caso del data model YANG. In aggiunta questo strumento è ampiamente documentato all'indirizzo: `https://json-schema.org/`. In questa risorsa è possibile usufruire di un tutorial per utilizzare velocemente questo strumento e di una panoramica sugli obiettivi futuri che la comunità di standardizzazione, insieme con l'IETF, vuole raggiungere.

Per ottimizzare la validazione dei dati, si è scelto di utilizzare il JSON Schema e i suoi parser all'interno dei validatori del framework Spring, in questo modo è stato possibile integrare perfettamente un sistema di validazione più accurato con i meccanismi standard del framework.

3.3.2 Conclusione

L'obiettivo principale di questo capitolo è stato analizzare lo sviluppo dell'architettura di *back-end* in stile microservizi. Si ricorda che è stato utilizzato il protocollo OAuth 2.0 per realizzare un sistema di controllo degli accessi impiegando il token "Self-encoded" JWT. Successivamente si è sviluppato un servizio per instradare le richieste e validare il token. Infine, è stato esaminato il servizio che permette di monitorare e scoprire i servizi attivi nell'infrastruttura. Con la seguente immagina si desidera completare questa analisi e introdurre il Servizio di Orchestrazione, l'ultimo servizio sviluppato, che svolge funzioni di orchestrazione cooperando con il resto dell'architettura a microservizi ed espone un'interfaccia grafica utilizzando un web server sviluppato con l'ausilio del framework Django.

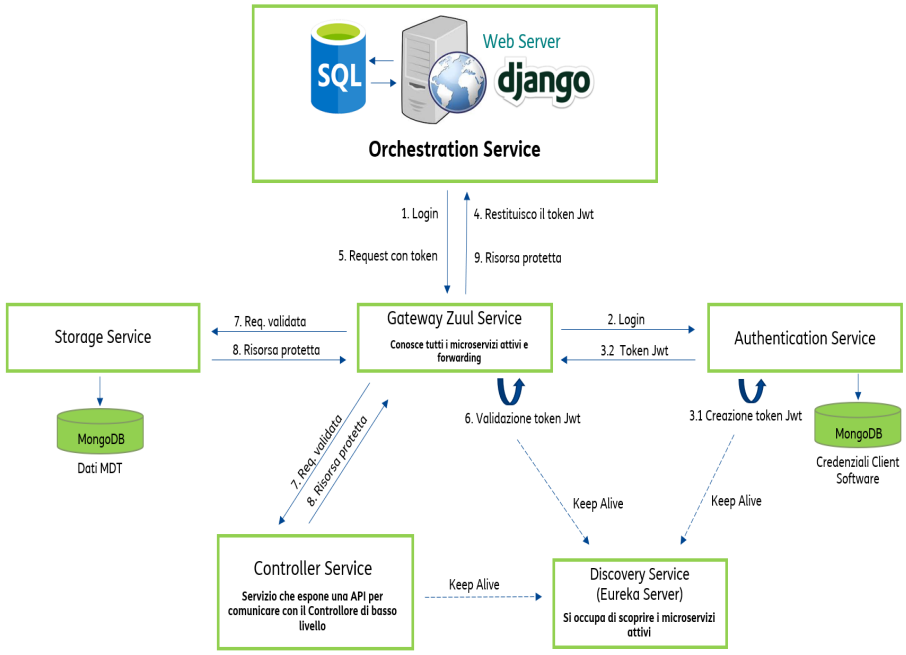

Figura 3.14. Architettura completa dei microservzi di back-end

Capitolo 4

Orchestration Service

4.1 Premessa

In questo capitolo si analizzerà l'ultimo servizio introdotto nell'architettura a microservizi. L'Orchestration Service è composto da due componenti:

- Un'API necessaria per comunicare con il resto dell'architettura a microservizi.
- Un Web Server che permette di effettuare il rendering dei dati ricevuti dall'architettura di back-end sull'interfaccia web.

Dato che il servizio di orchestrazione, per svolgere le sue funzioni, necessita di interagire con i vari servizi che compongono l'architettura di back-end è stato sviluppato uno strato di API per interagire con il resto dell'architettura a microservizi. L'Orchestration Service contatta l'Authentication Service per effettuare il login e ottenere il token JWT. Acquisito il token, il servizio di orchestrazione è autorizzato a contattare lo Storage Service per leggere i dati sulle campagne MDT e il Controller Service per poter comunicare con il Controllore RAN di basso livello. Si sottolinea che queste interazioni avvengono con l'ausilio del Gateway Zuul Service che svolge il controllo degli accessi e inoltra le richieste tra i vari attori.

4.2 Web Server

La componente che si analizzerà in questa sezione è il Web Server, essa riceve i dati dall'infrastruttura di back-end e li elabora per effettuare il rendering sull'interfaccia grafica. Inoltre gestisce le sessioni Http, molteplici tab e gli utenti iscritti alla pagina web. Dunque il Web Server svolge le funzioni di *front-end*, elaborando i file html e javascript necessari per il rendering sull'interfaccia grafica. Per sviluppare questa componente si è utilizzato un framework che semplificasse lo sviluppo

e permettesse di raggiungere risultati ottimali. Si sono valutati i prodotti disponibili e quale fosse il più adeguato al nostro progetto. Si sono analizzati diversi aspetti come: la documentazione, il supporto per i database, i sistemi di cache, l'autenticazione, ecc. Tra i framework disponibili si sono esclusi quelli in linguaggio PHP, ritenuti poco efficienti e non abbastanza moderni. La tabella 4.1 riassume gli aspetti valutati e i prodotti presi in considerazione.

Dall'analisi svolta è risultato che i framework Django e Ruby on Rails sono ugualmente validi e molto completi, unica differenza il linguaggio di sviluppo: il primo in Python, il secondo in Ruby. Il framework Express.js è un prodotto molto efficiente e valido in un'architettura software realizzata totalmente in javascript, un esempio è l'architettura MENA composta da MongoDB, Express.js, Node.js e Angular.js. Purtroppo la nostra architettura a microservizi è sviluppata in Java e pertanto questo prodotto non era adatto all'infrastruttura sviluppata. L'ultimo framework valutato è stato Phoenix, un prodotto con una comunità più piccola e non totalmente completo, ma molto veloce ed efficiente. Tuttavia si è escluso questo prodotto per le lunghe tempistiche d'installazione e di configurazione per un nuovo progetto e per la necessità di introdurre ulteriori componenti esterne per avere il supporto completo del framework. Concludendo, la scelta è ricaduta sul framework Django, questo è molto completo e ampiamente documentato al pari di Ruby on Rails ma è stato preferito a quest'ultimo per il linguaggio di sviluppo. Infatti il framework Ruby on Rails utilizza il linguaggio Ruby, questo è un linguaggio di programmazione orientato agli oggetti, diffuso principalmente nello sviluppo di applicazioni web. Django utilizza invece il linguaggio Python, questo è un linguaggio di programmazione orientato agli oggetti, ma ampiamente diffuso in tutti i settori di sviluppo. Pertanto si è ritenuto più importante imparare ad utilizzare il linguaggio Python per sviluppare il Web Server, maturando delle nuove conoscenze professionali facilmente impiegabili in altri settori di sviluppo.

Successivamente si è creato un progetto Django utilizzando l'IDE PyCharm della suite JetBrains. Questo progetto costituisce il Web Server che espone l'interfaccia Web per un amministratore di rete. Per la realizzazione dell'interfaccia si è utilizzato Bootstrap, un toolkit open source per lo sviluppo con HTML, CSS e JavaScript. Con Bootstrap è possibile utilizzare un'ampia libreria di componenti front-end per realizzare progetti reattivi e mobili sul Web.

Si è realizzata un'interfaccia Web che avesse un sistema di registrazione e di login. Gli utenti completano un form con alcune informazioni personali come: nome, cognome, password e ecc. Eseguita questa registrazione, l'utente viene rediretto alla pagina di login dove, inserendo le sue credenziali, può accedere alla console di comando. Per sviluppare questo sistema di registrazione e di login è stato utilizzato un database SQL. Si è scelto di utilizzare il database relazionale MySql che è

	Django	Express.js	Ruby on Rails	Phoenix
Linguaggio	(Python)	(Javascript)	(Ruby)	(Elixir) VM di Erlang
Documentazione	Ampia, chiara e con un tutorial.	Soltanto documentazione del codice.	Buona e guida all'installazione iniziale.	Buona
Open Source	Si	Si sotto licenza MIT. Ma è un mini-framework,semplice e che lascia molte implementazioni libere. Offre molte funzioni dal routing all'output JSON integrato. Integrato in framework più completi come Feathers, MEAN o Blueprint.	Si	Si, progetto GitHub.
Community	Enorme, prodotto continuamente aggiornato e revisionato.	Ampia e attiva.	Oltre 4500 persone.	Più piccolna 600 utenti.
Time to start	Davvero breve.		Molto veloce, i progetti hanno tutti la stessa struttura creata automaticamente con il comando Rails Struttura MVC. Un file chiamato "routes.rb" per fare associazioni tra funzioni e URL.	Lungo, si devono installare diversi componenti anche Node.js per la gestione di un client (incluso nel progetto) di tipo javascript.
Database	ORM + Api Python generata al volo per lo specifico database Interfaccia amministartore Add ,change and update + "template inheritance" ereditarietà tra i template. Se Usiamo MongoDB perdiamo questi vantaggi	No interfaccia di amministarzione. Supporto a MongoDB,si avvicina molto a questo perché MongoDB si basa sul javascript. Si inserisce nell'architettura MEAN , Insieme a Angular.js sul Browser e Node.js.	Supporta MongoDB dall'ultima versione, non è nativamente pensato per farlo. Nessun file xml o di mapping , paradigma DRY tutto solo una volta.Non ripetere nulla.	Utilizza Ecto per avere il supporto a MongoDB e Sql.
Authenticazione	Si, supporto specifico del framework		Si, molti strumenti come JWT o Recovery tramite email.	
Cache	Realizzata con Memcached che è un server di caching interno.	No	Si, ultima versione.	
Supporto Really Simple Syndacation	Si	No	Si	
Sicurezza da XSS	Si	Poco supporto di sicurezza, necessario il supporto di Helmet(protegge dalle vulnerabilità web note).	Si	Sono necessari componenti esterni.
Sicurezza da CSRF	Si	Suo nativo no ,ma è costruito su una strutta a middleware dove ciò che serve si aggiunge,sono massimo una trentina.	Si	
Sql injection (qualora si utilizzasse sql)	Si		Si	
Protezione da clickjacking	Si		Garantiscono aggiornamenti e patch soltanto delle ultime versioni e se necessario bisogna fare upgrade.	Si, ultima E14versione.
Host header validation	Validazione dell'host per negare la falsificazione della richiesta cross-site.			
Grossi Provider che lo usano	Instagram, Disqus, Pinterest, Mozilla.	Paypal l'ha usato mettendo in evidenza un guadagno di 200ms rispetto ad un'architettura interamente java (hanno utilizzato tutto lo stack javascript, troppo oneroso per i nostri obiettivi).	Basecamp, GitHub, Shopify, Airbnb, Twitch, SoundCloud, Hulu, Zendesk, Square, Highrise, Cookpad .	Inverse, Outline, Podium, Voicelayere Heroic Labs, Bleacher Report.
Scalabile	Django utilizza un'architettura "shared-nothing", che significa che è possibile aggiungere hardware a qualsiasi livello.	Un oggetto router è un'istanza isolata di middleware e le rotte sono gli End Point. Questo può essre considerato come una "mini- applicazione" capace solo di eseguire funzioni middleware e routing. Ogni applicazione Express ha un router app incorporato.	La soluzione è aggiungere hardware (scalare verticalmente) o (scalare orizzontalmente) con un load balancer e server replicati.	
Mobile	No	Si	Si	Si
Supporto crittografico	Si con le librerie Python.	No, forse con qualche modulo aggiuntivo.	Si, ma usano il concetto di credenziali che è nuovo.	
Configurazione	Si è necessaria		Non è necessaria, lo rende più veloce nell'inizializzazione.	

Figura 4.1. Tabella valutativa dei framework per lo sviluppo del Web Server

composto da un client da riga di comando e un server. Entrambi i software sono disponibili sia per sistemi Unix e Unix-like sia per Windows. In fase di sviluppo è stato utilizzato un client aggiuntivo, chiamato phpMyAdmin, che consiste di un applicativo web scritto in PHP, che consente di amministrare un database MySQL o MariaDB. Con questo strumento è stato più facile visualizzare lo stato del db e le interazioni col il web server. Il database MySql è stato velocemente aggiunto al progetto grazie al semplice e completo supporto del framework Django. Infatti quest'ultimo è fornito di un file di configurazione chiamato "setting.py" che gestisce vari aspetti come: middleware, applicativi esterni, database, autenticazione, cache e ecc. Pertanto è stato necessario modificare poche righe di codice nella sezione DATABASES per abilitare il supporto al server MySql.
Si prosegue illustrando le varie sezioni che compongono l'interfaccia web sviluppata.

4.3 Interfaccia Web

La prima sezione presentata è la "Dashboard", qui l'utente viene rediretto dopo il login e svolge le prime interazioni per ottenere le informazioni sugli eNodeB impiegati sul territorio. Il seguente screenshot illustra la pagina "Dashboard" realizzata.

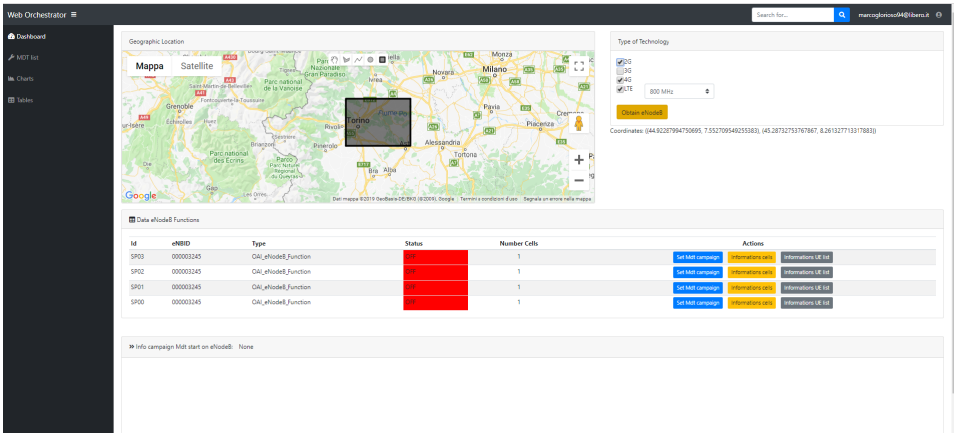

Figura 4.2. Sezione Dashboard

La richiesta per ottenere gli eNodeB viene inoltrata evidenziando un'area geografica d'interesse sulla cartina e selezionando i tipi di tecnologie con le checkbox in alto a destra. Eseguite queste scelte, col pulsante "Obtain eNodeB" è possibile inoltrare una richiesta al Web Server per ricevere la lista di eNodeB Function che soddisfano i requisiti precedenti e popolano la tabella "Data eNodeB Functions".

Ogni riga di questa tabella rappresenta un eNodeB. Ciascuna riga riporta le seguenti informazioni su un eNodeB: Id, eNBID, Type, Status e numero di celle. Nella parte destra di ogni riga troviamo invece le azioni che si posso eseguire su un eNodeB. Le azioni che si posso svolgere sono: Configurazione di una campagna MDT sull'eNodeB, richiedere informazioni sulle celle di un eNodeB e richiedere informazioni sugli UE collegati alle celle di un eNodeB. L'ultima azione è stata pensata, ma al momento non sviluppata infatti l'orchestratore dovrebbe conoscere i TMSI connessi alle celle tramite la core network attenendosi però alle normative sulla privacy del provider. Il pulsante per richiedere informazioni sulle celle di un eNodeB serve per visualizzare informazioni aggiuntive sulle singole celle e selezionare le celle su cui lanciare una campagna MDT. Premuto questo pulsante, viene inoltrata una richiesta al web server che risveglia l'architettura di back-end per contattare il controllore RAN e reperire le informazioni necessarie per popolare un'altra sezione dell'interfaccia web. Di seguito è riportato lo screenshot di questa sezione dedicata alle celle di un eNodeB.

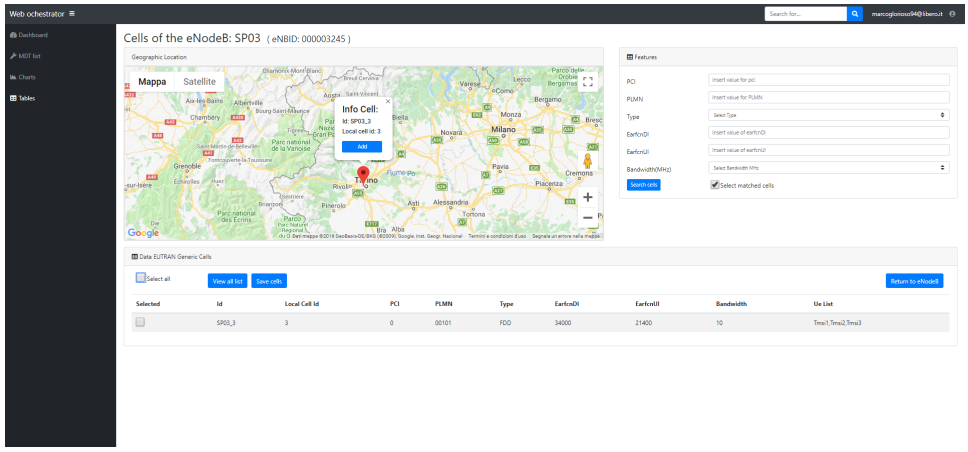

Figura 4.3. Sezione delle celle di un eNodeB

La sezione delle celle di un eNodeB è composta da una tabella, contenente una riga per ogni cella. Ciascuna riga fornisce informazioni tecniche sulla cella e contiene una casella cliccabile per selezionare la cella. Nella parte in alto a sinistra è possibile selezionare le celle in base alla loro posizione sul territorio. Infine nella parte destra troviamo un modulo in cui inserire dei parametri per visualizzare un sottoinsieme di celle dell'eNodeB. Sono abilitati tre pulsanti: il primo per salvare la lista di celle sui cui lanciare la campagna MDT dell'eNodeB, il secondo per visualizzare la lista completa di celle di un eNodeB e l'ultimo per ritornare alla pagina

Dashboard. Ritornando allo screenshot 4.2, il pulsante "Set Mdt campaign" abilita e permette di visualizzare la parte inferiore della pagina Dashboard, mostrando nella parte in basso a sinistra le celle precedentemente selezionate per un eNodeB, nella parte centrale un campo input in cui inserire data e ora d'inizio e fine della campagna MDT e il pulsante d'inizio che permette di notificare e registrare una nuova campagna MDT al controllore. Di seguito si riporta lo screenshot che mostra questa sezione per configurare una campagna MDT.

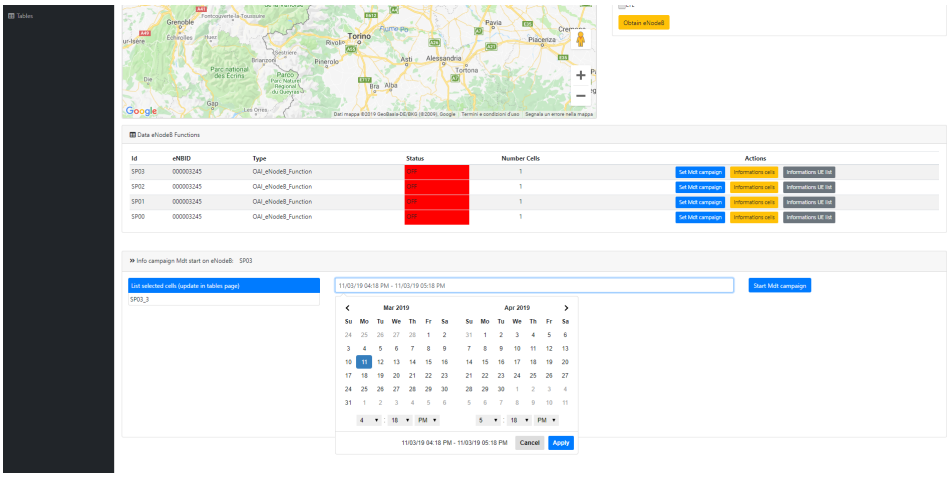

Figura 4.4. Configurazione di una campagna MDT

Una volta premuto il pulsante "Start Mdt Campaign" viene visualizzato un popup che mostra un riassunto della configurazione effettuata. Infine, se la richiesta è stata inoltrata con successo e la campagna è stata creata, l'utente riceve una notifica di successo.

L'utente può visualizzare lo storico delle campagne MDT programmate e concluse nella sezione "MDT List". In quest'ultima è possibile visualizzare una tabella, costituita da righe che rappresentano le singole campagne MDT. Su ogni riga è possibile leggere le seguenti informazioni:

1. Id univoco della campagna MDT
2. Stato corrente della campagna MDT
3. Numero di celle su cui la campagna MDT è registrata
4. Data e ora d'inizio della campagna

5. Data e ora di fine della campagna

Di seguito si riporta una screenshot che illustra la sezione "MDT List" contenente lo storico delle campagne MDT dell'utente.

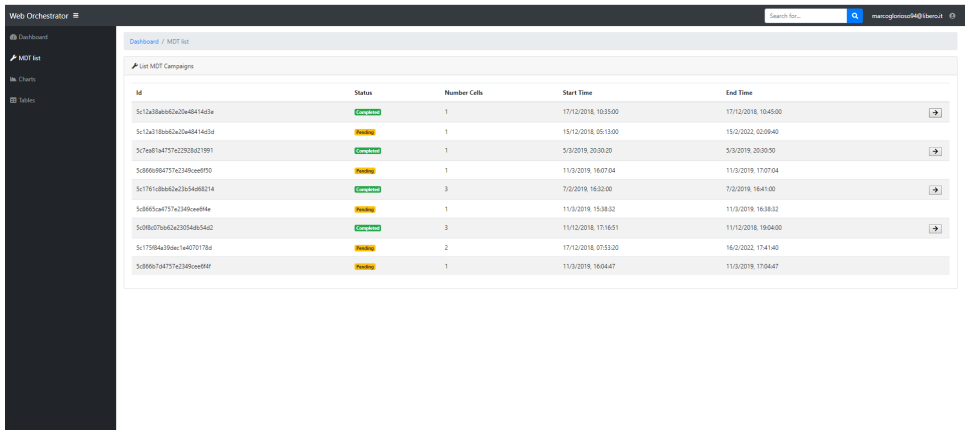

Figura 4.5. Sezione MDT List

Sono stati distinti due stati per una campagna MDT: "Pending" e "Completed". Il primo stato, con una etichetta gialla, indica che la campagna creata non è ancora conclusa o deve ancora iniziare. Lo stato "Completed", etichetta verde, indica che la campagna MDT è stata conclusa ed è possibile visualizzare i dati raccolti sulle celle durante la campagna. Le righe con lo stato "Completed" sono munite sul lato destro di una freccia che permette agi utenti di visualizzare i grafici della campagna MDT interessata.

La sezione "Charts" contiene i grafici risultati da una campagna MDT completata. I grafici sono identificati dalla coppia TMSI e id della cella base che ha ricevuto le misure. In particolare i grafici per ogni coppia sono due: il primo che rappresenta il valore di RSRP al trascorrere del tempo, il secondo che rappresenta il valore di RSRQ al trascorrere del tempo. Questi grafici contengono una sola linea se l'UE invia le misure solo alla cella base, ma anche più linee nel momento in cui l'UE invia le misure anche ad altre celle vicine. Le misurazioni inviate alle celle vicine, sono quelle presenti nella figura 3.6 sotto il campo "neighbors".

I grafici sono stati realizzati con la libreria javascrippt "Graph.js". Questa è una libreria open source semplice e chiara supportata da HTML5 per realizzare grafici animati e interattivi sulle interfacce web. Di seguito è riportato lo screen della sezione "Charts" dell'interfaccia web sviluppata.

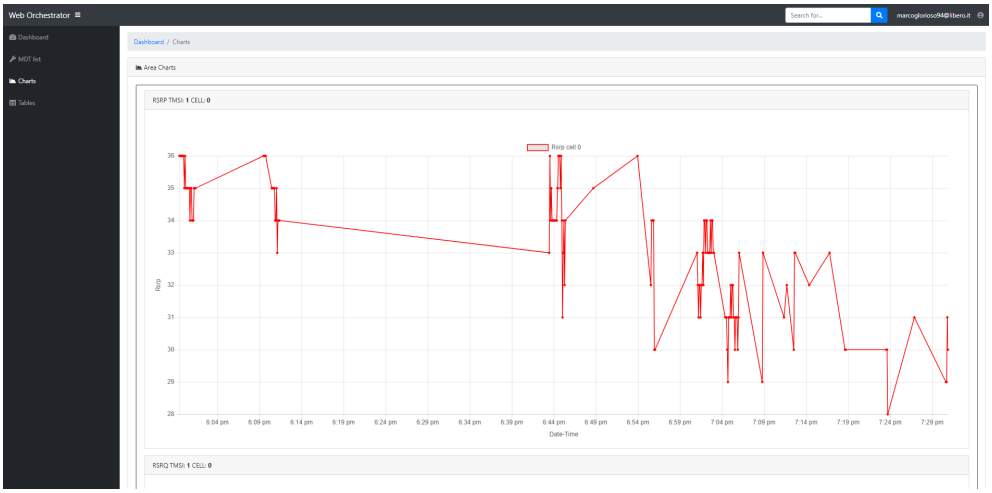

Figura 4.6. Sezione Charts

Questa sezione è l'ultima realizzata durante questo lavoro di tesi. L'interfaccia grafica analizzata sin qui, permette a un amministratore di rete di selezionare degli eNodeB specifici, di selezionare un insieme di celle per ciascun eNodeB e di programmare una campagna MDT su tali celle. Infine l'utente può osservare lo stato delle sue campagne e nel momento in cui una di queste è completata, usufruire dei dati raccolti. L'utente può godere di una rappresentazione grafica dei dati che può aiutarlo nell'interpretare i risultati e nel riconoscere anomalie di funzionamento delle celle impiegate. Quindi il sistema permette di monitorare la rete di accesso e in particolare le celle che la compongono, senza impiegare risorse umane o apparecchiature costose, utilizzando soltanto i dispositivi mobili connessi alla rete.

Si conclude l'analisi dell'interfaccia web analizzando la gestione delle sessioni Http e delle tab da parte del web server. Il framework Django supporta pienamente le sessioni e consente di archiviare e recuperare dati sul server. Il framework implementa il supporto alle sessioni attraverso una componente middleware. Per abilitare questo middleware si inserisce la stringa 'django.contrib.sessions.middleware.-SessionMiddleware' nella sezione "MIDDLEWARE" del file setting.py. Infine bisogna configurare dove memorizzare le informazioni delle sessioni. Per impostazione predefinita, Django memorizza le sessioni nel database con il modello 'django.contrib.sessions.models.Session'. Sono supportate anche altre impostazioni per memorizzare altrove i dati di sessione, per esempio sul file system o in una cache. Si è scelto di utilizzare l'impostazione predefinita e salvare i dati delle sessioni sull'istanza del database MySql precedentemente discussa. Per fare ciò, è stato necessario aggiungere 'django.contrib.sessions' nella sezione "INSTALLED_APPS" del file setting.py. Dopo aver configurato il framework, è stato necessario eseguire il comando "manage.py migrate" nella shell python. Questo comando ha permesso di creare la tabella che ospita i dati per le sessioni all'interno del server MySql. Con questa configurazione il web server è stato abilitato a supportare più sessioni Http in parallelo e di conseguenza più utenze che lavorano in parallelo. Si è proseguito ad abilitare l'apertura di più tab in parallelo sul browser per creare più finestre di lavoro per l'utente loggato. Il framework Django purtroppo non ha presentato una componente che fornisse tale servizio. Pertanto questa funzionalità è stata sviluppata interamente senza il supporto del framework. Per fare ciò, è stato necessario impiegare una funzione Javascript e il sessionStorage sul browser, viceversa sul web server è stato necessario abilitare due endpoint e modificare quelli delegati a restituire la vista al browser. Gli endpoint su cui viene contattato il web server sono tutti contenuti nel file view.py del framework Django, in questo file di solito si svolge il rendering delle pagine html. Con gli strumenti elencati precedentemente si è assegnato a ogni tab un id univoco chiamato "id_page", una volta ottenuto un id dal web server, questo viene salvato nel sessionStorage del browser e inviato come request param a ogni richiesta al web server. In questo modo il web server è in grado di creare istanze di lavoro separate per ogni *id* di ogni tab. Quindi i dati, le iterazioni dell'utente e le richieste di una tab sono circoscritte a essa sia sul browser sia sul web server, ma le tab di un utente loggato appartengono tutte alla stessa sessione Http.

Per ultimare l'analisi dell'architettura sviluppata si riportano di seguito due *sequence diagram* al fine di illustrare le iterazioni tra le varie componenti che costituiscono tutto il sistema sviluppato.

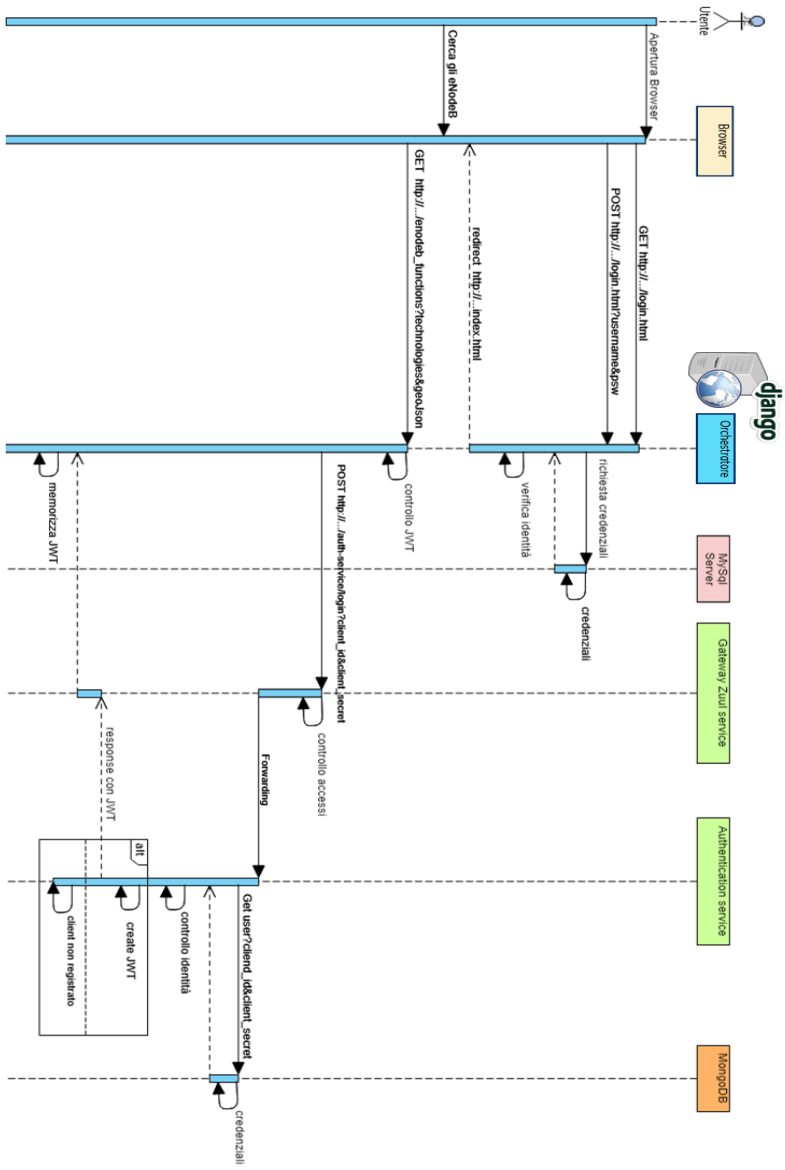

Figura 4.7. Sequence diagram per richiedere la lista degli eNodeB con JWT token.

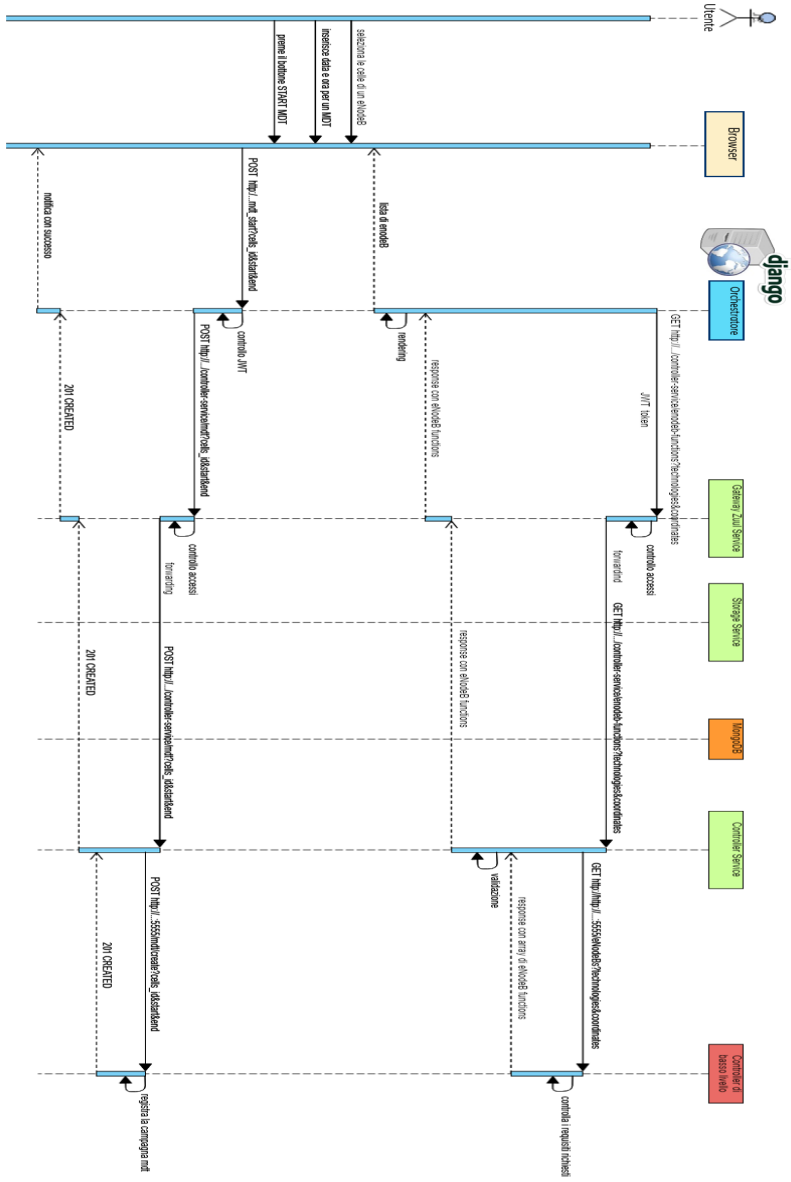

Figura 4.8. Sequence diagram per creare una nuova campagna MDT.

Capitolo 5

Conclusioni

Il lavoro svolto ha permesso lo sviluppo di un prototipo di orchestratore RAN in grado di monitorare la rete di accesso, impiegando la tecnologia dei Minimization of Drive Test(MDT). Questa tecnologia permette di ridurre l'OPEX dei drive test sul territorio e lo svolgimento di analisi più dinamiche, sia per quanto riguarda l'area geografica da analizzare, sia per l'intervallo di tempo durante il quale effettuare le misurazioni. L'architettura illustrata in questo lavoro di tesi coopera con il controllore RAN, che comunica con le celle di basso livello e con gli UE ad esse connessi. Quindi si è definita una comunicazione tra l'orchestratore e il controllore, sviluppando delle API REST e impiegando strumenti di validazione dei dati scambiati. Il sistema di orchestrazione sviluppato fornisce un'interfaccia grafica per un amministratore di rete, tramite questa è possibile inoltrare delle richieste al controllore RAN per eseguire delle funzionalità di alto livello. Le principali funzionalità sono: richiedere gli eNodeB Functions su un'area geografica e in base al tipo di tecnologia, richiedere le celle di un eNodeB e lanciare una campagna MDT sulle celle di un eNodeB per monitorare il loro funzionamento per un intervallo di tempo. L'orchestratore mette a disposizione dell'utente degli strumenti per monitorare la rete ad un livello più alto, distaccandosi dalla molteplicità e dalla varietà delle tecnologie impiegate nella rete. L'interfaccia Web consente di rilevare i cambiamenti della rete attraverso delle richieste inviate periodicamente al controllore per mantenere l'interfaccia sempre aggiornata. Infine i valori delle misurazioni ricevute dal controllore sono raccolti e rappresentati graficamente, in questo modo è possibile analizzare il funzionamento delle celle tramite gli UE impiegati sulla rete. Il sistema realizzato supporta due valori di misurazione RSRP e RSRQ.

Questo lavoro di tesi si pone come prima risposta all'esigenza di impiegare funzioni di orchestrazione nella rete di accesso. Infatti, con l'avvento delle tecnologie di rete di quinta generazione, assisteremo a procedure di densificazione della rete, cioè ad

un incremento del numero delle cellule dispiegate sul territorio. Pertanto il prototipo sviluppato svolge un primo use case di monitoraggio, permettendo di raccogliere dati di funzionamento della rete di accesso composta da un numero elevato di celle e mette a disposizione i dati raccolti per ottimizzazioni e predizioni future, impiegando possibili tecnologie di AI e ML. Inoltre l'orchestrazione diventa necessaria per automatizzare i processi di pianificazione e ottimizzazione delle risorse e per offrire le migliori prestazioni alle varie applicazioni. Questi processi si traducono con il concetto di Network Slicing, cioè la capacità di saper creare più reti virtuali (network slice) sulla stessa infrastruttura fisica. Il Network Slicing rivestirà un ruolo importante nell'avvento delle nuove applicazioni 5G, che necessitano di requisiti di funzionamento differenti. La figura 1.1 mostra una panoramica della varietà di tali applicazioni e dei requisiti funzionali di cui necessita ciascuna di esse.

5.1 Sviluppi successivi

Si desidera concludere questo elaborato evidenziando i possibili sviluppi successivi. Si riporta di seguito un elenco di essi:

1. Migrazione da VM a Container(es. Docker).
2. Sviluppo e integrazione di ulteriori use case: es. supporto di algoritmi AI/ML.
3. Adeguamento dell'architettura alle interfacce definite dall'O-RAN Alliance.
4. Test di carico dell'infrastruttura realizzata.

Del progetto sviluppato è stato eseguito il *deployment* su una VM nel laboratorio di TIM presso cui si è svolto il lavoro di tesi. Negli ultimi anni la virtualizzazione tramite Container ha permesso di realizzare degli applicativi che funzionano rapidamente e in modo affidabile da un ambiente di elaborazione ad un altro. Il software virtualizzato tramite i Container funzionerà sempre allo stesso modo, indipendentemente dall'infrastruttura sottostante. I Container realizzano una virtualizzazione leggera infatti non richiedono un sistema operativo per applicazione, aumentando l'efficienza dei server e riducendo i costi di server e licenze. Pertanto è auspicabile una futura virtualizzazione più leggera impiegando la tecnologia dei Docker, la quale si pone come standard nel settore dei Container e permette di realizzare degli applicativi totalmente portatili.

Nell'infrastruttura sviluppata è stato adottato il modello dell'architettura a microservizi, che è un approccio allo sviluppo di una singola applicazione come insieme di singoli servizi. Ognuno di questi è una *black-box*, che espone un'API astraendo

rispetto al dettaglio di come le funzionalità sono implementate. Per l'isolamento e la modularità di ogni servizio, il sistema si presta bene a successivi aggiornamenti, ad eventuali *rollback* e all'ingresso di nuovi servizi nell'architettura. Pertanto è auspicabile sviluppare nuovi servizi che realizzino ulteriori funzioni di orchestrazione. Ad esempio, sarebbe possibile sviluppare un servizio che ospita algoritmi di AI e ML. Questo servizio potrebbe allenare dei modelli sui dati memorizzati nel servizio di storage e restituirli al servizio di orchestrazione che li impiegherebbe per eseguire ulteriori use case di orchestrazione.

In questo elaborato si è fatto riferimento più volte all'O-RAN Alliance. Attualmente questo ente lavora per standardizzare l'interfaccia tra il livello di orchestrazione e il controllore RAN(A1) e le varie interfacce tra il controllore e i livelli più bassi(E2, E1, F1). Detto ciò, sarebbe opportuno adeguare l'architettura sviluppata alle future specifiche delle interfacce pubblicate dall'O-RAN Alliance.

Da ultimo si riconosce la necessità di svolgere un test di carico sull'infrastruttura sviluppata, in particolare sull'architettura a microservizi e sull'interfaccia web.

Bibliografia

[1] O-RAN: Towards an Open and Smart RAN.

[2] "What does orchestration really mean?" by Lukasz Mendyk, March 2016,https://inform.tmforum.org/features-and-analysis/2016/03/what-does-orchestration-really-mean/

[3] Minimization of Drive Tests (MDT) in Mobile Communication Networks by Daniel Baumann.

[4] Technical Specification Group Radio Access Network. Study on Minimization of drive-tests in Next Generation Networks. 3GPP TR 36.805 Version 9.0.0, 3GPP, Dezember 2009.

[5] Network Slicing Architecture draft-geng-netslices-architecture-02 by IETF, July 3, 2017.

[6] DigiRAN:il valore dell'automazione nell'accesso radio pubblicato da Tim,https://www.telecomitalia.com/tit/it/notiziariotecnico/edizioni-2018/n-1-2018/capitolo-6.html

[7] Top 5 Considerations When Evaluating NoSQL Databases, by MongoDB, February 2015.

[8] "A performance comparison of Sql and NoSql database by Yishan Li and Sathiamoorthy Manoharam".