
Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

# Towards security automation in Open Source MANO

**Relatori**
prof. Fulvio Valenza
prof. Riccardo Sisto
dott. Daniele Bringhenti

**Candidato**
Matteo Nifosì

Anno Accademico 2023-2024

# Sommario

Lo sviluppo delle reti è stato rivoluzionato dall'introduzione di tecnologie di virtualizzazione al mondo delle reti quali NFV e SDN che permettono di realizzare reti dinamiche e flessibili tramite una separazione tra hardware e software. Le funzioni virtualizzate tramite NFV possono essere connesse tra di loro in modo arbitrario potendo realizzare servizi complessi in base alle proprie esigenze. La gestione di queste catene di funzioni viene effettuata da degli orchestratori NFV, tra i quali sono presenti Open Source MANO e Openstack Tacker. Gli orchestratori semplificano di molto la gestione dei servizi automatizzando non solo la loro creazione ma anche le diverse operazioni del ciclo di vita che risulterebbero complesse e ripetitive nel caso dovessero essere effettuate da un'operatore umano.

Il numero crescente di attacchi informatici nel corso degli anni rende obbligatoria l'adozione di forme di sicurezza necessarie per non compromettere le funzionalità del servizio di rete che si vuole fornire. Molto spesso la definizione della sicurezza all'interno di una rete viene svolta da un 'amministratore di rete o un altro operatore specializzato, richiedendo quindi un intervento umano che rende l'intero processo soggetto a potenziali errori che potrebbero solo incrementare la vulnerabilità dell'intero sistema. Con l'idea di risolvere questo problema è stato sviluppato Verefoo, framework che permette di definire la sicurezza di una rete in maniera automatica.

L' idea alla base di questo lavoro di tesi era andare a semplificare ulteriormente il processo di creazione di un servizio di rete gestito da parte di un'orchestratore delegando la definizione della sicurezza, secondo specifici requisiti, a Verefoo, il cui risultato viene poi raccolto per creare il grafo reale del servizio con inclusa la configurazione di sicurezza desiderata.

La parte centrale della tesi è stata la realizzazione dell'interazione tra verefoo e OSM, effettuata cercando di riportare correttamente le funzioni presenti nel grafo prodotto da verefoo all'interno dell'orchestratore secondo le strautture dati da esso definiti e i comandi forniti, stesso discorso per il servizio nel suo complesso. Per questo motivo la parte finale della tesi contiene una prima parte per illustrare i procedimenti necessari per la creazione delle strutture che rappresentano funzioni e servizi di rete internamente a OSM, ovvero vnf package e ns package, concludendo con una semplice demo che mostra come creare un grafo definendo i collegamenti tra le funzioni di cui questo è composto. Infine viene mostrata un'interazione effettiva tra OSM e verefoo rappresentando un grafo prodotto in output da verefoo all'interno dell'orchestratore.

# Indice

# Elenco delle figure

# Listings

# Capitolo 1

# Introduzione

Nell'evoluzione delle reti un ruolo importante è stato svolto dalla virtualizzazione, tecnologia che ha permesso il passaggio da reti basate su hardware specifico a reti implementate attraverso funzioni software eseguite su hardware generico, portando quindi numerosi vantaggi.

Oltre alla virtualizzazione ,le tecnologie di Software-Defined Networking e di Network Functions Virtualization hanno e stanno ancora contribuendo a rivoluzionare l'ambiente delle reti.SDN consiste nella separazione tra il data plane e il control plane permettendo l'utilizzo di dispositivi di rete più semplici che vengono gestiti da un controllore centrale, con il vantaggio di poter definire reti on-demand. NFV invece separa la parte software delle funzioni di rete dalla loro parte hardware, rendendole solo software in modo che possano essere eseguite su qualsiasi tipo di hardware, con le sufficienti caratteristiche.

Come integrazione alla tecnologia NFV è stato sviluppato il concetto di MANO (Management and Orchestration), ovvero un componente dell'architettura NFV che si occupa di gestire le risorse dell'infrastruttura per definire funzioni e servizi, in modo più o meno automatico. Tra i principali orchestratori NFV esistono Open Source MANO e il progetto Tacker di OpenStack.

In un mondo dove le reti assumono sempre più importanza non può venir meno l'implementazione di una, anche seppur minima, forma di sicurezza.Spesso la configurazione della sicurezza di una rete viene gestita in modo manuale da parte degli amministratori di rete che, non volendo, possono contribuire a rendere le reti più insicure, data la presenza di possibili errori di configurazione. Una soluzione a questo problema può essere l'eliminazione del contributo umano alla configurazione rendendo il tutto automatizzato.[1]

Per questo motivo , basandosi sulle tecnologie di NFV e SDN, è stato sviluppato Verefoo (VERified REFinement and Optimized Orchestration) , un framework che permette di calcolare la posizione ottimale delle funzioni di sicurezza all'interno di un grafo generico con funzioni di rete già presenti [2], si occupa della loro allocazione e della loro configurazione[3], secondo certi requisiti che gli vengono forniti come input.

Un passo importante è rappresentato dalla possibilità di fondere i due tipi di automatizzazione, la prima legata alla gestione delle funzioni, la seconda alla definzione di sicurezza in una rete, cercando di realizzare un'interazione tra orchestratori e Verefoo.[4]

## 1.1 Descrizione della tesi

La tesi è suddivisa nei seguenti capitoli:

- Capitolo 2 : in questo capitolo viene introdotto il concetto di virtualizzazione e i meccanismi principali che ne permettono la realizzazione. Dopo un'introduzione generale si parlerà di come la virtualizzazione ha influenzato lo sviluppo delle reti tramite la diffusione di concetti come SDN e NFV, che tramite la loro collaborazione permettono la realizzazione di reti dinamiche e flessibili.Rimanendo nell'ambito NFV si enunciano le caratteristiche principali dell'architettura NFV e del componente MANO, impiegato per l'orchestrazione di funzioni virtuali

- Capitolo 3 : questo capitolo descrive le caratteristiche principali di VEREFOO , partendo dalla descrizione degli input necessari al suo funzionamento , come Allocation Graph o Service Graph e i Network Security Requirements, analizzando poi il suo ciclo di lavoro e il risultato di quest'ultimo

- Capitolo 4 : questo capitolo contiene una breve descrizione del lavoro svolto nella tesi, presentando l'obiettivo principale del documento.

- Capitolo 5: in questo capitolo vengono analizzati in dettaglio i due principali orchestratori del mondo NFV ovvero Open Source MANO e Tacker di Openstack. Dopo averne introdotto i componenti e il funzionamento si cercherà di identificare una possibile interazione di questi con il framework Verefoo precedentemente analizzato

- Capitolo 6 : questo capitolo contiene una esemplificazione dell'interazione tra l'orchestratore OSM e VEREFOO portando un esempio pratico di creazione di un servizio di rete basato su di un grafo prodotto in output da verefoo, definendo le singole funzioni e i collegamenti tra queste

- Capitolo 7 : in questo capitolo finale si presentano delle possibili estensioni del lavoro svolto in questa tesi

# Capitolo 2

# Virtualizzazione

La virtualizzazione è una tecnologia che , partendo da una macchina fisica, utilizzando un software specifico, detto hypervisor, permette di astrarre le risorse fisiche e renderle disponibili ad entità virtuali dette virtual machines(VM).Una stessa macchina fisica può essere usata per eseguire applicazioni diverse e anche sistemi operativi diversi, in questo caso il sistema operativo in esecuzione sulla macchina fisica è detto sistema operativo Host, mentre quello in esecuzione all'interno della VM è detto sistema operativo Guest.

La tecnica della virtualizzazione è stata ideata per la prima volta da IBM e applicata per la prima volta negli anni '60 con l'obiettivo di partizionare e poter utilizzare in modo più efficiente le risorse dei mainframe di grandi dimensioni, l'hardware maggiormente diffuso all'epoca.Questo tipo di hardware era molto costoso per cui un miglior utilizzo delle sue risorse permetteva di trarre vantaggio da un minor numero di dispositivi avendo un risparmio non indifferente. Il partizionamento logico permetteva di distribuire le risorse fisiche a più macchine virtuali, migliorando l'efficienza dell'infrastruttura dando la possibilità di eseguire più applicazioni sulla stessa macchina fisica.

Negli anni successivi la virtualizzazione venne messa da parte principalmente a causa della diffusione del modello dell'elaborazione distribuita, costituito da applicazioni client-server e l'utilizzo di server economici e desktop x86 che, essendo molto convenienti ,divennero presto lo standard maggiormente adottato.Il loro utilizzo sempre più diffuso ha portato l'introduzione di problematiche da dover affrontare, come il basso utilizzo dell'infrastruttura dovuto all'esecuzione di una sola applicazione su una macchina, l'elevato costo dell'infrastruttura , caratterizzata da una maggiore complessità, e poca reattività nella risoluzione di guasti e problemi. Per affrontare queste problematiche le società di sviluppo software alla fine del secolo scorso cercarono di applicare la tecnologia di virtualizzazione ai nuovi sistemi x86.Queste macchine però non sono state progettate tenendo in considerazione la virtualizzazione che quindi non è supportata, a causa della presenza di istruzioni sensibili ma non privilegiate che possono essere eseguite da qualunque applicazione con qualsiasi livello di privilegi.Queste istruzioni non sono facilmente rilevabili dall'hypervisor e si rischia che la vm esegua operazioni di cui non dispone i corretti privilegi causando un problema di sicurezza.Tramite tecniche di diversa

natura come la Dynamic Binary Translation, che prevede la modifica a run-time del codice eseguito dal guest per evitare di causare un 'interruzione, e l'Hardware Assisted Virtualisation, nel quale si rendono tutte le istruzioni sensibili privilegiate evitando la fase di traduzione, è stato reso possibile utilizzare la tecnologia di virtualizzazione anche nelle macchine x86.

Uno dei componenti fondamentali della virtualizzazione è il software dell'hypervisor, anche detto Virtual Machine Manager (VMM), che ha il compito di raccogliere le risorse della macchina fisica su cui si trova, o anche di un'intera infrastruttura fisica, e renderle disponibili come risorse virtuali. Gestisce l'allocazione di queste risorse alle vm di cui viene richiesta l'esecuzione e controlla che ogni vm lavori all'interno del proprio spazio operativo senza influenzarsi vicendevolmente con altre vm. L'hypervisor può essere eseguito direttamente sull'hardware come unico software disponibile o ,come avviene di solito, esso può essere un componente del sistema operativo host.

Uno dei modi attraverso cui la virtualizzazione può essere ottenuta è l'uso della full virtualization basato su VM. Queste sono delle vere e proprie macchine caratterizzate però da componenti virtuali quali cpu, ram , storage e scheda di rete che la rendono a tutti gli effettivi una macchina funzionante.Una macchina virtuale che per qualsiasi motivo termina la propria esecuzione non deve in alcun modo influenzare l'esecuzione delle altre VM in esecuzione sulla stessa architettura. Tramite le VM si riesce a ottenere uno degli obiettivi principali della virtualizzazione ovvero l'isolamento tra diverse istanze.Si ha un certo livello di eterogeneità grazie alla possibilità di poter eseguire diversi tipi di sistemi operativi così come la possibilità di emulare hardware diverso da quello di partenza che aumenta il numero di applicazioni che si possono eseguire.

Le VM si scontrano però con alcuni requisiti del mondo data center che non possono essere ignorati:

- Utilizzo eccessivo delle risorse. Ogni VM possiede un proprio sistema operativo guest che occupa un spazio non indifferente e anche nel caso in cui due o più VM abbiamo lo stesso OS questo va ripetuto più volte tante quante sono le VM in cui va eseguito.Per poter far funzionare la VM inoltre il sistema operativo va eseguito.Quindi le VM sono onerose da punto di vista computazionale e di archiviazione, considerando soprattutto la presenza di più vm sulla stessa macchina il problema diventa più evidente.

- Tempo di avvio. Una delle caratteristiche richieste nei data center è la scalabilità dei servizi. Nel momento in cui il carico di lavoro aumenta è importante non solo poter reagire a questo cambiamento ma farlo con una certa velocità.Usando le VM purtroppo questa caratteristica viene meno.

- La presenza di un OS. La presenza di un OS diverso da quello host non è sempre necessario, anzi è meglio cercare di utilizzare un solo sistema operativo che può essere facilmente identificato in linux.

Per questo motivo si è cercato di sviluppare un nuovo modo di attuare la virtualizzazione ottenuto tramite la lightweight virtualization, virtualizzazione leggera. Dato che Linux era l'OS più diffuso nei data center sono state sfruttate delle

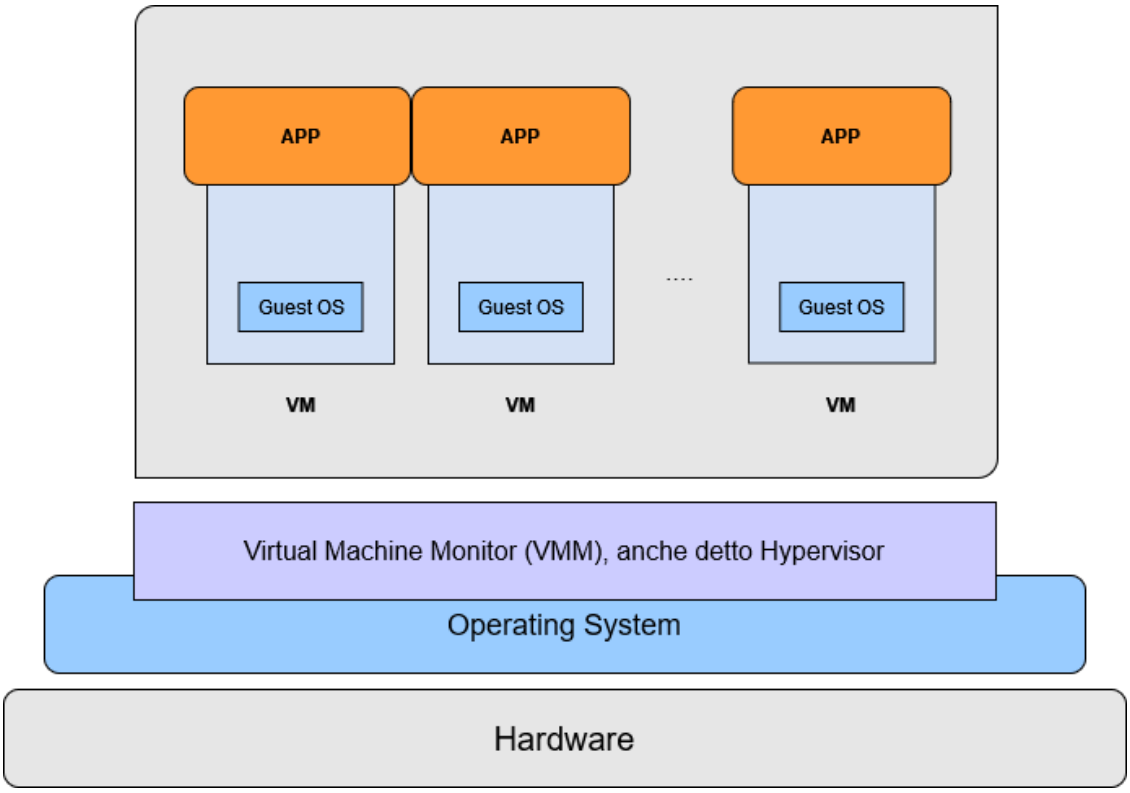


Figura 2.1. Struttura della virtualizzazione

funzionalità del suo kernel originariamente pensate per garantire isolamento tra i processi.Le funzionalità sono:

- Namespaces : feature del kernel linux che permette la creazione di ambienti virtuali separati in base a diverse risorse presenti, come computing, memory, networking, mount ecc.
- Cgroups : caratteristica del kernel linux per limitare,stimare, isolare o negare l'utilizzo delle risorse ad un processo o gruppo di processi, ad esempio impostare un valore massimo di memoria utilizzabile per un gruppo di processi

Unendo le due funzionalità appena descritte si è creato LXC (Linux containers) , unione di funzionalità del kernel linux che permette di isolare tra loro i processi e la creazione di *Container* completi.I Container permettono di implementare la virtualizzazione leggera, avendo le stesse proprietà della full virtualization senza le complessità da essa derivanti.Rappresentano un contenitore all'interno del quale sono presenti tutti i file necessari per l'esecuzione di un 'applicazione.I container non possiedono un sistema operativo proprio come le VM ma si basano su quello dell'host. Di conseguenza i container sono molto più veloci delle VM per quando riguarda avvio,terminazioni e altre operazioni legate al ciclo di vita, e più leggeri consumando meno cpu, meno memoria e senza l'overhead presente nella virtualizzazione. LXC presenta però alcune limitazioni tra le quali la più importante è la mancanza di portabilità, perché un container linux creato su un server non può essere portato su altri server.

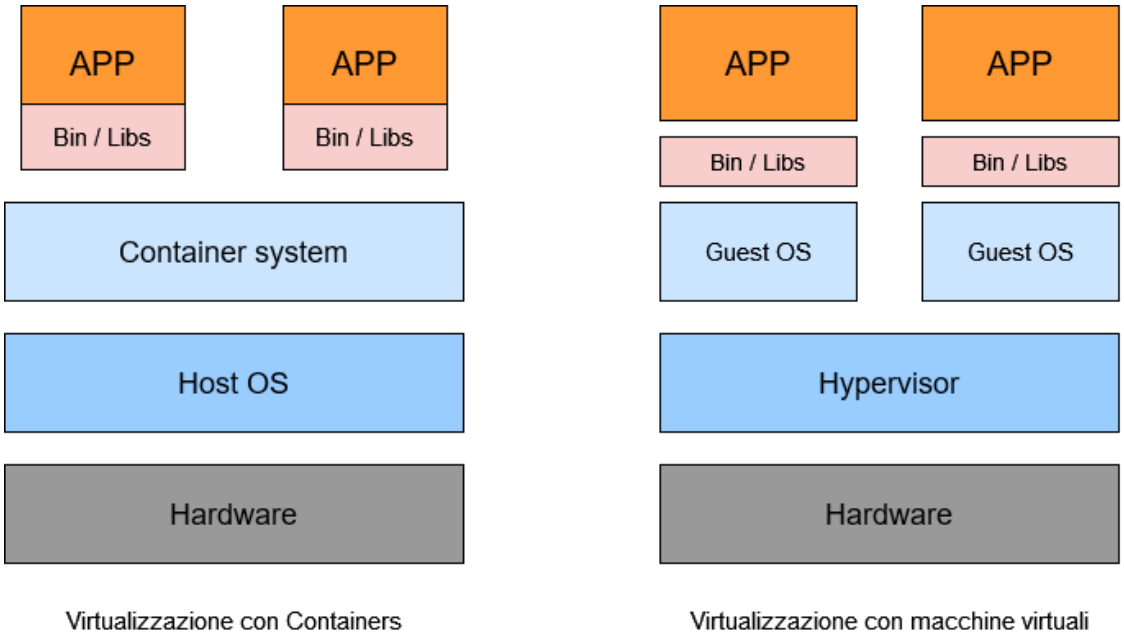


Figura 2.2. Struttura della virtualizzazione leggera

Come ulteriore tecnologia per la creazione di container si è diffuso Docker che ha avuto un notevole successo, nonostante si basi sulle stesse funzioni sfruttate da LXC , grazie al soddisfacimento della funzione di portabilità resa molto più semplice.Questa caratteristica è resa possibile grazie alla presenza di Repository online. Dopo aver creato un Docker questo può essere caricato in una repository così che possa essere scaricato da qualsiasi altra macchina ottenendo lo stesso docker funzionante. Anche la creazione del docker stesso è resa molto più semplice e immediata, infatti consiste nella compilazione di un Docker file, documento che contiene tutti i dati necessari per l'esecuzione del docker e dell'applicazione in esso contenuto, che è facilmente replicabile.I Docker, tramite l' introduzione del layered file system, possono condividere tra di loro porzioni di file risparmiando memoria.La mancanza di un forte isolamento tra container rimane l'unico dei problemi introdotti da questa nuova tecnologia, quindi docker non è la scelta migliore in caso di ambiente multi-tenant col rischio di collisione tra le funzioni.

## 2.1 Network Function Virtualization

All'inizio del ventunesimo secolo le reti dei service provider erano implementate solo tramite dispositivi hardware specializzati, costosi e acquistabili solo dai provider che ne effettuano la produzione. Nel caso si voglia inserire una nuova funzione o cambiare il posizionamento delle funzioni all'interno della rete, era necessario del tempo in quanto deve essere acquistato, ad un certo costo, il dispositivo che implementa la funzione richiesta , poi questa va inserita correttamente all'interno della rete e regolarmente configurata. Tutto questo richiede una manipolazione fisica della rete e l'azione di un operatore specializzato nell'utilizzo della funzione fisica.Quindi le

reti concepite nel modo classico non forniscono né flessibilità né dinamismo per la creazione di servizi. Queste caratteristiche sono però fondamentali al giorno d'oggi per poter fornire velocemente servizi su richiesta senza dover consumare tempo e risorse preziose. Le tecnologie che sono state sviluppate per soddisfare tali richieste sono la Network Functions Virtualization (NFV) [5] e Software Defined Networking (SDN). La prima rende disponibili le classiche funzioni, prima disponibili solo come dispositivi fisici, come funzioni virtualizzate , spesso distribuite tramite macchine virtuali, che quindi possono essere eseguite su hardware commerciale senza dover ricorrere ad hardware specifico.Anche la creazione da parte dei provider delle funzioni diventa più semplice e meno dispendiosa , trattandosi semplicemente di scrittura di codice della funzione da rendere disponibile. Banalmente anche il tempo per ottenere la funzione è quasi immediato, e inoltre una stessa funzione può essere riutilizzata tante volte quanto necessario, cosa precedentemente impossibile da fare con funzioni fisiche. La seconda tecnologia introdotta permette invece di poter definire connessioni tra le funzioni realizzate tramite NFV in modo semplice, veloce e su richiesta creando una Service chain (catena di servizi) un 'unione di funzioni diverse che combinate tra di loro in un certo ordine realizzano una funzione complessa . Questa tecnologia è basata su dispositivi di rete molto semplici, delegando la gestione di questi ad un controllore centrale, questo permette un elaborazione dei pacchetti di rete più veloce ed efficiente, anche se l'unico controllore risulta essere un single point of failure.

Tramite NFV si possono realizzare tutti i tipi di funzioni di rete come Firewall, NAT , Load Balancer ecc..., basandosi sulla loro versione virtualizzata non dovendo più ricorrere alla loro versione hardware, più costosa, più esigente in termini di risorse (spazio fisico, alimentazione,ecc..) e di complessa manutenzione. Al contrario la nuova tipologia di funzioni è vantaggiosa dal punto di vista economico e dello spazio occupato e permettono di avere una maggiore flessibilità e scalabilità delle reti.

NFV porta la separazione tra hardware e software permettendo un 'evoluzione parallela e indipendente di entrambi, partendo da questa divisione è possibile riassegnare e condividere le risorse all'interno della rete, potendo eseguire funzioni diverse tra loro sullo stesso hardware senza problemi. Nella concezione passata delle reti rispondere alle esigenze di traffico e di fornitura del servizio richiedeva tempo, denaro e pianificazione, invece lo scaling delle funzioni virtualizzate è molto più semplice e immediato. Un altro vantaggio è il ridotto consumo di energia dovuto al fatto che le funzioni e i servizi non devono essere necessariamente sempre attivi ma solo quando sono effettivamente richiesti. Infine l'impostazione e la configurazione delle reti è molto più veloce con l'NFV.

L'architettura NFV [6] viene definita dall' European Telecommunications Standards Institute (ETSI) che contribuisce anche alla pubblicazione di standard per la sua implementazione. L'architettura NFV è composta da:

- Virtual Network Function (VNF): applicazioni software che implementano le funzioni di rete

- Network Functions Virtualization Infrastructure (NFVI): è l'infrastruttura fisica che fornisce l'hardware (computazione,storage e networking) di base per l'esecuzione delle funzioni di rete

- Management and Orchestration (MANO): questo componente fornisce le funzionalità per gestire sia la NFVI che le VNF, semplificando l'orchestrazione dei servizi e la gestione dei dispositivi

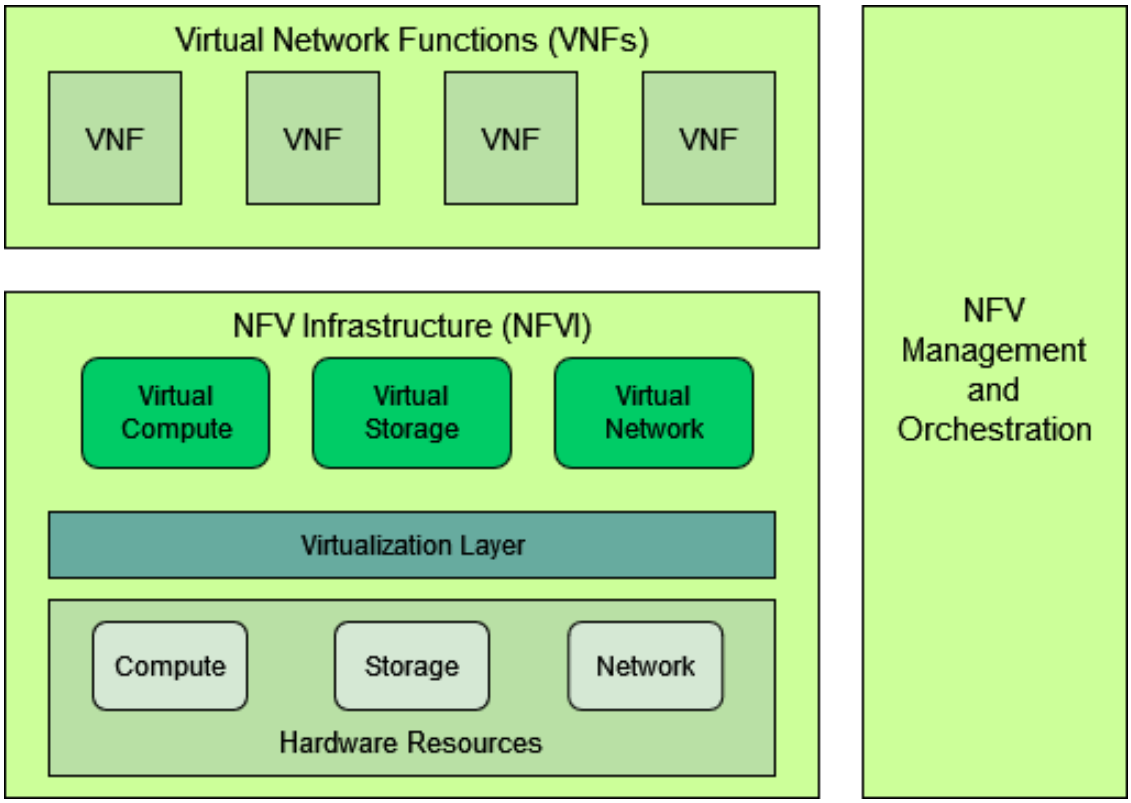


Figura 2.3. Architettura NFV

Una versione standard di questa architettura prevede la presenza di:

- OSS/BSS: comprende le componenti di sistema di supporto alle operazioni (OSS) sistema di supporto al business (BSS) che però sono componenti proprietarie del service provider , non incluse nella infrastruttura NFV. Il MANO prevede interfacce per l'interazione con questo componente.

- VNF: si intende le macchine virtuali e i servizi eseguiti al loro interno; queste possono essere formate da più componenti e possono essere eseguite o su VM o su container.

- NFVI: con questo termine facciamo riferimento ai componenti hardware e software che contribuiscono alla creazione del corretto ambiente per le funzioni.é composto da:

  - Hardware Layer: l'hardware effettivo, include i dispositivi che forniscono capacità di calcolo, storage e funzionalità di rete
  - Strato di virtualizzazione : a questo livello le risorse dello strato fisico sono esposte come risorse virtualizzate, quindi computing, storage e networking virtualizzato.Questo strato , il componente principale dell'infrastruttura, è caratterizzato dll'hypervisor, detto anche VMM (Virtual Machine Manager), che ha il compito di virtualizzare le risore separandole dalla loro versione fisica , allocare VMs e gestire la condivisione di risorse e l'isolamento.
  - Virtualized Layer: sopra lo strato di virtualizzazione è presente l'infrastruttura virtualizzata, astrazione delle risorse fisiche di computing storage e di networking.

- MANO: questo componente si occupa della gestione e dell'orchestrazione dei componenti NFVI e delle varie VNF. Può essere suddiviso in tre moduli distinti:
  - Virtualized Infrastructure Manager (VIM): responsabile della gestione delle risorse virtuali e della loro allocazione
  - VNF Manager (VNFM): si occupa del ciclo di vita delle VNFs, tramite le operazioni di istanziazione , configurazione e terminazione.
  - NFV orchestrator (NFVO): orchestra e gestisce il servizio nel suo complesso.

## 2.2 NFV MANO

In questo capitolo vengono introdotte le motivazioni e i fondamenti dell'orchestrazione per la tecnologia NFV. Il principio base di NFV è la separazione tra l'hardware e il software quindi, mentre prima le funzioni di rete erano strettamente collegate con le risorse utilizzate , adesso l'implementazione software delle NF è disaccoppiata dalle risorse di rete, storage e computazione che utilizzano. Questa proprietà, realizzata tramite lo strato di virtualizzazione e l'hypervisor, introduce nuove funzionalità e possibilità non presenti nel passato per cui va considerato un nuovo modo di orchestrare e gestire le funzioni , la loro interconnessione e i rapporti con le risorse della NFVI. Viene quindi introdotto il componente Network Functions Virtualization Management and Orchestration (NFV-MANO) [7] che ha il compito di gestire la NFVI e orchestrare l'allocazione delle risorse richieste dalle VNFs e dagli NSs, unione di più funzioni di rete.

Analizziamo adesso gli aspetti dell'orchestrazione secondo la seguente struttura gerarchica:

- Infrastruttura virtualizzata
- Funzioni di rete virtualizzate
- Servizio di rete

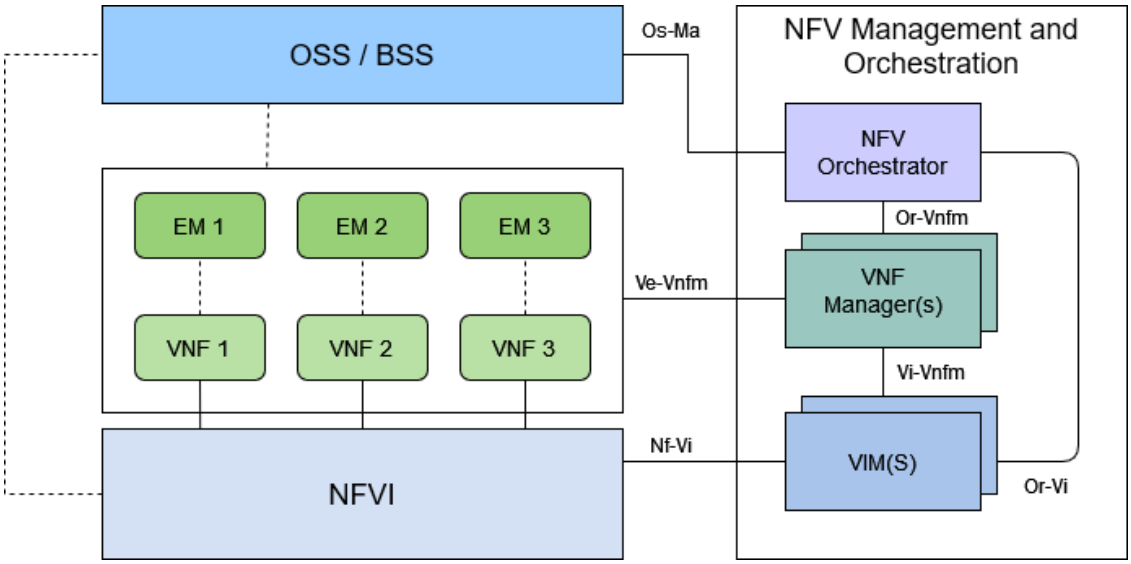


Figura 2.4. Architettura NFV con dettaglio su componente MANO

### 2.2.1 Aspetti MANO della NFVI

Le risorse da considerare quando si parla di NFVI sono sia risorse virtuali che risorse non virtuali, utilizzate per la creazione di funzioni di rete virtualizzate o parzialmente virtualizzate. Le risorse virtualizzate di nostro interesse sono quelle che sono state raccolte e rese disponibili all'uso attraverso delle astrazioni opportune, per esempio:

- Risorse di computazione incluse macchine, macchine virtuali, come risorse che comprendono memoria e CPU
- Archiviazione: volumi di storage o a blocchi o a livello di file system
- Risorse di rete come reti, sotto-reti, porte , indirizzi, link e regole di inoltro per realizzare le comunicazioni all'interno e tra le VNF

Le operazioni di gestione a questo livello consistono nell'orchestrazione delle risorse della NFVI, distribuite nei NFVI Points of Presence (NFVI-PoPs), fornendo connettività ai PNF, utile ad esempio quando un servizio è distribuito su diversi PoP.
Le risorse virtualizzate rese disponibili vengono poi usate per allocare le funzioni di rete. Quest'operazione però non è di semplice realizzazione perché bisogna rispettare vari requisiti e vincoli, come i requisiti di connettività per cui una funzione può richiedere connessioni a bassa latenza o alta banda.

### 2.2.2 Aspetti delle VNFs

Oltre alle funzioni di orchestrazione e gestione tradizionali (FCAPS) presenti già nelle reti di vecchia generazione, il disaccoppiamento delle funzioni di rete dall'infrastruttura fisica introduce un nuovo aspetto riferito come VNF Management.

Sotto questa definizione vengono raccolte tutte le funzionalità di creazione e gestione del ciclo di vita delle risore virtualizzate richieste dalle VNF. Le funzioni di VNF Management includono funzioni del tipo:

- Istanziazione della VNF
- Scaling della VNF
- Aggiornamento della VNF
- Terminazione della VNF

Le caratteristiche di allocazione di funzionalità della VNF vengono raccolte in un template che viene archiviato nel MANO all'interno di un catalogo per poter essere riutilizzato in futuro.Questo template descrive in maniera dettagliata e precisa i requisiti per poter riprodurre la VNF e per poterne gestire il ciclo di vita, su questi requisiti si baserà il lavoro di gestione del ciclo di vita eseguito dalle funzioni di VNF Management. Al momento dell'istanziazione vengono assegnate alla VNF le risorse che sono descritte nel template e considerando specifici requisiti, vincoli e policy che sono stati forniti in precedenza o al momento dell'istanziazione. Le funzioni di VNFM possono anche monitorare gli indici di prestazione di una vNF, se questi sono stati inseriti nel template, e usarli per effettuare operazioni di scaling : cambio di configurazione delle risorse virtuali (scale up, aggiungere cpu, scale down, rimuovere una cpu),aggiungere risorse virtuali (scale out, aggiungere una vm), spegnere e rimuovere una vm (scale in), o rilascio di alcune risorse (scale down).

### 2.2.3 Aspetti MANO dei Servizi di rete

L'orchestrazione del Network Service è responsabile delle seguenti operazioni:

- On-boarding del NS, ovvero registrare un Network Service nel catalogo
- Istanziazione del NS, creare un servizio basandosi sui template
- Scalare il servizio di rete , aumentare o ridurre la capacità del servizio di rete
- Aggiornare il servizio di rete supportando cambiamenti alla configurazione di varia complessità
- Creare,eliminare,interrogare e aggiornare il VNFFG associato al NS
- Terminazione del Network Service, di conseguenza terminare le VNF costituenti, richiedendo il rilascio delle risorse della NFVI associate, in modo che queste ritornino disponibili per altre funzioni

Come per le funzioni, anche per i servizi di rete esistono dei template che includono le loro caratteristiche per poterne effettuare correttamente l'allocazione. Sono incluse informazioni per gestire l'associazione tra VNF diverse , la topologia del servio di rete e il VNFFG associato al servizio.

### 2.2.4 NVF MANO Architecture

Come descritto in precedenza l'architettura MANO è composta dei seguenti blocchi funzionali:

- Virtualised Infrastructure Manager (VIM)
- NFV Orchestrator (NFVO)
- VNF Manager (VNFM)

Inoltre il MANO, per adempiere alle sue funzionalità, dispone di alcune repositories nelle quali vengono salvati i cataloghi di funzioni e servizi e informazioni riguardanti le risorse della NFVI e le VNF istanziate:

- NS catalogue
- VNF catalogue
- NFV Instance repository
- NFVI resource repository

#### NFV Orchestrator (NFVO)

L'NFVO può essere visto come l'unione di due componenti che rispondono ad esigenze diverse. Il primo componente è il Resource Orchestrator (RO) che si occupa appunto della gestione delle risorse, anche tra VIM diversi, mentre il secondo è il Network Service Orchestrator (NSO) il cui compito è gestire il ciclo di vita dei servizi di rete. In realtà questa divisione non è realizzata fisica ma dal punto di vista concettuale, in futuro però per una miglior separazione delle responsabilità potrebbe essere richiesta una divisione più netta.
Il Resource Orchestrator raccoglie le richieste di risorse proveniente dal VNF Manager, ne effettua l'autenticazione e la validazione prima di allocare effettivamente le risorse per l'utilizzo. La relazione tra le VNF allocate e le risorse da loro utilizzate viene salvata nella repositgory delle risorse della NFVI, utile a gestire meglio i rapporti tra fisico e virtuale. Può anche localizzare i/il VIM effettuarne l'accesso e fornire la posizione del VIM al VNFM , quando richiesto. Si occupa anche di controllare il rispetto delle policy di allocazione che sono state espresse o all'interno dei template o al momento dell'istanziazione del servizio e/o delle funzioni.
Il Network Service Orchestrator si occupa delle gestione dei template di deployment degli NS e dei package delle VNF, necessari prima di poter creare il servizio. Durante l'operazione di on-boarding è prevista una fase di validazione per controllare che all'interno del descrittore siano presenti tutti i file richiesti e che questi siano corretti, verificandone autenticità e integrità.Al momento dell'on-boarding le immagini che sono state specificate nei template vengono caricate, grazie all'azione del VIM, nei vari PoP, per poter poi istanziare le funzioni. Una volta svolto il passo preliminare di caricamento dei template, sia delle funzioni che del servizio , è possibile effettuare l'istanziazione delle funzioni, in collaborazione con il VNFM , e di

conseguenza anche del servizio di rete. L'NSO a questo punto può gestirne il ciclo di vita, modificandolo, scalandolo e gestendone la topologia, tramite il concetto di VNF forwarding graph.

### VNF Manager

Il VNFM ha il compito di gestire il ciclo di vita delle VNF, ogni VNF è associata ad un solo VNFM mentre lo stesso VNFM può gestire più VNF anche di tipo diverso. La maggior parte delle funzioni svolte dal VNFM sono funzioni generiche e non complesse ma può succedere le VNF richiedano l'esecuzione di funzioni più dettagliate e questa possibilità deve essere supportata dall'architettura che infatti permette la descrizione di queste funzioni nel package.
Quando il Network Service Orchestrator deve istanziare un servizio prima deve comunicare con i/il VNF Manager il cui compito principale è quello di istanziare, dopo aver controllato la fattibilità dell'operazione, le VNF ed eventualmente gestirne la configurazione se questa è specificata all'interno del package.Si occupa anche della modifica della VNF, dell'aggiornamento dei software, dello scaling (in/out e up/down), controllo sull'integrità della funzione durante il suo ciclo di vita, così come la raccolta delle informazioni legate all'utilizzo delle risorse della NFVI.Una volta che la VNF termina il proprio compito il VNFM ne ordina la terminazione permettendo il rilascio delle risorse occupate.

Anche nel caso delle funzioni , le caratteristiche del deployment e il loro comportamento è descritto in un template detto Virtualised Network Function Descriptor (VNFD) che viene salvato all'interno del catalogo delle VNF. Grazie alla presenza di questo descrittore è possibile creare istanze della VNF rappresentata e gestirne il ciclo di vita. Un VNFD viene incluso all'interno di un Package e contiene tutti i requisiti necessari per poter eseguire la VNF.Anche le risorse della NFVI che devono essere associate alla VNF sono espresse nel descrittore.Un punto fondamentale che deve essere rispettato per poter garantire la creazione di funzioni che siano flessibili e portabili potendo essere eseguite su diverse architetture NFV è l'utilizzo di risorse hardware correttamente astratte e fare riferimento a queste astrazioni.

### Virtualized Infrastructure Manager (VIM)

Il VIM è il componente responsabile della gestione e del controllo delle risorse di compute, storage e rete della NFVI, di solito appartenenti alla stessa infrastruttura posseduta da una singola entità. Un VIM può essere generico , quindi capace di gestire tipi diversi di risorse ,oppure può essere specilizzato nella gestione di una sola tipologia di risorse. Un esempio può essere un WIM (WAN Infrastructure Management) che permette la creazione di reti tramite tecnologia SDN e quindi gestione delle risorse di rete. Il VIM si interfaccia al livello sottostante con diversi hypervisor e controllori di rete per poter eseguire le funzionalità che espone.Come già detto queste funzionalità possono essere esposte direttamente da controllori specializzati, ad esempio il WIM che permette di creare connessioni tra funzioni diverse , localizzate su PoP differenti.

Il VIM tiene traccia in una repository di tutte le risorse fisiche e virtuali disponibili e informazioni legate al loro utilizzo per poi poterle fornire ai livelli superiori se richieste.Queste informazioni vengono usate per allocare/rilasciare le risorse della NFVI ,potendo gestire meglio l'associazione tra risorse virtuali e risorse fisiche. Grazie al VIM è anche possibile creare i VNF Forwarding Graph richiesti dai livelli superiori, in quanto esso svolge le funzioni di creazione e gestione dei link virtuali, delle reti virtuali, sottoreti così come la gestione di policy per controllo di accesso e traffico. Il VIM si occupa anche della gestione delle immagini software richieste dagli altri blocchi del MANO, attrsverso operazioni di aggiunta, rimozione, aggiornare, interrogazione e copia. Le immagini , dopo aver passato una fase di validazione, vengono inserite all'interno di una repository per semplificare il processo. Viene mantenuto un catalogo delle risorse virtualizzate appartenenti alla NFVI che possono essere utilizzate.

### NS Catalogue

Il catalogo NS è la repository usata per mantenere tutti i Network Service di cui è stato effettuato l'on-boarding, è importante per la creazione e la gestione dei deployment template degli NS (Network Service Descriptor(NSD, Virtual Link Descriptor(VLD), VNF Forwrding Graph Descriptor (VNFFGD))

### VNF Catalogue

Il catalogo VNF è la controparte delle VNF del NS catalogue, archiviando i package(contententi descrittore, immagini software, ecc..) dopo l'operazione di onboarding, permette di supportare la creazione e la gestione dei package attraverso le operazioni dell' NFVO. Questa repository può essere acceduta sia dall'NFVO che dal VNFM che gestisce la VNF per compiere le loro funzioni

### NFV Instances repository

All'intero di questa repository vengono salvate le informazioni sulle istanze delle VNF e sulle istanze degli NS.Un'istanza viene memorizzata attraverso un record del medesimo tipo, istanza VNF avrà un record VNF.Questi record possono essere modificati nel corso dell'esistenza dell'istanza, in base a operazioni che vengono svolte sull'istanza stessa riportando questi cambiamenti all'interno del record

### NFVI Resources repository

Questa repository mantiene informazioni rigurado alle risorse della NFVI rese astratte dal VIM, quali sono libere, quali occupa, se sono state riservate. E' fondamentale il suo contributo al ruolo del RO del NFVO dato che permette di tenere traccia di quali risorse sono associate con le istanze NS e VNF attive.

### Element management (EM)

Questo componente si occupa dell'esecuzione delle funzioni FCAPS per una VNF, tra cui:

- Configurazione delle funzioni di rete fornite dalla VNF
- Gestione dei guasti delle funzioni di rete di una VNF
- Accounting sull'uso delle funzioni della VNF
- RAccolta di risultati relativi a misurazioni di prestazione delle funzioni della VNF
- Gestione della sicurezza delle funzioni della VNF

Per poter svolgere le ue funzioni l'EM deve essere al corrente della virtualizzazione e quindi spesso comunica con il VNFM per conoscere le risorse dell'infrastruttura associate alla vnf da gestire

### Operations Support System/Business Support System (OSS/BSS)

E' la combinazione delle funzioni di supporto alle operazioni e al business dell'operatore che non possono essere racchiuse nei componenti del framework MANO, coi quali componenti però è atteso uno scambio di informazioni. Le funzioni di questo "componente" prevedono la gestione e l'orchestrazione dei sistemi obsoleti dell'operatore e quindi hanno completa visibilità sui servizi di reti offerti all'intero della rete dell'operatore.

### Network Functions Virtualisation Infrastructure

La NFVI comprende tutti i componenti hardware e software che insieme forniscono l'infrastruttura sulla quale le VNF possono essere eseguite

# Capitolo 3

# VEREFOO

Uno dei problemi nella gestione delle reti e della loro sicurezza è la ripetività di certe operazioni che sommandosi alle altre problematiche contribuisce a rendere il processo di messa in sicurezza di una rete più soggetto alla presenza di errori. Per risolvere questo problema è stato creato VEREFOO (Verified Refinement and Optimized Orchestration) [8], un framework che permette di definire in modo automatico e opportuno la sicurezza all'interno di una rete tramite soddisfacimento di requisiti e con la corretta allocazione delle funzioni di sicurezza di rete all'interno del grafo formato dall'insieme delle funzioni (sia di rete che non di rete).[9] [10][11]

Prima di esaminare in dettaglio il funzionamento di verefoo è importante introdurre alcuni concetti su cui esso basa il suo lavoro:

- Service Graph: si tratta di una topologia logica della rete composta da un insieme di funzioni di rete che vengono usate per fornire un servizio. Queste possono essere funzioni generiche
- Allocation Graph: è una topologia simile al service graph , caratterizzata dalle stesse funzioni, ma con l'aggiunta di nodi particolari detti Allocation Places. Questi nodi rappresentano i punti in cui una funzione potrebbe essere allocata
- Physical Graph: rappresenta l'infrastruttura fisica sulla quale le funzioni di rete già presenti nel service graph e le funzioni di sicurezza aggiunte nell'Allocation graph vengono istanziate, considerando ovviamente le risorse fisiche dei nodi e le risorse richieste da ogni funzioni

Usando come riferimento la figura 3.1 analizzeremo il flusso di esecuzione di Verefoo identificando il ruolo svolto dai diversi componenti. Un operatore che intende interagire con il framework deve fornire in input un insieme di Network Security Requirements (NSR) che rappresentano i vincoli di sicurezza che devono essere rispettati e un Service Graph o , se si dispone della conoscenza necessaria , un Allocation Graph [12]. Entrambi possono essere forniti usando delle GUI , rispettivamente la Policy GUI e la Service GUI; quest'ultima fornisce una serie di funzioni, disponibili da un catalogo , per poter costruire facilmente il grafo. Per prima cosa i vincoli forniti vengono analizzati dal modulo PAN per individuare possibili conflitti

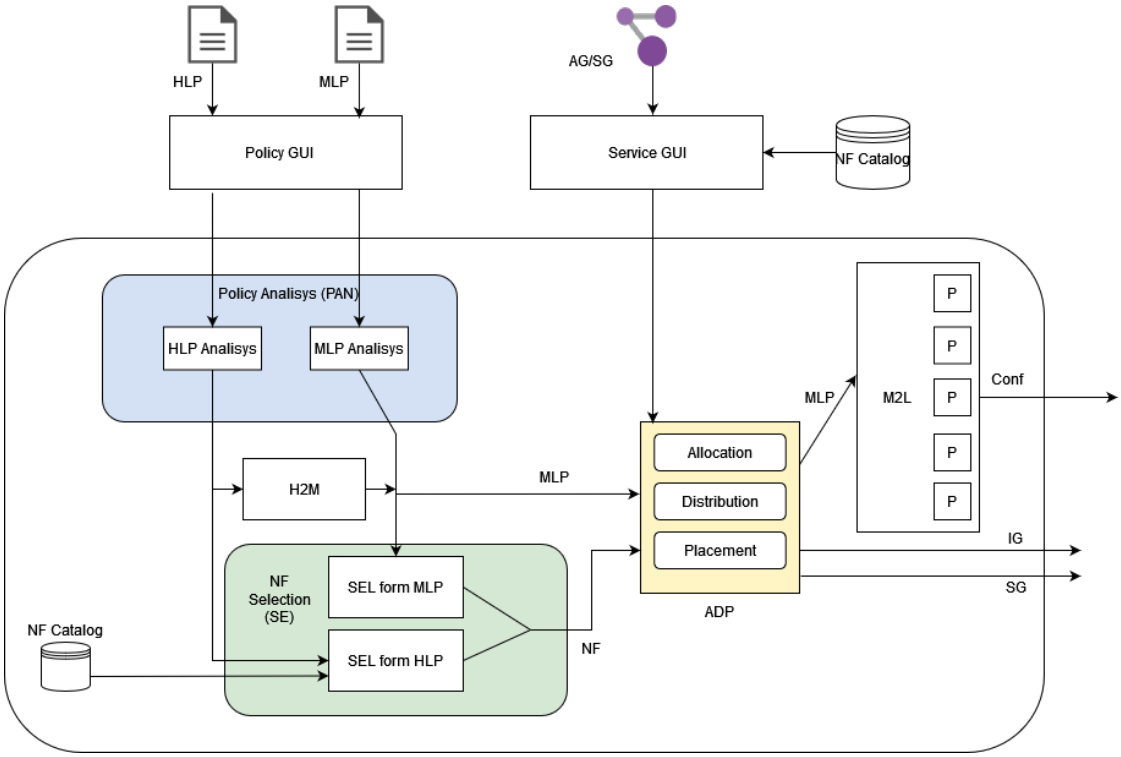


Figura 3.1. Architettura di Verefoo

e l'insieme minimo di requisiti da introdurre nella rete per rispettare tali vincoli. Subito dopo gli stessi vincoli, se sono stati espressi tramite un liguaggio ad alto livello (HLP) ,vengono tradotti in un insieme di requisiti di medio livello (MLP), contenenti tutte le informazioni utili per la creazione delle policy su cui si baserà l'allocazione delle funzioni di rete e la definizione delle loro configurazioni. Successivamente il modulo SE (NF Selection) basandosi sulle due tipologie di requisiti identifica le funzioni di sicurezza adatte per soddisfare i requisiti, usando come riferimento le funzioni presenti nel catalogo NF. Infine unendo il service graph (o allocation graph) iniziale , la lista delle funzioni di rete e l'insieme dei requisiti ,il framework fornisce come output una versione del service graph nel quale sono state posizionate anche le funzioni di sicurezza, fornendone anche una configurazione, espressa sotto forma di regole specifiche, che contribuisce al soddisfacimento dei requisiti.

Con questo funzionamento verefoo è quindi capace di definire quali funzioni di sicurezza di rete vadano inserite nella topologia e identificarne la miglior posizione all'interno di essa, definendo anche una configurazione specifica per la determinata funzione per poter rispondere alle esigenze di sicurezza espresse tramite i NSR. Verefoo è anche in grado di verificare una certa configurazione che gli viene fornita in input, ad esempio se un operatore vuole controllare se l'attuale configurazione di rete è in grado di soddisfare una nuova versione di requisiti senza dover operare modifiche prima del dovuto o anche solo definire le configurazioni delle funzioni in un grafo le cui funzioni sono già state inserite.[13] Inoltre è possibile anche effettuare l'istanziazione delle VNF che rappresentano le funzioni di rete nel service graph sul

physical graph, inserendo tra i vincoli per l'allocazione le risorse fisiche presenti nei nodi e richieste dalle funzioni.

# Capitolo 4

# Obiettivi della tesi

Per garantire il corretto funzionamento di un servizio di rete, realizzato e mantenuto tramite un orchestratore, è necessario prendere in considerazione eventuali minacce alla sicurezza esterne. Queste, interagendo con il servizio possono comprometterne la normale esecuzione impedendo agli utenti reali del servizio di sfruttare le sue funzionalità o anche di poter ottenere dati sensibili di proprietà degli utenti. Il servizio va difeso non solo da minacce esterne ma anche da possibili interazioni non previste tra le funzioni che lo compongono.

Quindi durante lo sviluppo del servizio di rete vanno identificate tutte i possibili attacchi per poter definire le dovute misure di sicurezza. Sebbene è possibile svolgere questo compito in modo manuale da parte di un operatore specializzato , la possibilità di errori dovuta all'azione umana fa sorgere un'ulteriore problematica.

Per definire la sicurezza all'interno di un servizio di rete può essere utilizzato VEREFOO che calcola il miglior posizionamento all'interno del grafo del servizio delle funzioni di rete e ne fornisce una configurazione. Quest'operazione viene svolta considerando i requisiti di sicurezza di rete che vengono forniti in input da un operatore. L'output del processo è , nel caso in cui tutte le richieste possono essere soddisfatte, un grafo completo anche di funzioni di sicurezza di rete scelte da verefoo stesso.

L'obiettivo del lavoro di tesi è l'analisi dei due orchestratori principali OpenSource MANO e Openstack Tacker. Si presenterà la struttura di entrambi ,identificando i componenti principali, e i file attraverso i quali permettono la rappresentazione delle funzioni di rete e dei servizi di rete, ponendo una certa attenzione anche alla possibilità sia di definire una configurazione per le funzioni che si andranno a creare sia di assegnare manualmente, o all'istanziazione del servizio o prima in maniera statica all'interno di un descrittore, un indirizzo IP alle funzioni.

Una volta completato il lavoro di analisi si prenderà in considerazione l'orchestatore più efficiente da utilizzare, o entrambi se presentano lo stesso grado di difficoltà, e si cercherà di attuarne una forma di interazione con verefoo, per poter definire un livello si sicurezza in un servizio di rete gestito dall'orchestratore.

Prima si identificheranno le funzioni o i comandi specifici dell'orchestratore per poter definire funzioni e servizi. Poi utilizzando i comandi si andranno a definire delle funzioni e un servizio di rete generico per presentare in dettaglio come

queste operazioni vengono svolte interagendo con l'orchestatore. Una volta compresi i passi fondamentali di creazione di un servizio tramite un esempio semplice, verrà definito un servizio di rete complesso , caratterizzato da varie funzioni di rete diverse tra loro, a partire da un grafo prodotto in output da verefoo, quindi con le funzioni di rete già posizionate nella posizione ottimale e caratterizzate da una configurazione che deve anch'essa essere replicata all'interno della corrispettiva funzione nel servizio di rete. Si identificheranno le funzioni presenti nel grafo e verranno realizzate tramite descrittori di funzioni per poi definire il descrittore del servizio di rete. Di questo servizio verranno presentati i file descrittori di tutte le funzioni uniche presenti nel grafo , il loro file di configurazione, sia esso generico o calcolato da verefoo in base ai requisiti, e il file descrittore del servizio di rete.

# Capitolo 5

# Orchestratori MANO

In questo capitolo vedremo in dettaglio i due orchestratori di funzioni virtualizzate attualmente più diffusi sul mercato: OpenSource MANO e Tacker di OpenStack.

## 5.1 OpenSource MANO

OpenSource MANO [14] è un progetto sviluppato dall'ETSI con l'obiettivo di creare un software di Management and Orchestration in ambito NFV con caratteristiche congruenti a quelle descritte dal modello informativo NFV ETSI. OSM è un framework caratterizzato da vari componenti che contribuiscono a fornire funzionalità MANO in un' architettura NFV.Inoltre un obiettivo importante di OSM è cercare di minimizzare il tempo richiesto per ottenere integration, basando il suo lavoro su 4 aspetti:

- Definisce un Information Model [15] utile per modellare le funzioni di rete (Network Functions) e i servizi di rete(Network Services) e usato per automatizzare il loro ciclo di vita, dal deployment alla fase operativa e di monitoring. Il modello non viene condizionato dall'effettiva infrastruttura, quindi lo stesso modello può essere usato su VIM diversi potendo ricreare la stessa funzione/ lo stesso servizio

- Fornisce una NorthBound Interface (NBI) che espone all'utilizzatore del sistema tutte le sue funzioni, che sono richieste per controllare il ciclo di vita del servizio

- OSM espande il concetto di Network Service andando a considerare servizi che possono essere composti non solo da funzioni virtuali ma anche da funzioni fisiche o finanche ibride, controllando il ciclo di vità del NS interagendo con VNF, PNF e HNF in modo trasparente

- OSM è capace anche di gestire il ciclo di vita di un Network Slice

Il framework è composto da tre componenti principali:

1. Resource Orchestrator (RO): gestisce e orchestra le risorse dell'infrastruttura NFV, eseguendo parte delle operazioni dell NFVO

2. VNF Configuration and Abstraction (VCA): questo componente gestisce il ciclo di vita delle VNF

3. Service Orchestrator (SO): si occupa del coordinamento dei precedenti componenti; gestisce anche il ciclo di vita del Network Service nel suo complesso, dalla fase di on-boarding alla fase di gestione on-line

Nella figura 5.1 viene rappresentata un' architettura logica di OSM , includendo i componenti precedentemente descritti che di seguito verranno approfonditi.

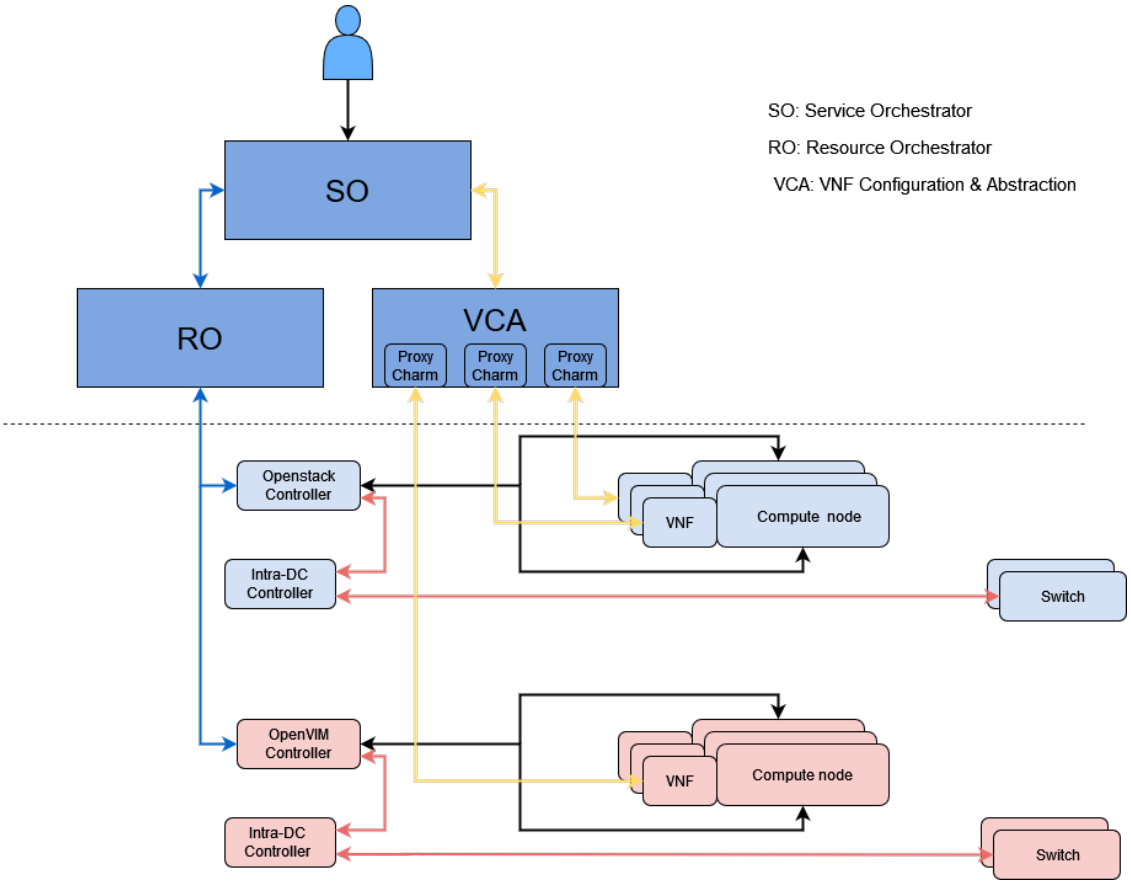


Figura 5.1. Architettura Logica di OSM

### 5.1.1 Resource Orchestrator

Il Resource Orchestrator utilizzato da OSM è OpenMANO che fornisce una gestione completa delle risorse attraverso l'uso di uno o più VIM. E' composto da diversi moduli scritti in python:

- openmanod.py : il programma principale

- httpserver.py : un processo che gestisce e coordina le richieste provenienti dalla northbound interface tramite richieste http
- nfvo.py : questo è il componente che implementa le funzionalità necessarie per gestire il ciclo di vita delle vnf attraverso le operazioni di creazione,eliminazioni, management di vnf e servizi
- nfvo_db.py: modulo usato per interagire con l' openmano DB, il quale contiene i cataloghi delle istanze e dei servizi di rete disponibili
- openmano_schemas.py: modulo usato per validare le richieste API e il contenuto delle risposte
- vim/openstack_connector.py: modulo che mappa le richieste in forma generica del componente NFVO in richieste per il VIM specifico.

Il VIM principale per cui è stato sviluppato OpenMano era OpenVIM, anch'esso sviluppato da telefonica.Però questo vim non ha avuto molto successo quindi si è optato per adottare altri VIM più diffusi come OpenStack.

### 5.1.2 VNF Configuration and Abstraction (VCA)

Questo componente è un'astrazione di un VNFM , infatti è possibile allocare più VNFM e EM al suo interno. Il VNF Manager di default di OSM è JUJU, il cui uso è strettamente raccomandato per concentrare le risorse su di una singola applicazione. Juju rappresenta entrambi i concetti di VNFM e EM, identificati rispettivamente dai suoi due moduli juju controller e proxy charms.

### 5.1.3 Service Orchestrator

Questo modulo fornisce funzionalità utili per eseguire l'on-boarding di NS e VNF e per poter gestire il ciclo di vita dei servizi di rete. Il SO usato da OSM è RIFT.ware che è capace di tradurre i descrittori partendo da vari formati ad un formato che può essere gestito dai componenti python di OSM, utlizzando un modello YANG. Inoltre gestisce l'interazione tra JUJU e l' RO ed espone una GUI utile a gestire il framework di OSM.

I suoi moduli sono basati su due strati, il CORE e il MANO. Il primo contiene funzioni generiche, del tipo:

- Logging: logging del sistema per l'SO
- DTS: supporto alla gestione di basi di dati
- YANG: gestione del modello dati YANG
- CAL: insieme di moduli interfaccia per connettere i vari VIM

Il secondo contiene plug-in per eseguire funzionalità SO generiche:

- Model Translation: traduzione dei vari formati in quello standard
- Catalog Manager: gestore dei cataloghi di VNFD e NSD
- VNF Monitoring: modulo per il monitoraggio
- NSO: plugin che abilità la possibilità di interagire con il Resource Orchestator di OSM, ovver OpenMano
- COnfiguration Manager: gestore delle configurazioni delle VNF, deve interagire con JUJU
- NFVI Metrics: metriche sull'infrastruttura NFV
- Account Manager: gestore degli accessi alla gestione di OSM

### 5.1.4 OSM Workflow

Come discusso prima, abbiamo affermato che OSM è basato su di un'infrastruttura fisica le cui risorse vengono esposte come risorse virtuali tramite un hypervisor e il modulo VIM si occupa di gestire l'infrastruttura. Ogniuna delle VNF istanziate sull'infrastruttura implementano una porzione specifica del servizio di rete che si intende realizzare. Le VNF devono essere gestite da un VNFM, che nel nostro caso è rappresentato da un componente interno di OSM, chiamato juju. Quando si vuole istanziare nuove VNF e VL (VIrtual Link), i link virtuali usati per connettere le funzioni tra di loro, usando OSM quest'ultimo deve comunicare con il VIM che generare le funzioni richiesete usando l'infrastruttura sottostante.Eventualmente OSM opererà sulle VM per eseguire delle operazioni di configurazione, o per cambiarne una versione già presente.

In OSM la configurazione viene effettuata in tre fasi diverse:

1. **Day-0 configuration**: questa è la fase nella quale la macchina viene preparata per poter essere gestita nelle fasi successive.
2. **Day-1 configuration**: durante questa fase viene applicata la configurazione desiderata alle VNF per ottenere le funzione richiesta
3. **Day-2 configuration**: durante questa fase lo stato della macchina viene modificato con quello nuovo che viene forntio

OSM è capace di creare reti su richiesta (Network as a Service), perchè opera come un Network Service Orchestrator (NSO) capace di creare servizi di rete e ritornare oggetti che possono essere usati nel processo di controllo del ciclo di vita e delle operazioni dell 'NS utilizzando le NBI API di OSM. I tipi di oggetto di rete che possono essere forniti sono due: il Network Service (NS) e il Network Slice Instance (NSI), composizione di servizi di rete che possono essere considerati come una singola entità.

OSM, in quanto funzione di gestione in una piattaforma di servizi, trae vantaggio da altre piattaforme e controlla un certo numero di funzioni gestibili per poter

fornire funzioni di livello superiore. Utilizza servizi offerti dalla piattaforma in carica di gestire l' Infrastruttura Virtualizzata (Per poter creare le VM) e la piattaforma in carica di creare reti SW-Defined (per creare le connessioni tra le VM), controlla e configura le funzioni di rete per controllare la gestione del ciclo di vita dell'interno NS che deve essere fornito.

Uno dei modi usati da OSM per gestire le reti fornite come servizio è il Network Service, che racchiude in un singolo oggetto un insieme (eventualmente ordinato) di funzioni di rete e può essere definito usando tecnologie diverse sparse in locazioni diverse.Il Network Service non può essere fornito come risultato di una procedura specifica , perdendo le caratteristiche on-demand, ma deve essere il risultato di un semplice metodo basato sull'invocazione di API e su descrittori secondo l 'Information Model. Questi descrittori dovrebbero facilitare la creazione di servizi di rete composti da funzioni di rete creati da vendor differenti, in modo che i vendor possono concentrare i loro sforzi sul modellamento della funzione stessa e invece i service provider sulla definizione del servizio.

Un NS può anche essere modellato fornendo un template che può essere ulteriormente specificato durante la creazione includendo attributi specifici per quella particolare istanza di NS.

OSM, seguendo il concetto di NS dell'ETSI NFV, incorpora i domini fisici e di trasporto per abilitare servizi E2E che possono andare oltre i domini virtuali. Di conseguenza in OSM è possibile creare un NS che combina VNF, PNF e anche funzioni composte da elementi fisici ed elementi virtuali.L'NS può anche essere deployato attraverso una rete distribuita creando delle connessioni inter-site, sfruttando le API delle piattaforme di rete SW-Defined.

Al livello superiore possono essere presenti delle piattaforme OSS e BSS che dovrebbero utilizzare i servizi creati da OSM, e in alcuni casi prendere controllo di una particolare funzione del servizio se necessario

### 5.1.5 Ciclo di vita e operazioni del NS

In questa sezione descriveremo più in dettaglio tutte le fasi del ciclo di vita e delle operazioni del NS. Le fasi sono:

0. Fase di preparazione: modellamento del servizio (e delle funzioni), in questa fase non vi è interazione con le API e il NS stesso ancora non esiste.
1. OnBoarding
2. Creazione del NS (day-0 e day-1)
3. Fase operativa del NS (day-2)
4. Terminazione del NS

La fase zero anche se non è una vera fase è utile per capire meglio l'intero processo di creazione.Analizziamo ora tutte le fasi in dettaglio

**Fase 0**

L' Information Model di OSM definisce un meccanismo per includere le informazioni riguardanti il comportamento di un NS, la sua topologia, le operazioni del ciclo di vita e le primitive del servizio; includendo tutte queste informazioni è capace di fornire una descrizione completa del NS. Il NS non è un elemento unitario ma è composto da varie funzioni di reti di diverso tipo e con funzionalità diverse quindi l 'IM fornisce un modo al fornitore del servizio per descrivere topologia,risorse, procedure e ciclo di vita delle funzioni di rete; queste informazioni vengono raccolte nei così detti NF Packages.

Questo approccio presenta diversi vantaggi: il progettista del servizio di rete può concentrarsi sulla composizione del servizio, basando il suo lavoro esclusivamente su proprietà esterne della VNF; lo stesso package NF può essere riutilizzato in più NS senza dover andare a modificare dettagli interni alla VNF.

**Fase 1**

Dopo che i modelli sono completi e pronti per l'utilizzo, possono essere inseriti nel sistema per essere usati come template nella successiva creazione del NS; questo processo viene chiamato on-boarding. La API di OSM permette di effettuare le operazioni CRUD sui package delle NF e dei NS

**Fase 2**

Dopo che i package NS e NF sono stati caricati in OSM, è tutto pronto per l'istanziazione del servizio da essi rappresentato; OSM offre delle chiamate API per supportare le operazioni CRUD relative alle istanze dei NS. In caso di creazione del servizio di rete (detta anche istanziazione), OSM riceve come input un Package NS e , opzionalmente, un insieme di vincoli e parametri per specializzare il NS. Durante la creazione del NS, OSM interagisce con altre piattaforme e con le funzioni sottostanti per creare l'istanza del NS.

**Fase 3**

Dopo che il NS è stato creato, l'unica risorsa oggetto di interesse rimane il NS Istance al quale bisogna fare riferimento per eseguire ulteriori operazioni sul servizio. Le operazioni che possono essere svolte sui NS fanno parte di una di queste tre categorie:

- Common Lifecycle operations: chiamate API che permettono di eseguire azioni standard sui NS, come scaling/pausa/ripresa, e monitoraggio
- Actions derived from NS primitives: oltre alle operazioni comuni condivise da tutti i NS, ognuno di loro è caratterizzato da un'insieme di funzionalità unico per lo specifico NS.Queste operazioni vengono presentate nel corrispondente NS package e sono esposte dalle API come azioni primarie.

- Esiste un gruppo di azioni che, anche se non sono richieste direttamente dal client tramite API, possono essere eseguite come risultato di un meccanismo di policy a circuiti chiuso definito nel package del servizio.Di solito queste operazioni coinvolgono l'osservazione di alcuni parametri, e l'esecuzione di un'operazione viene attivata se il valore del parametro considerato una determinata soglia.

**Fase 4**

Così come qualsiasi altro servizio on-demand, è possibile terminare il NS e effettuare il rilascio delle risorse assegnate, preservando solo i componenti che non devono essere rimossi.Delle API fa parte la chiamata delete che si occupa di far partire questo processo e riportare lo stato della sua esecuzione.

### 5.1.6 Descrittori

In OSM un rulo centrale viene svolto dai descrittori , documenti che permettono la descrizione completa di un oggetto di rete, descrivendono caratteristiche fisiche (topologia, risorse richieste, ecc..), configurazioni e funzionalità utilizzando come base un modello informativo standard. Esistono due descrittori principali:

- VNF Descriptor (VNFD): descrittore della funzione di rete virtuale.
- NS Descritor (NSD): descrittore del servizio di rete.

**VNF Descriptor**

Questo tipo di descrittore specifica tutte le caratteristiche che una funzione di rete specifica deve avere. In esso vengono specificati dati identificativi della funzione, prestazioni da rispettare, configurazione e dati di supporto. In OSM una funzione viene modellata usando le seguenti informazioni:

- Il descrittore effettivo
- La Virtual Deployment Unit (VDU): il più piccolo elemento che può essere istanziato.Ogni VDU possiede una propria immagine software e una funzione può essere composta da più VDU, ognuna con la propria immagine.
- Il link virtuale interno (IVL): si tratta del link interno tra le varie unità virtuali che insieme costituiscono la VNF completa.

**NSD**

In OSM un servizio di rete viene specificato usando le seguenti informazioni:

- Il descrittore generale del servizio di rete

- Il virtual Link (VL): si tratta di un'astrazione dei link che connettono tutte le VNFs, ovvero i link che connettono le interfacce esterne delle VNF; di solito viene implementato come una rete virtuale creata o esposta dal VIM
- Il forwarding graph: astrazione di un grafo, contenente le informazioni sui percorsi dei pacchetti.

## 5.2 OpenStack Tacker

Tacker [16] è un orcchestratore NFV e un VNF Manager che , usando un'infrastruttura NFV sottostante (può essere definita tramite OpenStack o Kubernetes), crea e gestisce funzioni di rete virtuali potendo aggregarle per formare un Network Service esponendo un servizio più complesso. Il progetto Tacker presenta due diverse architettura, l'implementazione Legacy e un 'implementazione basata sull' ETSI NFV-SOL, noi ci concentreremo sulla seconda dato che la prima sarà deprecata in futuro.

Seguendo i principi dell' ETSI NFV-SOL i blocchi funzionali del MANO sono NFVO, VNFM e il VIM, come mostrato nella figura 5.2. Tacker è capace di fornire le funzionalità di NFVO e VNFM, mentre il ruolo del VIM può essere ricoperto da OpenStack o Kubernetes. Il compito principale è quello di creare istanze di VNF, prima però va definito un VNF package , un file che include il descrittore della VNF , le immagini software per le macchine virtuali, e risorse aggiuntive come scripts e file di configurazione. Quindi all 'NFVO viene ordinato di effettuare il deployment di un'istanza VNF con il package definito precedentemente, richiedendo al VNFM l'allocazione effettiva della VNF sull'infrastruttua NFV.

### 5.2.1 Architecture

In questa sezione esamineremo come tacker è composto quali sono i suoi principali componenti. Tacker può essere diviso in due packages:

- python-tackerclient: questo package viene usato per l'interfaccia a riga di comando e per le chiamate all REST API, on generale quindi per poter interagire con il sistema.
- tacker : questo il package principale di Tacker e comprende il tacker-server e il tacker-conductor

Il primo occupa il ruolo di NFVO, gestendo VNF e NS , il secondo corrisponde al VNFM che riceve richieste di creazione delle VNF e si occupa della loro istanziazione e configurazione in relazione al VIM.

I due package contengono i seguenti componenti:

- tacker-client: fornisce la CLI e abilita la comunicazione con Tacker attraverso la REST API.

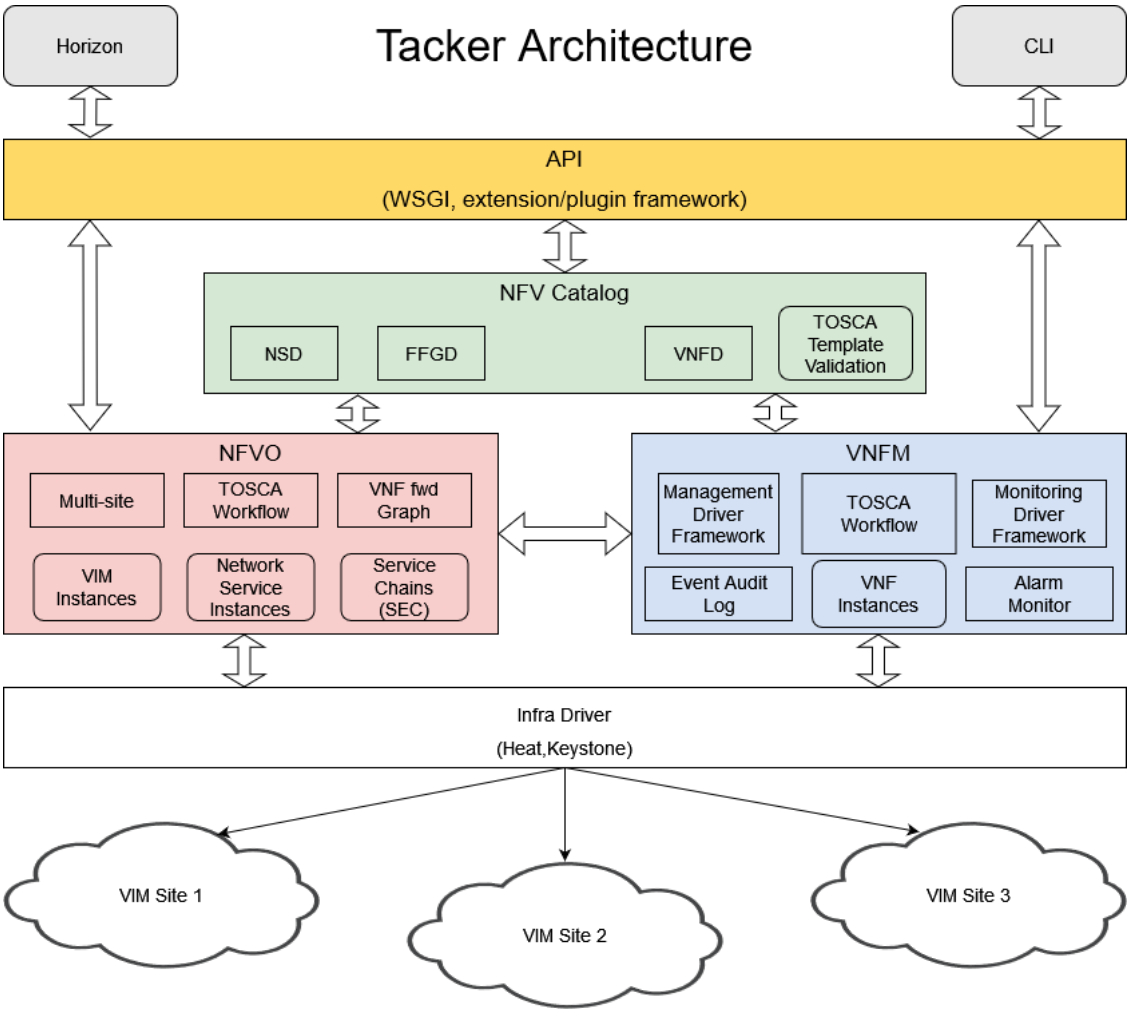


Figura 5.2. Vista logica di Tacker

- server : espone le REST API e chiama il conductor tramite RPCs.
- conductor : definisce tutta la logica di gestione delle VNF e si occupa di interpellare il driver corretto che espone l'infrastruttura NFV.
- infra-driver : questo modulo è responsabile di generare le operazioni esatte per operare le risorse di Openstack o Kubernetes.
- vim-driver : questo componente si occupa della registrazione dei VIM; tiene conto di quanti e quali VIM sono stati registrati e sono pronti all'uso.
- mgmt-driver : responsabile delle operazioni di configurazione delle vnf instanziate.
- monitor-driver : utilizzando questo componente si può monitorare lo stato delle vnf
- policy-driver : responsabile per le operazioni sulle VNF basate su delle policy

Il Tacker Service è caratterizzato da due processi principali:

- tacker.service
- tacker-conductor.service

tacker.service è un web server con una Web Server Gateway Interface (WSGI) che resta in attesa di chiamate REST per poi passarle ai driver sottostanti, una porzione delle operazioni viene inoltrata al prossimo processo. Il tacker-conductor.service definisce una logica avanzata per l'orchestrazione e la gestione delle VNF ed è inoltre responsabile di gestire le API basate su ETSI NFV-SOL e per la comunicazione con i VIM usando gli infra-driver

Quando un utente richiede certe operazioni usando il client , viene generata una chiamata REST API che viene inviata al tacker-server, poi il server esegue delle semplici operazioni interagendo con il DB usando delle query. Le altre operazioni vengono redirette al Conductor server tramite RPC, quindi il Lifecycle Driver chiama l'infra-driver corretto per eseguire la logica necessaria a controllare e gestire le risorse virtualizzate. Usando i driver di monitoring e di gestione , richiamati dal conductor, tacker fornisce dei meccanismi per configurare e monitorare le VNF

Per istanziare e creare una VNF è prima necessario definire dei packages.I package principali sono il VNF package e il VNF descriptor, col primo che include il secondo.

### 5.2.2 VNF Package

Il package VNF è un file compresso che include il VNFD, le immagine software per la VM, e altre risorse come script e file di configurazione; il package VNF deve essere un file compresso secondo le specifiche TOSCA-Simple-Profile-YAML, che viene chiamato TOSCA YAML Cloud Service Archive. Il file CSAR può presentare una delle seguenti strutture:

- CSAR con TOSCA-Metadata directory
- CSAR senza TOSCA-Metadata directory

Il VNF descriptor è un documento modello secondo il template TOSCA service, è un componente del VNF package ed è in formato YAML.

Questo descrittore comprende quattro tipi di file diversi, appartenenti a due categorie: file di definizione dei tipi, i file di questa categoria definiscono le proprietà delle risorse; file template di topologia, questi file descrivono la struttura delle vnf. I quattro tipi sono:

- File di definizione dei tipi ETSI NFV
- File di definizione dei tipi definiti dall'utente
- File template di topologia ad alto livello

- File template di topologia con deployment flavor

I file template di topologia ad alto livello rappresentano un designa astratto della VNF; invece il design effettivo viene descritto dall'ultima tipologia di file. Le risorse descritte nei file template di topologia devono essere state definite come "Tipi" nei file di definizione delle risorse; gli utenti possono anche estendere i tipi forniti per creare delle proprietà aggiuntive

## 5.3 Openstack

Il VIM principale usato da Tacker è OpenStack, una piattaforma libera e open source per il cloud computing; appartiene alla categoria di Infrastructure as a Service in quanto fornisce server come risorse virtuali. Usando le risorse fisiche di un'infrastruttura è capace di creare un insieme di risorse virtuali (compute,storage e network) attraverso il cui uso poter creare e gestire VM e il loro ciclo di vita.Possono essere creati dei link tra le VM per permetterne la comunicazione , anche utilizzando un controller SDN.

Openstack è composto da diversi moduli di cui i principali sono:

- Nova : Questo è il componente centrale di OpenStack e ha il compito di gestire ed automatizzare le risorse virtuali per permettere la creazione di macchine virtuali; supporta diverse tecnologie di virtualizzazione come KVM, VMWare e Hyper-V e anche tecnologie di container come LXD e Docker
- Neutron : attraverso Neutron possiamo gestire la rete, modellando delle topologie, impostando delle policy e creando connessioni tra le VM e da/verso l'esterno. Si tratta del punto dove confluiscono tutte le richieste legate a configurazione di tipo SDN in un cloud OpenStack, esso agisce come strato di astrazione con i propri plugin delle varie soluzioni SDN disponibili sul mercato, come Open vSwitch.

A causa della complessità della documentazione e delle operazioni richieste per poter realizzare funzioni e servizi in Tacker non è stata presa in considerazione l'interazione con verefoo, preferendo invece OSM con la sua semplicità. Per questo nella sezione successiva analizzeremo l'interazione tra verefoo e OSM.

## 5.4 Interazione tra OSM e Verefoo

L'obiettivo della tesi è quello di analizzare una possibile interazione tra Verefoo e uno dei più diffusi orchestratori di funzioni di rete virtuali disponibili sul mercato , OpenSource MANO. Lo studio verterà sul capire quanto verefoo possa sfruttarne la struttura, e , qualora l'interazione risulti possibile, identificare l'indice di fattibilità nell'ottenerla, considerando le caratteristiche di entrambi.

Per capire come le due entità possano entrare in collaborazione possiamo partire dal consirerare i wokflow di entrambi. Il lavoro di verefoo consiste nel ricevere

in ingresso un grafo , i cui nodi sono dispositivi di rete (si distinguono due casi: Allocation Graph nel quale le funzioni di rete non sono ancora allocate e quindi sta a verefoo prendere la decisione riguardo al loro posizionamento considerando dei punti di allocazione; Service Graph le funzioni di rete sono già presenti) e dei requsiti di sicurezza che devono essere rispettati dalle funzioni di rete presenti nel grafo effettuando le dovute configurazioni di quest'ultime. OpenSource MANO ha il compito di instanziare dei servizi di rete, insiemi di funzioni di rete che nel complesso forniscono funzioni più complesse rispetto alle singole funzioni. Si può riscontrare una certa corrispondenza tra i grafi con cui lavora verefoo composti da funzioni e i servizi di rete gestiti da OSM. Entrando nel dettaglio di OSM, prima di instanziare il servizio è necessario definire al suo interno quello che viene chiamato NSD, descrittore del servizio di rete, un file che contiene tutte le informazioni e le caratteristiche del servizio di rete che si desidera instanziare nelle fasi successive. All'interno del file si fa riferimento alle funzioni che compongono il servizio, ognuna di queste deve avere il corrispondente descrittore detto VNFD (stesso compito dell'NSD ma per quanto riguarda le funzioni di rete), decsrittore della funzione di rete virtuale, presente all'interno di OSM.

L'interazione che si può pensare di realizzare è quella di rappresentare, ed eventualmente instanziare, il grafo prodotto da verefoo usando OSM, producendo un servizio di rete con tutte le funzioni presenti nel grafo e correttamente configurate. Che il grafo di partenza venga preso da verefoo o sia già presente in OSM non è importante nel caso in cui si decida di preparare il servizio prima dell'instanziazione, mentre nel caso in cui si voglia prima instanziare il grafo e poi modificarlo allora il grafo andrà preso da OSM. Comunque la scelta dipende dalla scelta dell'ordine delle operazioni. Essendo la preparazione dei descrittore non ininfluente si consiglia di partire da verefoo e poi realizzare i descrittori.

Considerando che l'istanziazione effettiva del servizio risulta dell'esecuzione di un semplice comando, il compito principale viene identificato nella rappresentazione del servizio e delle funzioni di cui questo è composto. La realizzazione dei descrittori può essere facilitata dalla consultazione della repository ufficiale di OSM dedicata alle vnf e agli ns; difficilmente può essere realizzata in maniera automatica, in quanto all'interno dei descrittori devono essere presenti informazioni che all'interno del grafo non sono presenti, e le info che sono presenti vanno comunque gestite in maniera opportuna. Chiaramente la prima forma di interazione può essere svolta manualmente oppure prendere in considerazione la possibilità di introdurre uno script che automatizzi il solo processo di creazione dei descrittori; nel caso si provi questa seconda soluzione si può pensare di ridurne la complessità andando a creare automaticamente il solo package del NS facendo riferimento a VNF già presenti in OSM.

Un'altra caratteristica oggetto di studio è la complessità dei grafi che si possono creare e capire vantaggi e svantaggi della loro creazione tramite interfaccia o cli, in quest'ultimo caso però il package deve essere già stato composto e l'unica operazione che viene effettuata è l'on-boarding che , come le altre operazioni, può essere svolto sia tramite GUI che tramite CLI. Se si vuole usare la CLI la creazione del descrittore

va fatta manualmente generando ogni singolo elemento, mentre usando l'interfaccia grafica all'inserimento di una nuova funzione questa presenta una struttura base senza che l'utente debba crearla da zero. Nel seguito verranno date informazioni aggiuntive riguardo all'utilizzo della cli per le operazioni di creazione e on-boarding.

Partendo dall'elemento base , la VNF, si può notare che in realtà possono essere composte da una o più unità elementari dette VDU; l'identificazione di più VDU all'interno della VNF avviene nel caso in cui le operazioni della VNF interessino diversi campi tra loro separati in modo che le operazioni possano essere divise univocamente tra le componenti. Va detto che la suddivisione in componenti non è necessaria e viene usata per semplificare la progettazione delle funzioni. Le componenti all'interno della VNF vengono connesse tramite link virtuali.

Per lo studio risulta fondamentale capire se è possibile definire una configurazione delle vnf quando queste sono ancora in stato di package o se si è impossibilitati finchè non viene instanziato il servizio con le sue funzioni. Nel caso in cui la configurazione possa essere definita in modo completo dopo l'istanziazione allora si potrebbe creare solo il pacchetto, che però risulterà generico. Dalla documentazione di OSM si evince che la configurazione è posibile effettuarla in diversi modi. Il primo avviene usando un file di cloud-config, dove possono essere espresse configurazioni base ; il secondo prevede la definizione di charm per una configurazione più complessa. Mentre la configurazione viene realizzata effettivamente solo dopo che il servizio è stato attivato , i file degli charm possono comunque essere definiti.

Le due configurazioni che noi prendiamo in considerazione vengono dette day-0 e day-1.

Nella configurazione day-0 quello che va specificato è la composizione della vnf in vdu, le risorse che l'infrastruttura deve impegnare per allocare la funzione, eventualmente attributi relativi alle prestazioni. In questa fase una forma di configurazione può essere iniettata nelle vnf tramite il già citato file di cloud-init, va però capito quanto si può andare in dettaglio. Inserendo il file di cloud-init nel package è possibile stabilire preventivamente dei comandi per la vnf quando questa non è stata ancora allocata.

L'obiettivo della configurazione day-1 prevede l'inizializzazione automatica dei servizi della vnf dopo la sua istanziazione. La configurazione avviene tramite proxy charm, delle vnf appositamente realizzate solo per iniettare comandi di configurazione all'interno delle "vere" vnf. Come il file di cloud-init anche i file relativi allo charm possono essere definiti e inseriti nel package.

Nella configurazione day-2, realizzata sempre tramite charm, vengono effettuate operazioni di ridefinizione del servizio.

Si analizzano ora il contenuto dei descrittori sia delle funzioni che del servizio.

All'interno del package della vnf vengono descritti in ordine :

- I dati identificativi relativi alla funzione come nome ,id ,ecc..
- I punti di connessione con l'esterno (connection-point) , di cui uno di questi deve essere definito come interfaccia di management
- Nel caso di più VDU vanno definiti i link virtuali che le collegano
- Vengono definiti i dati delle singole VDU :
  - Dati identificativi come id e nome
  - L'immagine di base
  - Il numero di copie che devono essere istanziate della sottofunzione
  - Le specifiche fisiche come numero di cpu, dimensione di ram e di storage
  - Definizione delle interfacce, va inoltre specificato se collegano l'interno o l'ersterno e va specificata l'interfaccia sorrispondente del collegamento
  - Specifica dell'internal connection point usato al punto precedente per la connessione tra VDU
  - Come ultimo dato delle VDU va inserito il nome del file di cloud-init contenente le configurazioni da effettuare alla specifica VDU

Invece per quanto riguarda il descrittore del servizio va specificato:

- Le VNF che compongono il servizio, eventualmente può essere riusata la setssa funzione più volte
- I dati identificativi del servizio come : nome , id , descrizione, ecc..
- I link virtuali che collegano le funzioni tra di loro

Incluso nella CLI di OSM è presente un tool che permette di creare un descrittore base da zero in modo automatico specificando alcuni parametri, permettendo poi la successiva creazione di un package di cui poter effettuare l'on-boarding.

Nelle nuove versioni di OSM è inoltre presente la possibilità di definire delle catene di servizi scegliendo un percorso, in base a dei requisiti che corrispondono al traffico in esame, tra le varie VNF, specificando ingress ed egress port, rispettivamente ingresso nella catena e terminazione di essa.

Analizzando nel dettaglio la configurazione day-0 essa è più che altro usata con lo scopo di rendere accessibile la VNF da parte di OSM. Le possibili operazioni che si possono realizzare sono:

- Costruzione e aggiunta degli script di cloud-init
- Impostare un hostname dell'istanza

- Generare delle chiavi private ssh o definire password
- Aggiungere chiavi ssh alla sezione .ssh/authorized-keys di un utente
- Impostare mount point effimeri
- Aggiungere utenti e/o gruppi
- Aggiungere file

In questa fase bisogna preparare il descrittore in modo che questo descriva accuratamente i requisiti della VNF, si preparano gli script di cloud-init (se necessario, ma necessario nel nostro caso se possibile utilizzarli per configurare le funzioni di rete), identificare i parametri che dovranno essere forniti al momento dell'istanziazione

Si crea quindi il package da zero usando il client di OSM , obbligatorio per eseguire questo passaggio. Per creare il package della vnf si usa il comando : *osm package-create –base-directory /home/ubuntu –image myVNF.qcow2 –vcpu 1 –memory 4096 –storage 50 –interfaces 2 –vendor OSM vnf vLB*. Quello che viene specificato è la cartella all'interno della quale va creato il package, il nome dell'immagine di partenza della funzione , il numero di cpu, la dimensione della ram e dello storage , il numero di interfacce(oltre quella di management già inclusa), il vendor e infine viene indicato che il package creato riguarda una funzione e ne viene detto il nome. Per creare il package di un servizio di rete : *osm package-create –base-directory /home/ubuntu –vendor OSM ns vLB*. Una volta creati i package questi devono essere caricati in OSM tramite il comando vnfpkg-create per le funzioni e nspkg-create per i servizi.

Il modo per permettere l'inizializzazione dei servizi all'interno di una vnf dopo la sua istanziazione è tramite la creazione di uno charm e il suo inserimento nel package. Nel package finale saranno quindi presenti una sezione di metadati, dati dichiarativi specificati nel file .yaml e una parte di codice che si occupa delle operazioni relative alle vnf. Questo codice viene detto charm.

Per la configurazione day-1 bisogna generare 4 file : layer.yaml, matadata.yaml (contiene le informazioni sui dati), config.yaml (contiene i parametri di configurazione) e actions.yaml (contiene le azioni le cui implementazioni saranno presenti nel file reattivo charm.py) ovvero una lista delle azioni con descrizione e lista di parametri opzionali. Nel file actions è presente una parte fissa necessaria per il file charm.osm.sshproxy, per l'esecuzione generica di comandi. Successivamente viene definito il file charm.py in python contenente l'implementazione di tutte le azioni necessarie per far partire il servizio. La classe SampleProxyCharm va usata come base per la definizione delle classi di charm; nell'inizializzazione dello charm vanno osservati degli eventi: start, install e config-changed , in più si osservano gli eventi legati alle azioni. All'interno dei metodi vanno chiamati i metodi rispettivi della classe SPC che contengono azioni importanti per i proxy charm. Per la configurazione day-1 è obbligatoria la presenza nella vdu di un'interfaccia di management.

All'interno dello charm è anche possibile deinifire un comando in maniera statica come stringa e poi inviarlo alla macchina per l'esecuzione tramite il comando charms.sshproxy-run("comando")

Ricapitolando i passi nell'interazione con OSM sono :

1. Ottenere il grafo con funzioni allocate e configurazioni di sicurezza effettuata

2. Creare, tramite il tool della cli di osm osm package-create , un package base per quanto riguarda le varie VNF che possono essere identificate all'interno del grafo di verefoo (es. vm client,vm packet filter, ecc.) e modificarlo in base alle proprie necessità; si pensa di creare vnf con una sola vdu e di conseguenza una sola interfaccia verso l'esterno

3. Una volta creati i package e modificati in modo opportuno va capita come può essere realizzata la configurazione delle VNF, se con cloud-init o con i juju charm. Nel primo caso va inserito nel decsrittore .vnfd.yaml la linea col nome del file di cloud-init e in una cartella va poi inserito il file effettivo. Nel secondo caso va scaricata la suite che permette la definizione dei file degli charm , modificati in modo opportuno e poi vanno copiati all'interno del package.

4. Dopo aver definito anche la parte di configurazione delle VNF si devono caricare i package realizzati per le varie vnf presenti nel grafo all'interno di osm. é possibile svolgere quest'operazione in diversi modi: o attraverso la GUI, ma bisogna aver preparato una versione compressa del package in formato tar.gz e trascinarlo nella sezione di creazione delle vnf; utilizzando la cli e il comando osm vnfpkg-create "nomePacchetto" che effettuerà il procedimento precedente in maniera automatica, per utilizzare la cli di osm bisogna però averla installata e configurato i parametri di accesso correttamente, a meno che non si utilizzi la stessa macchina dove è presente l'istanza di osm, nel qual caso la cli è già presente e correttamente configurata; tramite protocollo HTTP usando la NBI di OSM, ma in questo caso va ottenuto il token per poter effettuare le operazioni.

5. Creati i package delle funzioni, va creato il package del servizio , specificando le funzioni presenti al suo interno e le loro connessioni, e se necessario un grafo all'interno del ns per creare una catena di servizi.

Infine per quanto riguarda la realizzazione delle proprietà di sicurezza va capito se queste possono implementate tramite comandi che possono essere iniettati all'interno della macchina. Se la risposta è positiva allora l'interazione tra OSM e Verefoo risulta essere non solo possibile ma anche efficace.

Riguardo all'assegnazione degli indirizzi IP , questi possono essere assegnati come parametri nel momento dell' instanziazione, va considerato però che se il numero di funzioni è elevato allora va preparato un file di parametri di configurazione per semplificare l'interazione ed evitare di scrivere direttamente su linea di comando l'intero gruppo di parametri. Oppure come esemplificato in questo paragrafo della

sezione uso di OSM della sua guida ufficiale [17] nella sezione virtual link connectivity può essere specificato l'indirizzo IP della macchina, o meglio l'indirizzo IP per la specifica interfaccia che si considera. Questa opzione di IP statico dovrebbe essere supportata da tutti i VIM disponibili per OSM.

# Capitolo 6

# Interazione tra Verefoo e OSM

Per creare il grafo di Verefoo all'interno di OSM, bisogna inanzitutto avere a disposizione un grafo già definito, nel nostro caso nella sezione gitlab di verefoo abbiamo accesso ad alcune topologie. Dopo aver ottenuto il grafo vanno capite le diverse funzioni presenti all'interno di esso per poi replicarle come funzioni di rete tramite package, quindi effettuare il loro on-boarding. In particolare tramite il tool di osm client e il comando osm package-create , si vanno a creare i package delle diverse funzioni che devono essere presenti all'interno del servizio di rete. Per semplificare la realizzazione delle funzioni tra le due tipologie di configurazioni si può scegliere la day-0 che comununque permette di eseguire dei comandi sulla macchina obiettivo, cosa che per noi risulta sufficiente per l'implementazione delle funzioni di firewall e forwarder usando iptables ed eventualemente altre configurazioni per le restanti macchine. Le configurazioni da scrivere all'interno del file cloud-init sono state ricavate dalla repository gitlab di verefoo, i file così creati vanno legati alle vnf (in particolare alla vdu che caratterizza la vnf) e inseriti in una cartella apposita all 'interno del package poi il package viene validato e caricato su OSM tramite osm vnfpkg-create "argomenti", successivamente va creato il package del servizio e va caricato allo stesso modo del pacchetto delle funzioni. Le succesive fasi di installazione di OSM e la creazione dei package sono descritte nelle sezioni di guida generale [18] e linee guida di on-boarding [19] di OSM.

## 6.1 Installazione di OSM

Per poter lavorare con OSM è necessario installarlo su di una macchina con dei requisiti che sono :

- MINIMI: 2 CPU, 8 GB RAM, 50 GB di disco e un'interfaccia con accesso ad internet
- RACCOMANDATI: 4 CPU, 16 GB RAM, 80 GB disco e un'interfaccia con accesso ad internet
- Immagine di base: Ubuntu 22.04 (sia la versione cloud che la versione server,in questo lavoro si è usata la versione server)

Avendo a disposizione una macchina con le caratteristiche sopra elencate si può procedere all'installazione di OSM tramite i comandi:

Listing 6.1. Comandi per installare OSM

```
wget https://osm-download.etsi.org/ftp/osm-15.0-fifteen/ /...
...install_osm.sh
chmod +x install_osm.sh
./install_osm.sh
```

Durante l'installazione verrà richiesto di installare e configurare LXD, juju, docker CE e un cluster k8s locale e busognerà rispondere di si (y). E' possibile installare componenti aggiuntive, si rimanda alla guida ufficiale per i dettagli (https://osm.etsi.org/docs/user-guide/latest/03-installing-osm.html).

Dopo un po' di tempo, circa una decina di minuti, l'installazione di OSM dovrebbe essere completata e pronta all'uso. Per verificare si può accedere alla UI all'indirizzo http://1.2.3.4, rimpiazzando 1.2.3.4 con l'indirizzo della macchina sul quale è installato OSM. Per accedere si utilizzando delle credenziali : user: admin, password: admin.

Tramite la UI si può accedere facilmente a tutte le funzionalità di OSM tra le quali visualizzazione e creazione dei package dei servizi e delle funzioni e instanza di funzioni e servizi.

## 6.2 Installazione e preparazione del tool osm client

OSM client è un tool di linea di comando per poter lavorare con OSM , effettuando l'accesso alla NBI di OSM e permettendo all'utente di gestire descrittori, VIM, servizi di rete e il loro ciclo di vita. Anche se il client di OSM è già presente nella macchina di OSM è utile installarlo su un'altra macchina in modo da poter operare da remoto anche dalla propria macchina.

Esistono due modi per installare il client di OSM: tramite Snap o pacchetto Debian

Nei sistemi che supportano snap, il client di OSM di può installare con il seguente comando

Listing 6.2. Comando per installare osm-client

```
sudo snap install osmclient --channel 14.0/stable
```

Si può installare la versione che si desidera tramite l'opzione channel, se si desidera l'ultima release disponibile l'opzione può essere omessa

Per installare il client di OSM nella propria macchina Linux, bisogna seguire la seguente procedura:

Listing 6.3. comandi alternativi per installare osm-client

```
sudo sed -i "/osm-download.etsi.org/d" /etc/apt/sources.list
```

```
wget -qO -
   https://osm-download.etsi.org/repository/osm/debian/ /
ReleaseFIFTEEN/OSM%20ETSI%20Release%20Key.gpg | sudo apt-key
   add -

sudo add-apt-repository -y "deb [arch=amd64] https://osm- /
download.etsi.org/repository/osm/debian/ReleaseFIFTEEN stable
   devops IM /
osmclient"

sudo apt-get update

sudo apt-get install -y python3-pip

sudo apt-get install -y python3-osm-im python3-osmclient

python3 -m pip install -r
   /usr/lib/python3/dist-packages/osm_im/ /
requirements.txt

python3 -m pip install -r
   /usr/lib/python3/dist-packages/osmclient/ /
requirements.txt
```

Per poter utilizzare il client OSM in una macchina diversa da quella di OSM è necessario specificare l'host di OSM, o tramite una variabile d'ambiente oppure tramite la command line di OSM. Ad esempio quello che si può fare è impostare il client per farlo accedere all'host che sta eseguendo all'indirizzo "1.2.3.4" usando il comando :

Listing 6.4. Comando per associare la macchina con OSM alla macchina con il client

```
export OSM_HOSTNAME="1.2.3.4"
```

Si può visualizzare una lista dei possbili comandi digitando osm nella command line.

## 6.3 Creazione dei package delle VNF

Dopo aver installato OSM e configurato il client si può passare alla definizione dei descrittori e dei package. Il comando da eseguire per creare un package è :

Listing 6.5. Comando generale per creare vnf package

```
osm package-create --base-directory "cartella di
   destinazione" --image "immagine da assegnare alla vnf" -- /
vcpu 1 --memory 4096 --storage 50 --interfaces 1 --vendor OSM
   vnf verefoovnf
```

Attraverso questo comando possiamo specificare le informazioni principali legate alla vnf che si va a definire, l'immagine con cui si definisce la VM,il numero di core assegnati, quantità di memoria principale e secondaria, numero di interfacce , tipologia di package e nome del package. Per semplificare la realizzazione del package del servizio si possono andare a modificare i punti di connessione esterni della vnf per avere una visione più chiara dei collegamenti (sezione ext-cpd, e se l'interfaccia è solo una assegnare lo stesso nome all'attributo mgmt-cp, interfaccia di management che nel caso di una sola interfaccia corrisponde all'unica presente). Una volta che il package è stato generato andiamo a controllare che i dati al suo interno siano corretti validando il package tramite il comando :

Listing 6.6. Comando per validare il package

```
osm package-validate "path del package"
```

Poi si devono definire i file di cloud-init contenenti i comandi per definire le funzionalità delle vnf , si crea un file di testo con all'interno la parola chiave runcmd seguita poi da tutti i comandi che devono essere eseguiti. Una volta definiti i file va inserito un riferimento a questi all'interno della sezione vdu nel packge della vnf corretta. Se si lavora con il client un file cloud-init vuoto è già presente nella cartella scripts del package e quindi va solo modificato opportunamente in base alla vnf che si sta realizzando. Un esempio del file di configurazione di un firewall è riportato di seguito:

Listing 6.7. Esempio file di configurazione per Firewall

```
#cloud-config
runcmd:
 - sudo iptables -F
 - sudo iptables -P INPUT DROP
 - sudo iptables -P FORWARD DROP
 - sudo iptables -P OUTPUT DROP
 - sudo iptables -A FORWARD -p tcp -s 220.124.30.1/32 -d
    130.10.0.4/32 --dport 80 -j ACCEPT
 - sudo iptables -A FORWARD -p tcp -s 40.40.41.0/24 -d
    130.10.0.4/32 --dport 80 -j ACCEPT
 - sudo iptables -A FORWARD -p tcp -s 130.10.0.4/32 -d
    0.0.0.0/0 -j ACCEPT
 - sudo iptables -A FORWARD -p udp -s 130.10.0.4/32 -d
    0.0.0.0/0 -j ACCEPT
```

Per collegare una vnf al file di configurazione bisogna inserire il nome del file nella sezione dedicata alla vdu:

Listing 6.8. Sezione descrittore per la configurazione

```
   vdu:
 id: example
 int-cpd:
  ...
 name: example_vnf
 sw-image-desc: ubuntu20.04
```

```
    cloud-init-file: cloud-config.txt
```

Una volta definiti i package con i file di configurazione al loro interno vanno compressi e caricati in OSM tramite i comandi :

Listing 6.9. Comandi per crezione package compresso e caricamento su OSM

```
      cd "cartella del package"
  tar -cvzf vEPC_vnfd.tar.gz
  osm vnfd-create vEPC_vnfd.tar.gz
```

Oppure usando direttamente il comando :

Listing 6.10. Comando diretto per caricare il package su OSM

```
  osm vnfpkg-create cartellaPackage/nomePackage
```

il quale effettua un controllo di correttezza del package, effettua la compressione e l'on-boarding.

## 6.4 Creazione del descrittore del servizio

Effettuato l'on-boarding delle funzioni si può creare il package del servizio tramite il comando :

Listing 6.11. Comando per creazione di package generico di servizio di rete

```
      osm package-create --base-directory "cartella" --vendor
         Verefoo ns "nomePackage"
```

Se si usa un nome proprio per il servizio che non corrisponde a nessuna delle funzioni create precedentemente allora esso sarà inizializzato con una funzione fittizia. Oppure dando un nome già presente tra le funzioni, la scelta sarà presente nel descrittore. Entrambe le scelte sono possibili ma si consiglia la prima perchè il grafo è comunque composto da multiple funzioni diverse e non da una sola tipologia.

Creato il package va definita la topologia, definendo le funzioni presenti nel servizio , i link virtuali che servono per connettere le funzioni tra di loro e come le funzioni vengono connesse da questi link. Un esempio di descrittore del servizio è il seguente:

Listing 6.12. esempio di descrittore di servizio di rete

```
  nsd:
nsd:
- description: Descrizione del servizio
  df:
  - id: Id del deployment flavor
  #Da qui comincia la sezione delle VNF nel grafo
    vnf-profile:
    - id: vnf1 #ID legato all'istanza singola della funzione di
       cui grazie agli id possono esserne istanziate un
       qualsiasi numero
```

```
    #Sezione dove vengono specificate le connessione della
       macchina in esame, in questo caso due
     virtual-link-connectivity:
     - constituent-cpd-id:
       - constituent-base-element-id: vnf1 # id della funzione
         constituent-cpd-id: vnf-mgmt-ext # nome del connection
            point definito nel descrittore della funzione
       virtual-link-profile-id: Link # nome de link
     - constituent-cpd-id:
       - constituent-base-element-id: vnf1
         constituent-cpd-id: vnf-data-ext
       virtual-link-profile-id: Virt
     vnfd-id: hackfest_cloudinit-vnf # nome della funzione a
        cui fa riferimento questa istanza
#Dati identificativi del servizio
id: hackfest_cloudinit-ns
name: hackfest_cloudinit-ns
version: 1.0
#In questa sezione vanno inseriti TUTTI i link virtuali usati
   nel grafo
virtual-link-desc:
- id: Link
  mgmt-network: true
- id: Virt
#Nella sezione sottostante vanno specificati i nomi delle
   funzioni presenti
vnfd-id:
- hackfest_cloudinit-vnf
```

In base a ciò che è presente nel grafo precedente quello che va fatto è riempire le sezioni di vnfd-id per le funzioni, virtual-link-desc per i link e vnf-profile per le funzioni realmente presenti.Basandosi sul seguente template:

Listing 6.13. Template della singola VNF all'interno del nsd

```
id: "id dell'istanza"
vnfd-id: "nome del package"
    virtual-link-connectivity:
    - constituent-cpd-id:
      - constituent-base-element-id: "id istanza"
        constituent-cpd-id: "nome cp esterno"
      virtual-link-profile-id: "nome del link"
```

e replicandolo per tutte le funzioni e modificando i parametri in maniera opportuna , andando ad aggiungere connessione alle vnf quando necessario in base alle connessioni in cui essa è coinvolta, ottenendo infine un grafo completo. Come ultimo passo per assegnare un indirizzo ip alle singole funzioni esso va inserito nella sezione consituent-cpd-id specificando l'indirizzo per la singola interfaccia:

Listing 6.14. Sezione per specificare i collegamenti di una vnf con le altre e assegnare un indirizzo ip

```
constituent-cpd-id:
     - constituent-base-element-id: "id istanza"
       constituent-cpd-id: "nome cp esterno"
       ip-address: "indirizzo dell'interfaccia"
```

Scrivendo l'indirizzo in tutte le connessioni presenti nella sezione virtual-link-desc.

## 6.5 Esempio creazione di grafo generico su OSM

In questa sezione andremo ad effettuare la creazione di un servizio di rete di esempio, composto da funzioni ipotetiche.Per poter creare e inserire in OpenSource MANO un servizio di rete questo va realizzato tramite descrittore da inserire in un package e la stessa operazione deve essere svolta per tutte le funzioni che compongono il servizio. Vogliamo creare un servizio composto da due funzioni (istanza dello stesso package) monolitiche, con un solo componente, connesse tra di loro e caratterizzate da una configurazione generica. Avendo già preparato una macchina con un'istanza di OSM e un'altra, o la medesima ,con una versione del client di OSM per poter comunicare con la macchina remota si può passare alla definizione di funzioni e servizio.
Partiamo creando il package delle funzioni attraverso il comando :

Listing 6.15. Comando per creazione di package vnf generico

```
osm package-create --base-directory ~/test --image testimg
--vcpu 1 --memory 4096 --storage 10 --interfaces 1 --vendor
Verefoo vnf Test
```

Eseguito il comando, nel path specificato sarà presente la cartella Test_vnf con i seguenti file al suo interno:

- Files (cartella)
- Licenses (cartella)
- Scripts (cartella)
- checksums.txt
- README.md
- Test_vnfd.yaml

Di nostro interesse sono il file descrittore ovvero Test_vnfd.yaml e il file cloud-config.txt presente al path Scripts/cloud_init. La struttura del file descrittore della funzione appena creato è la seguente:

Listing 6.16. Descrittore Vnf generico

```
  vnfd:
id: vTest_vnfd
product-name: vTest_vnfd
description: Generated by OSM package generator
provider: OSM
version: '1.0'
mgmt-cp: vnf-cp0-ext
virtual-storage-desc:
- id: vTest_vnfd-VM-storage
  size-of-storage: 10
virtual-compute-desc:
- id: vTest_vnfd-VM-compute
  virtual-cpu:
    num-virtual-cpu: 1
  virtual-memory:
    size: 2048
sw-image-desc:
- id: "testimg"
  name: "testimg"
  image: "testimg"
df:
- id: default-df
  instantiation-level:
  - id: default-instantiation-level
    vdu-level:
    - vdu-id: vTest_vnfd-VM
      number-of-instances: 1
  vdu-profile:
  - id: vTest_vnfd-VM
    min-number-of-instances: 1
    max-number-of-instances: 1
  # At least one VDU need to be specified
  # Additional VDUs can be created by copying the
  # VDU descriptor below
vdu:
- id: vTest_vnfd-VM
  name: vTest_vnfd-VM
  description: vTest_vnfd-VM
  sw-image-desc: "test"
  virtual-storage-desc:
  - vTest_vnfd-VM-storage
  virtual-compute-desc: vTest_vnfd-VM-compute
  int-cpd:
  - id: eth0-int
    virtual-network-interface-requirement:
    - name: eth0
```

```
      virtual-interface:
        type: PARAVIRT
  ext-cpd:
  - id: vnf-cp0-ext
    int-cpd:
      vdu-id: vTest_vnfd-VM
      cpd: eth0-int
```

Le sezioni principali del file sono:

- Dati identificativi della funzione
- Risorse virtuali utilizzate
- Composizione della funzione in componenti
- Descrizione della/e vdu che compongono la funzioni
- Punto di connessione con l'esterno

All'interno del descittore va specificato il nome del file di configurazione associato alla vnf (in particolare al vnfc, componente della vnf) nella sezione della vdu e per facilitare la successiva definizione del grafo è consigliato andare ad assegnare un nome specifico identificativo (dal punto di vista umano) al connection point esterno della funzione.Vengono riportate di seguito le sole modifiche al file precendemente mostrato:

```
      vnfd:
      mgmt-cp: vnf-TEST-ext
      ...
      vdu:
      - ...
        cloud-init-file: cloud-config.txt
      ext-cpd:
      - id: vnf-TEST-ext
      ...
```

Nel file di configurazione andiamo ad inserire dei comandi per specificare la vm che verrà creata.Ad esempio:

Listing 6.17. Esempio di file di configurazione generico

```
runcmd:
  - comando esempio 1
  - comando esempio 2
```

Definiti in modo opportuno i file può essere effettuato l'on-boarding del package tramite il comando:

Listing 6.18. On-boarding package vnf esempio

```
    osm nfpkg-create ~/test/Test_vfn
```

Il package viene verificato, validato e caricato in OSM, all'interno della sezione package della GUI si può controllare la presenza del package appena caricato.

Una volta completata la definizione delle funzioni va definito il servizio.Creiamo un package base del servizio tramite il comando:

Listing 6.19. Comando per la creazione di un package generico di servizio di rete

```
osm package-create --base-directory ~/test --vendor Verefoo
   ns Test
```

Avendo usato come nome del servizio il nome della funzione esso verrà preparato con la funzione Test come componente.Il servizio appena creato è una versione base e va quindi modificato in modo da essere composto da due funzioni connesse tra di loro come da obiettivo. Eseguito il comando, nella cartella specificata sarà presente la cartella Test_ns che conterrà i seguenti file:

- Files (cartella)
- Licenses (cartella)
- Scripts (cartella)
- checksums.txt
- README.md
- Test_nsd.yaml

Il file descrittore appena creato risulta essere nella seguente configurazione:

Listing 6.20. Package del servizio di rete di esempio

```
  nsd:
nsd:
- id: Test_nsd
  name: Test_nsd
  designer: Verefoo
  description: Generated by OSM package generator
  version: '1.0'
  vnfd-id:
  - Test_vnfd
  df:
  - id: default-df
    vnf-profile:
    - id: "1"
      vnfd-id: Test_vnfd
      virtual-link-connectivity:
      - virtual-link-profile-id: Test_nsd_vld0
        constituent-cpd-id:
        - constituent-base-element-id: "1"
          constituent-cpd-id: vnf-cp0-ext
```

```
  virtual-link-desc:
  - id: Test_nsd_vld0
    mgmt-network: true
```

Il prossimo passo è modificare il file descrittore aggiungendo una funzione nella sezione dei collegamenti e modificare il nome del link eventualmente per una maggiore comprensione in caso di grafi complessi. Come detto nella sezione precedente basandosi sul componente:

Listing 6.21. Componente da aggiungere per la definizione del servizio di rete

```
  df:
  - id: default-df
    vnf-profile:
    - id: "1"
      vnfd-id: Test_vnfd
      virtual-link-connectivity:
      - virtual-link-profile-id: Test_nsd_vld0
        constituent-cpd-id:
        - constituent-base-element-id: "1"
          constituent-cpd-id: vnf-TEST-ext
```

Si replica la porzione id della sezione df in base a quante funzioni sono presenti nel servizio (anche se sono derivate dallo stesso package, da qui la presenza dall'attibuto id) modificando la sezione virtual-link-connectivity specificando gli identificativi opportuni per le due funzioni presenti nel grafo.Il file descrittore così modificato risulterà essere:

Listing 6.22. Descrittore finale del servizio di rete di esempio

```
  nsd:
nsd:
- id: Test_nsd
  name: Test_nsd
  designer: OSM
  description: Generated by OSM package generator
  version: '1.0'
  vnfd-id:
  - Test_vnfd
  df:
  - id: default-df
    vnf-profile:
    - id: test_1
      vnfd-id: Test_vnfd
      virtual-link-connectivity:
      - virtual-link-profile-id: vlink_1
        constituent-cpd-id:
        - constituent-base-element-id: test_1
          constituent-cpd-id: vnf-TEST-ext
    - id: test_2
```

```
    vnfd-id: Test_vnfd
    virtual-link-connectivity:
    - virtual-link-profile-id: vlink_1
      constituent-cpd-id:
      - constituent-base-element-id: test_2
        constituent-cpd-id: vnf-TEST-ext
virtual-link-desc:
- id: vlink_1
  mgmt-network: true
```

Completata la definizione del descrittore l'intero package può essere caricato tramite il comando:

Listing 6.23. Comando per l'on-boarding del servizio di rete

```
osm nspkg-create ~/test/Test_ns
```

All'interno della GUI si può verificare il caricamento del package appena definito, selezionandolo e andando nella sezione grafo si può osservare un'anteprima della sua topologia.

## 6.6 Creazione di grafo prodotto da verefoo su OSM

In questa parte della tesi si andrà ad eseguire la creazione di un servizio di rete usando come base un grafo fornito come risultato di un'esecuzione di verefoo.Il grafo, che rappresenta la topologia in figura utilizzato è il seguente:

```
<NFV xsi:noNamespaceSchemaLocation="./xsd/nfvSchema.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<graphs>
    <graph id="0">
        <node name="130.10.0.1" functional_type="WEBSERVER">
            <neighbour name="1.0.0.1"/>
            <configuration name="httpserver1" description="e1">
                <webserver>
                    <name>130.10.0.1</name>
                </webserver>
            </configuration>
        </node>
        <node name="130.10.0.2" functional_type="WEBSERVER">
            <neighbour name="1.0.0.2"/>
            <configuration name="httpserver2" description="e2">
                <webserver>
                    <name>130.10.0.2</name>
                </webserver>
            </configuration>
        </node>
```

```
<node name="130.10.0.3" functional_type="WEBSERVER">
    <neighbour name="1.0.0.3"/>
    <configuration name="httpserver3" description="e3">
        <webserver>
            <name>130.10.0.3</name>
        </webserver>
    </configuration>
</node>
<node name="1.0.0.1" functional_type="FORWARDER">
    <neighbour name="130.10.0.1"/>
    <neighbour name="130.10.0.4"/>
    <configuration name="ForwardConf">
        <forwarder>
            <name>Forwarder</name>
        </forwarder>
    </configuration>
</node>
<node name="1.0.0.2" functional_type="FORWARDER">
    <neighbour name="130.10.0.2"/>
    <neighbour name="130.10.0.4"/>
    <configuration name="ForwardConf">
        <forwarder>
            <name>Forwarder</name>
        </forwarder>
    </configuration>
</node>
<node name="1.0.0.3" functional_type="FORWARDER">
    <neighbour name="130.10.0.3"/>
    <neighbour name="130.10.0.4"/>
    <configuration name="ForwardConf">
        <forwarder>
            <name>Forwarder</name>
        </forwarder>
    </configuration>
</node>
<node name="130.10.0.4" functional_type="LOADBALANCER">
    <neighbour name="1.0.0.1"/>
    <neighbour name="1.0.0.2"/>
    <neighbour name="1.0.0.3"/>
    <neighbour name="1.0.0.4"/>
    <configuration name="loadbalancer"
       description="s9">
        <loadbalancer>
            <pool>130.10.0.1</pool>
            <pool>130.10.0.2</pool>
            <pool>130.10.0.3</pool>
        </loadbalancer>
    </configuration>
```

```
</node>
<node name="1.0.0.4" functional_type="FIREWALL">
    <neighbour name="130.10.0.4"/>
    <neighbour name="33.33.33.2"/>
    <configuration name="AutoConf" description="1">
       <!-- description in firewalls was added
       manually, it should be output of VEREFOO-->
        <firewall defaultAction="DENY">
            <elements>
                <action>ALLOW</action>
                <source>220.124.30.1</source>
                <destination>130.10.0.4</destination>
                <protocol>TCP</protocol>
                <src_port>*</src_port>
                <dst_port>80</dst_port>
            </elements>
            <elements>
                <action>ALLOW</action>
                <source>40.40.41.-1</source>
                <destination>130.10.0.4</destination>
                <protocol>TCP</protocol>
                <src_port>*</src_port>
                <dst_port>80</dst_port>
            </elements>
            <elements>
                <action>ALLOW</action>
                <source>130.10.0.4</source>
                <destination>-1.-1.-1.-1</destination>
                <protocol>ANY</protocol>
                <src_port>*</src_port>
                <dst_port>*</dst_port>
            </elements>
        </firewall>
    </configuration>
</node>
<node name="33.33.33.2" functional_type="FORWARDER">
    <neighbour name="1.0.0.4"/>
    <neighbour name="1.0.0.5"/>
    <neighbour name="1.0.0.6"/>
    <neighbour name="1.0.0.7"/>
    <configuration name="ForwardConf">
        <forwarder>
            <name>Forwarder</name>
        </forwarder>
    </configuration>
</node>
<node name="1.0.0.5" functional_type="FORWARDER">
    <neighbour name="33.33.33.2"/>
```

```
    <neighbour name="40.40.41.-1"/>
    <configuration name="ForwardConf">
        <forwarder>
            <name>Forwarder</name>
        </forwarder>
    </configuration>
</node>
<node name="1.0.0.6" functional_type="FIREWALL">
    <neighbour name="33.33.33.2"/>
    <neighbour name="40.40.42.-1"/>
    <configuration name="AutoConf" description="2">
        <firewall defaultAction="DENY">
            <elements>
                <action>ALLOW</action>
                <source>40.40.42.-1</source>
                <destination>40.40.41.-1</destination>
                <protocol>ANY</protocol>
                <src_port>*</src_port>
                <dst_port>*</dst_port>
            </elements>
            <elements>
                <action>ALLOW</action>
                <source>88.80.84.-1</source>
                <destination>40.40.42.-1</destination>
                <protocol>ANY</protocol>
                <src_port>*</src_port>
                <dst_port>*</dst_port>
            </elements>
        </firewall>
    </configuration>
</node>
<node name="40.40.41.-1" functional_type="WEBCLIENT">
    <neighbour name="1.0.0.5"/>
    <configuration name="officeA" description="e4">
        <webclient nameWebServer="130.10.0.1"/>
    </configuration>
</node>
<node name="40.40.42.-1" functional_type="WEBCLIENT">
    <neighbour name="1.0.0.6"/>
    <configuration name="businessofficeA"
       description="e5">
        <webclient nameWebServer="130.10.0.1"/>
    </configuration>
</node>
<node name="1.0.0.7" functional_type="FORWARDER">
    <neighbour name="33.33.33.2"/>
    <neighbour name="33.33.33.3"/>
    <configuration name="ForwardConf">
```

```
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="33.33.33.3" functional_type="FORWARDER">
        <neighbour name="1.0.0.7"/>
        <neighbour name="1.0.0.8"/>
        <neighbour name="1.0.0.9"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="1.0.0.8" functional_type="FORWARDER">
        <neighbour name="33.33.33.3"/>
        <neighbour name="88.80.84.-1"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="88.80.84.-1" functional_type="WEBCLIENT">
        <neighbour name="1.0.0.8"/>
        <configuration name="companyB" description="e6">
            <webclient nameWebServer="130.10.0.1"/>
        </configuration>
    </node>
    <node name="1.0.0.9" functional_type="FORWARDER">
        <neighbour name="33.33.33.3"/>
        <neighbour name="220.124.30.1"/>
        <configuration name="ForwardConf">
            <forwarder>
                <name>Forwarder</name>
            </forwarder>
        </configuration>
    </node>
    <node name="220.124.30.1" functional_type="NAT">
        <neighbour name="1.0.0.9"/>
        <neighbour name="1.0.0.10"/>
        <neighbour name="1.0.0.11"/>
        <configuration name="nat" description="s12">
            <nat>
                <source>192.168.3.-1</source>
                <source>192.168.2.-1</source>
            </nat>
```

```
            </configuration>
        </node>
        <node name="1.0.0.10" functional_type="FORWARDER">
            <neighbour name="220.124.30.1"/>
            <neighbour name="192.168.3.-1"/>
            <configuration name="ForwardConf">
                <forwarder>
                    <name>Forwarder</name>
                </forwarder>
            </configuration>
        </node>
        <node name="1.0.0.11" functional_type="FIREWALL">
            <neighbour name="220.124.30.1"/>
            <neighbour name="192.168.2.-1"/>
            <configuration name="AutoConf" description="3">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>-1.-1.-1.-1</source>
                        <destination>192.168.-1.-1</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="192.168.3.-1" functional_type="WEBCLIENT">
            <neighbour name="1.0.0.10"/>
            <configuration name="officeC" description="e7">
                <webclient nameWebServer="130.10.0.1"/>
            </configuration>
        </node>
        <node name="192.168.2.-1" functional_type="WEBCLIENT">
            <neighbour name="1.0.0.11"/>
            <configuration name="businessofficeC"
               description="e8">
                <webclient nameWebServer="130.10.0.1"/>
            </configuration>
        </node>
    </graph>
  </graphs>
</NFV>
```

Il grafo, che descrive la topologia rappresentata in figura 6.1, è composto da web server collegati ognuno ad un forwarder , rappresentati in questo caso da un unico forwarder (s9),il tutto protetto da un firewall (f1).All'esterno della rete dei server è presente un forwarder a cui sono collegate tre reti principali: la prima composta

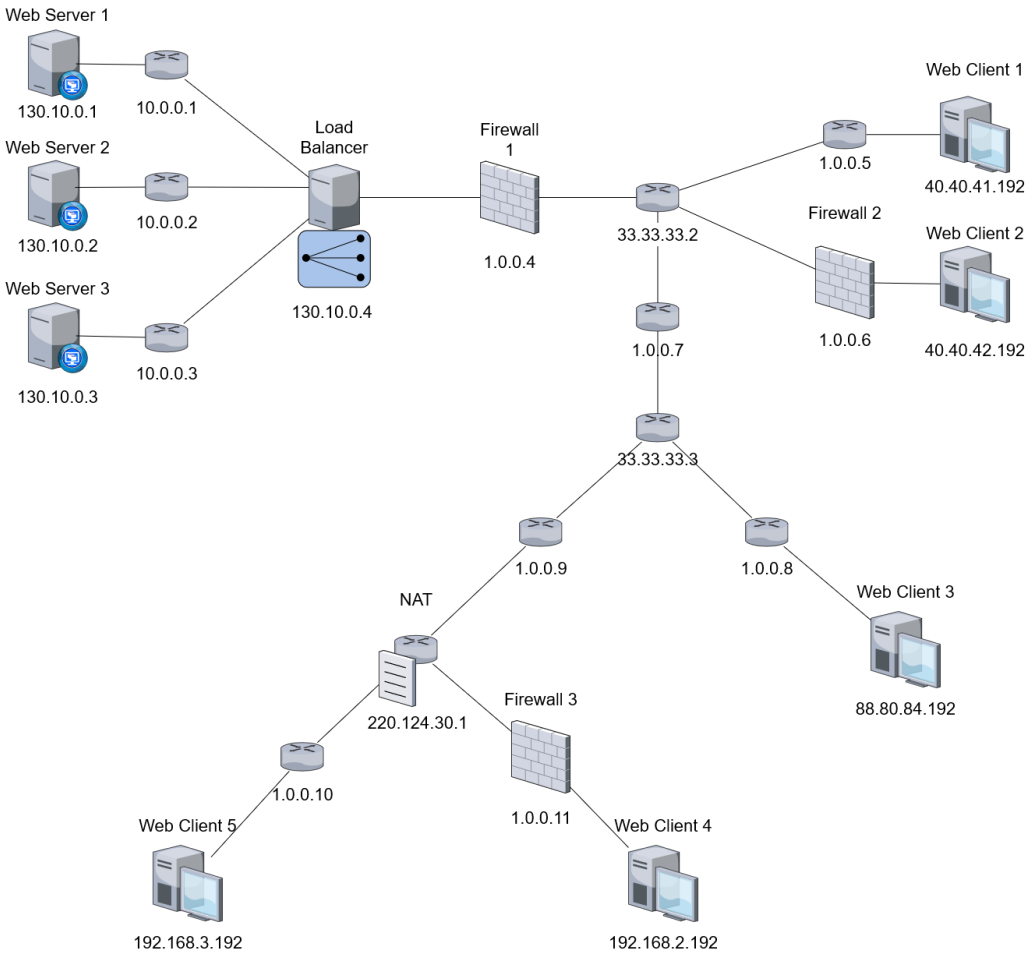


Figura 6.1. Topologia di Verefoo

solo da un forwarder collegato ad un client;la seconda presenta un firewall a cui è collegato un altro client; la terza , che è la più complessa, presenta un forwarder collegato ad un altro forwarder che collega due reti, una con un forwarder ed un client, un 'altra con NAT e forwarder e client ad un 'estremità e firewall e client ad un'altra estremità. Per poter effettuare la realizzazione del grafo come primo passo è necessario individuare le funzioni presenti all'interno del grafo di interesse. Le diverse tipologie che possono essere individuate sono le seguenti:

- End-points che possono essere divisi in webclient e webserver, possono essere raggruppati nella stessa categoria in quanto la configurazione è simile
- NAT
- Firewall , nella lista viene inserito un solo firewall , ma devono essere considerate tre istanze diverse dato che la configurazione dei firewall è diversa
- Forwarder, si può considerare una solo entità con la stessa configurazione

Identificate le funzioni del grafo si può procedere alla creazione dei package delle funzioni.

## 6.7 Creazione dei package delle funzioni

Per la creazione dei package partiamo dalla prima categoria di funzioni presentata ovvero gli endpoints.Tramite il comando :

```
osm package-create ~/graph --image "endpointimg" --vcpu 1
   --memory 4096 -- storage 25 --interfaces 2 --vendor
   verefoo vnf vEND
```

andiamo a definire il package con all'interno il descrittore yaml che modificando opportunamente risulterà essere:

```
  vnfd:
id: vEND_vnfd
product-name: vEND_vnfd
description: Generated by OSM package generator
provider: OSM
version: '1.0'
mgmt-cp: vnf-ENDpoint-ext
virtual-storage-desc:
- id: vEND_vnfd-VM-storage
  size-of-storage: 25
virtual-compute-desc:
- id: vEND_vnfd-VM-compute
  virtual-cpu:
    num-virtual-cpu: 1
  virtual-memory:
    size: 4096
sw-image-desc:
- id: "endPoint.qcow2"
  name: "endPoint.qcow2"
  image: "endPoint.qcow2"
df:
- id: default-df
  instantiation-level:
  - id: default-instantiation-level
    vdu-level:
    - vdu-id: vEND_vnfd-VM
      number-of-instances: 1
  vdu-profile:
  - id: vEND_vnfd-VM
    min-number-of-instances: 1
    max-number-of-instances: 1
  # At least one VDU need to be specified
  # Additional VDUs can be created by copying the
  # VDU descriptor below
vdu:
- id: vEND_vnfd-VM
```

```
    name: vEND_vnfd-VM
    description: vEND_vnfd-VM
    sw-image-desc: "endPoint.qcow2"
    virtual-storage-desc:
    - vEND_vnfd-VM-storage
    virtual-compute-desc: vEND_vnfd-VM-compute
    int-cpd:
    - id: eth0-int
      virtual-network-interface-requirement:
      - name: eth0
        virtual-interface:
          type: PARAVIRT
    cloud-init-file: cloud-config.txt
  ext-cpd:
  - id: vnf-ENDpoint-ext
    int-cpd:
      vdu-id: vEND_vnfd-VM
      cpd: eth0-int
```

Per quanto riguarda la configurazione da inserire nel file cloud-config.txt useremo la seguente lista di comandi:

```
    #cloud-config
runcmd:
 - route add -net 20.0.0.0 netmask 255.255.255.0 gw 20.0.1.1
 - route add -net 20.0.3.0 netmask 255.255.255.0 gw 33.33.33.2
 - route add -net 40.40.42.0 netmask 255.255.255.0 gw 20.0.2.2
 - route add -net 88.80.84.0 netmask 255.255.255.0 gw 33.33.33.3
 - route add -net 130.10.0.0 netmask 255.255.255.0 gw 20.0.1.1
 - route add -net 192.168.3.0 netmask 255.255.255.0 gw 33.33.33.3
 - route add -net 192.168.2.0 netmask 255.255.255.0 gw 33.33.33.3
 - route add -net 220.124.30.0 netmask 255.255.255.0 gw 33.33.33.3
```

che permettono di definire una semplice lista di route che verranno aggiunte ai dispositivi.

La seconda funzione che deve essere definita è il NAT, il cui package viene generato usando il comando :

```
    osm package-create --base-directory ~/graph --image "natimg"
       --vcpu 1 --memory 4096 --storage 10 --interfaces 2
       --vendor verefoo vnf vNAT
```

modificando il package definito con il comando precedente il risultato sarà:

```
    vnfd:
  id: vNAT_vnf_vnfd
  product-name: vNAT_vnf_vnfd
  description: Generated by OSM package generator
  provider: OSM
```

```
version: '1.0'
mgmt-cp: vnf-nat-ext
virtual-storage-desc:
- id: vNAT_vnf_vnfd-VM-storage
  size-of-storage: 10
virtual-compute-desc:
- id: vNAT_vnf_vnfd-VM-compute
  virtual-cpu:
    num-virtual-cpu: 1
  virtual-memory:
    size: 4096
sw-image-desc:
- id: "nat-base"
  name: "nat-base"
  image: "nat-base"
df:
- id: default-df
  instantiation-level:
  - id: default-instantiation-level
    vdu-level:
    - vdu-id: vNAT_vnf_vnfd-VM
      number-of-instances: 1
  vdu-profile:
  - id: vNAT_vnf_vnfd-VM
    min-number-of-instances: 1
    max-number-of-instances: 1
  # At least one VDU need to be specified
  # Additional VDUs can be created by copying the
  # VDU descriptor below
vdu:
- id: vNAT_vnf_vnfd-VM
  name: vNAT_vnf_vnfd-VM
  description: vNAT_vnf_vnfd-VM
  sw-image-desc: "nat-base"
  virtual-storage-desc:
  - vNAT_vnf_vnfd-VM-storage
  virtual-compute-desc: vNAT_vnf_vnfd-VM-compute
  int-cpd:
  - id: eth0-int
    virtual-network-interface-requirement:
    - name: eth0
      virtual-interface:
        type: PARAVIRT
  cloud-init-file: cloud-config.txt
ext-cpd:
- id: vnf-nat-ext
  int-cpd:
    vdu-id: vNAT_vnf_vnfd-VM
```

```
      cpd: eth0-int
```

mentre la configurazione per il NAT inserita nel file cloud-config.txt è la seguente:

```
    #cloud-config
runcmd:
 - sudo iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -j SNAT
    --to-source 220.124.30.1
 - sudo iptables -t nat -A POSTROUTING -s 192.168.2.0/24 -j SNAT
    --to-source 220.124.30.1
```

Adesso definiamo le funzioni firewall per semplicità riportiamo il file descrittore di un solo firewall generico , sostituendo a questa configurazione i valori identificativi dei firewall (1,2,3) al posto della n si ottiene la struttura del firewall specifico. Il comando tramite il quale possiamo definire il package generico è il seguente:

```
    osm package-create --base-directory ~/graph --image
       firewallimg --vcpu 1 --memory 4096 --storage 10
       --interfaces 2 --vendor verefoo vnf vFW_n
```

i file descrittori,modificando la n, saranno i seguenti:

```
    vnfd:
  id: vFW_n_vnfd
  product-name: vFW_n_vnfd
  description: Generated by OSM package generator
  provider: OSM
  version: '1.0'
  mgmt-cp: vnf-FWn-ext
  virtual-storage-desc:
  - id: vFW_n_vnfd-VM-storage
    size-of-storage: 10
  virtual-compute-desc:
  - id: vFW_n_vnfd-VM-compute
    virtual-cpu:
      num-virtual-cpu: 1
    virtual-memory:
      size: 2048
  sw-image-desc:
  - id: "fw-base"
    name: "fw-base"
    image: "fw-base"
  df:
  - id: default-df
    instantiation-level:
    - id: default-instantiation-level
      vdu-level:
      - vdu-id: vFW_n_vnfd-VM
```

```
        number-of-instances: 1
    vdu-profile:
    - id: vFW_n_vnfd-VM
      min-number-of-instances: 1
      max-number-of-instances: 1
    # At least one VDU need to be specified
    # Additional VDUs can be created by copying the
    # VDU descriptor below
  vdu:
  - id: vFW_n_vnfd-VM
    name: vFW_n_vnfd-VM
    description: vFW_n_vnfd-VM
    sw-image-desc: "fw-base"
    virtual-storage-desc:
    - vFW_n_vnfd-VM-storage
    virtual-compute-desc: vFW_n_vnfd-VM-compute
    int-cpd:
    - id: eth0-int
      virtual-network-interface-requirement:
      - name: eth0
        virtual-interface:
          type: PARAVIRT
    cloud-init-file: cloud-config.txt
  ext-cpd:
  - id: vnf-FWn-ext
    int-cpd:
      vdu-id: vFW_n_vnfd-VM
      cpd: eth0-int
```

Di seguito vengono presentate le configurazioni dei tre firewall. Configurazione del firewall 1:

```
    #cloud-config
runcmd:
 - sudo iptables -F
 - sudo iptables -P INPUT DROP
 - sudo iptables -P FORWARD DROP
 - sudo iptables -P OUTPUT DROP
 - sudo iptables -A FORWARD -p tcp -s 220.124.30.1/32 -d
    130.10.0.4/32 --dport 80 -j ACCEPT
 - sudo iptables -A FORWARD -p tcp -s 40.40.41.0/24 -d
    130.10.0.4/32 --dport 80 -j ACCEPT
 - sudo iptables -A FORWARD -p tcp -s 130.10.0.4/32 -d 0.0.0.0/0
    -j ACCEPT
 - sudo iptables -A FORWARD -p udp -s 130.10.0.4/32 -d 0.0.0.0/0
    -j ACCEPT
```

La seguente è la configurazione del firewall 2:

```
    #cloud-config
runcmd:
 - sudo iptables -F
 - sudo iptables -P INPUT DROP
 - sudo iptables -P FORWARD DROP
 - sudo iptables -P OUTPUT DROP
 - sudo iptables -A FORWARD -p tcp -s 40.40.42.0/24 -d
    40.40.41.0/24 -j ACCEPT
 - sudo iptables -A FORWARD -p udp -s 40.40.42.0/24 -d
    40.40.41.0/24 -j ACCEPT
 - sudo iptables -A FORWARD -p tcp -s 88.80.84.0/24 -d
    40.40.42.0/24 -j ACCEPT
 - sudo iptables -A FORWARD -p udp -s 88.80.84.0/24 -d
    40.40.42.0/24 -j ACCEPT
```

Per ultima riportiamo la configurazione del firewall 3 :

```
    #cloud-config
runcmd:
 - sudo iptables -F
 - sudo iptables -P INPUT DROP
 - sudo iptables -P FORWARD DROP
 - sudo iptables -P OUTPUT DROP
 - sudo iptables -A FORWARD -p tcp -s 0.0.0.0/0 -d 192.168.0.0/16
    -j ACCEPT
 - sudo iptables -A FORWARD -p udp -s 0.0.0.0/0 -d 192.168.0.0/16
    -j ACCEPT
```

Infine andiamo a definire il package e il descrittore per l'ultima tipologia di funzioni, i forwarder. Il comando per creare il package è il seguente:

```
    osm package-create --base-directory ~/graph --image
       forwarderimg --vcpu 1 --memory 2048 --storage 10
       --interfaces 2 --vendor verefoo vnf vFORWARDER
```

il file yaml dei forwarder modificato è il seguente:

```
    vnfd:
  id: vFORWARD_vnfd
  product-name: vFORWARD_vnfd
  description: Generated by OSM package generator
  provider: OSM
  version: '1.0'
  mgmt-cp: vnf-Forward-ext
  virtual-storage-desc:
  - id: vFORWARD_vnfd-VM-storage
    size-of-storage: 10
  virtual-compute-desc:
  - id: vFORWARD_vnfd-VM-compute
```

```
    virtual-cpu:
      num-virtual-cpu: 1
    virtual-memory:
      size: 2048
  sw-image-desc:
  - id: "forward-base"
    name: "forward-base"
    image: "forward-base"
  df:
  - id: default-df
    instantiation-level:
    - id: default-instantiation-level
      vdu-level:
      - vdu-id: vFORWARD_vnfd-VM
        number-of-instances: 1
    vdu-profile:
    - id: vFORWARD_vnfd-VM
      min-number-of-instances: 1
      max-number-of-instances: 1
    # At least one VDU need to be specified
    # Additional VDUs can be created by copying the
    # VDU descriptor below
  vdu:
  - id: vFORWARD_vnfd-VM
    name: vFORWARD_vnfd-VM
    description: vFORWARD_vnfd-VM
    sw-image-desc: "forward-base"
    virtual-storage-desc:
    - vFORWARD_vnfd-VM-storage
    virtual-compute-desc: vFORWARD_vnfd-VM-compute
    int-cpd:
    - id: eth0-int
      virtual-network-interface-requirement:
      - name: eth0
        virtual-interface:
          type: PARAVIRT
    cloud-init-file: cloud-config.txt
  ext-cpd:
  - id: vnf-Forward-ext
    int-cpd:
      vdu-id: vFORWARD_vnfd-VM
      cpd: eth0-int
```

La configurazione dei forwarder è la seguente:

```
    #cloud-config
runcmd:
 - sudo iptables -F
```

```
- sudo iptables -A INPUT -m conntrack --ctstate
   ESTABLISHED,RELATED -j ACCEPT
- sudo iptables -A INPUT -i lo -j ACCEPT
- sudo iptables -A INPUT -j ACCEPT
- sudo iptables -A FORWARD -j ACCEPT
```

Una volta definite tutte le funzioni ne possiamo effettuare l'onboarding tramite il comando opportuno.Adesso possiamo passare alla definzione del servizio di rete.

## 6.8 Creazione del servizio

Per la creazione del servizio partiamo dalla generazione del package con all'interno uno scheletro del servizio di rete. Questo può essere fatto eseguendo il comando:

```
osm package-create --base-directory ~/graph --vendor verefoo
   ns vGRAPH
```

Seguendo i passi mostrati nelle sezione precedente , si definiscono le funzioni presenti nel grafo, i link virtuali usati per collegare le funzioni e come i collegamenti tra funzioni vengono realizzati. Ultimate queste azioni il grafo finale sarà il seguente :

```
  nsd:
nsd:
  - description: Generated by OSM package generator
    designer: Verefoo
    df:
      - id: default-df
        vnf-profile:
          - id: vWEB_SERVER_1
            virtual-link-connectivity:
              - constituent-cpd-id:
                  - constituent-base-element-id: vWEB_SERVER_1
                    constituent-cpd-id: vnf-ENDpoint-ext
                    ip-address: 130.10.0.1
                virtual-link-profile-id: vGraph_nsd_vldS1
            vnfd-id: vEND_vnfd
          - id: vWEB_SERVER_2
            virtual-link-connectivity:
              - constituent-cpd-id:
                  - constituent-base-element-id: vWEB_SERVER_2
                    constituent-cpd-id: vnf-ENDpoint-ext
                    ip-address: 130.10.0.2
                virtual-link-profile-id: vGraph_nsd_vldS2
            vnfd-id: vEND_vnfd
          - id: vWEB_SERVER_3
            virtual-link-connectivity:
              - constituent-cpd-id:
```

```
        - constituent-base-element-id: vWEB_SERVER_3
          constituent-cpd-id: vnf-ENDpoint-ext
          ip-address: 130.10.0.3
      virtual-link-profile-id: vGraph_nsd_vldS3
  vnfd-id: vEND_vnfd
- id: vFORWARDER_1
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_1
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.1
      virtual-link-profile-id: vGraph_nsd_vldS1
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_1
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.1
      virtual-link-profile-id: vGraph_nsd_vldLB1
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_2
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_2
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.2
      virtual-link-profile-id: vGraph_nsd_vldS2
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_2
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.2
      virtual-link-profile-id: vGraph_nsd_vldLB2
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_3
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_3
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.3
      virtual-link-profile-id: vGraph_nsd_vldS3
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_3
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.3
      virtual-link-profile-id: vGraph_nsd_vldLB3
  vnfd-id: vFORWARD_vnfd
- id: vLB
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vLB
```

```
            constituent-cpd-id: vnf-LB-ext
            ip-address: 130.10.0.4
          virtual-link-profile-id: vGraph_nsd_vldLB1
        - constituent-cpd-id:
          - constituent-base-element-id: vLB
            constituent-cpd-id: vnf-LB-ext
            ip-address: 130.10.0.4
          virtual-link-profile-id: vGraph_nsd_vldLB2
        - constituent-cpd-id:
          - constituent-base-element-id: vLB
            constituent-cpd-id: vnf-LB-ext
            ip-address: 130.10.0.4
          virtual-link-profile-id: vGraph_nsd_vldLB3
        - constituent-cpd-id:
          - constituent-base-element-id: vLB
            constituent-cpd-id: vnf-LB-ext
            ip-address: 130.10.0.4
          virtual-link-profile-id: vGraph_nsd_vldLBFW
      vnfd-id: vLB_vnfd
    - id: vFIREWALL_1
      virtual-link-connectivity:
        - constituent-cpd-id:
          - constituent-base-element-id: vFIREWALL_1
            constituent-cpd-id: vnf-FW1-ext
            ip-address: 1.0.0.4
          virtual-link-profile-id: vGraph_nsd_vldLBFW
        - constituent-cpd-id:
          - constituent-base-element-id: vFIREWALL_1
            constituent-cpd-id: vnf-FW1-ext
            ip-address: 1.0.0.4
          virtual-link-profile-id: vGraph_nsd_vldFW
      vnfd-id: vFW_1_vnfd
    - id: vFORWARDER_4
      virtual-link-connectivity:
        - constituent-cpd-id:
          - constituent-base-element-id: vFORWARDER_4
            constituent-cpd-id: vnf-Forward-ext
            ip-address: 33.33.33.2
          virtual-link-profile-id: vGraph_nsd_vldFW
        - constituent-cpd-id:
          - constituent-base-element-id: vFORWARDER_4
            constituent-cpd-id: vnf-Forward-ext
            ip-address: 33.33.33.2
          virtual-link-profile-id: vGraph_nsd_vldFORW1
        - constituent-cpd-id:
          - constituent-base-element-id: vFORWARDER_4
            constituent-cpd-id: vnf-Forward-ext
            ip-address: 33.33.33.2
```

```
      virtual-link-profile-id: vGraph_nsd_vldFORW2
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_4
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 33.33.33.2
      virtual-link-profile-id: vGraph_nsd_vldFORW3
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_5
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_5
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.5
      virtual-link-profile-id: vGraph_nsd_vldFORW1
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_5
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.5
      virtual-link-profile-id: vGraph_nsd_vldC1
  vnfd-id: vFORWARD_vnfd
- id: vCLIENT_1
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vCLIENT_1
          constituent-cpd-id: vnf-ENDpoint-ext
          ip-address: 40.40.41.192
      virtual-link-profile-id: vGraph_nsd_vldC1
  vnfd-id: vEND_vnfd
- id: vFIREWALL_2
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFIREWALL_2
          constituent-cpd-id: vnf-FW2-ext
          ip-address: 1.0.0.6
      virtual-link-profile-id: vGraph_nsd_vldFORW2
    - constituent-cpd-id:
        - constituent-base-element-id: vFIREWALL_2
          constituent-cpd-id: vnf-FW2-ext
          ip-address: 1.0.0.6
      virtual-link-profile-id: vGraph_nsd_vldC2
  vnfd-id: vFW_2_vnfd
- id: vCLIENT_2
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vCLIENT_2
          constituent-cpd-id: vnf-ENDpoint-ext
          ip-address: 40.40.42.192
      virtual-link-profile-id: vGraph_nsd_vldC2
```

```
  vnfd-id: vEND_vnfd
- id: vFORWARDER_6
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_6
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.7
      virtual-link-profile-id: vGraph_nsd_vldFORW3
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_6
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.7
      virtual-link-profile-id: vGraph_nsd_vldFORW4
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_7
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_7
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 33.33.33.3
      virtual-link-profile-id: vGraph_nsd_vldFORW4
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_7
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 33.33.33.3
      virtual-link-profile-id: vGraph_nsd_vldFORW5
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_7
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 33.33.33.3
      virtual-link-profile-id: vGraph_nsd_vldFORW6
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_9
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_9
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.9
      virtual-link-profile-id: vGraph_nsd_vldFORW6
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_9
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.9
      virtual-link-profile-id: vGraph_nsd_vldC3
  vnfd-id: vFORWARD_vnfd
- id: vFORWARDER_8
  virtual-link-connectivity:
    - constituent-cpd-id:
```

```
        - constituent-base-element-id: vFORWARDER_8
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.8
      virtual-link-profile-id: vGraph_nsd_vldFORW5
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_8
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.8
      virtual-link-profile-id: vGraph_nsd_vldNAT1
  vnfd-id: vFORWARD_vnfd
- id: vCLIENT_3
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vCLIENT_3
          constituent-cpd-id: vnf-ENDpoint-ext
          ip-address: 88.80.84.192
      virtual-link-profile-id: vGraph_nsd_vldC3
  vnfd-id: vEND_vnfd
- id: vNAT
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vNAT
          constituent-cpd-id: vnf-nat-ext
          ip-address: 220.124.30.1
      virtual-link-profile-id: vGraph_nsd_vldNAT1
    - constituent-cpd-id:
        - constituent-base-element-id: vNAT
          constituent-cpd-id: vnf-nat-ext
          ip-address: 220.124.30.1
      virtual-link-profile-id: vGraph_nsd_vldNAT2
    - constituent-cpd-id:
        - constituent-base-element-id: vNAT
          constituent-cpd-id: vnf-nat-ext
          ip-address: 220.124.30.1
      virtual-link-profile-id: vGraph_nsd_vldFW3
  vnfd-id: vNAT_vnf_vnfd
- id: vFORWARDER_10
  virtual-link-connectivity:
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_10
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.10
      virtual-link-profile-id: vGraph_nsd_vldNAT2
    - constituent-cpd-id:
        - constituent-base-element-id: vFORWARDER_10
          constituent-cpd-id: vnf-Forward-ext
          ip-address: 1.0.0.10
      virtual-link-profile-id: vGraph_nsd_vldC5
```

```
        vnfd-id: vFORWARD_vnfd
      - id: vFIREWALL_3
        virtual-link-connectivity:
          - constituent-cpd-id:
              - constituent-base-element-id: vFIREWALL_3
                constituent-cpd-id: vnf-FW3-ext
                ip-address: 1.0.0.11
            virtual-link-profile-id: vGraph_nsd_vldFW3
          - constituent-cpd-id:
              - constituent-base-element-id: vFIREWALL_3
                constituent-cpd-id: vnf-FW3-ext
                ip-address: 1.0.0.11
            virtual-link-profile-id: vGraph_nsd_vldC4
        vnfd-id: vFW_3_vnfd
      - id: vCLIENT_4
        virtual-link-connectivity:
          - constituent-cpd-id:
              - constituent-base-element-id: vCLIENT_4
                constituent-cpd-id: vnf-ENDpoint-ext
                ip-address: 192.168.1.192
            virtual-link-profile-id: vGraph_nsd_vldC4
        vnfd-id: vEND_vnfd
      - id: vCLIENT_5
        virtual-link-connectivity:
          - constituent-cpd-id:
              - constituent-base-element-id: vCLIENT_5
                constituent-cpd-id: vnf-ENDpoint-ext
                ip-address: 192.168.3.192
            virtual-link-profile-id: vGraph_nsd_vldC5
        vnfd-id: vEND_vnfd
id: vGraph_nsd
name: vGraph_nsd
nsdOnboardingState: ONBOARDED
nsdOperationalState: ENABLED
nsdUsageState: NOT_IN_USE
version: '1.0'
virtual-link-desc:
  - id: vGraph_nsd_vldS1
    mgmt-network: true
  - id: vGraph_nsd_vldS2
    mgmt-network: true
  - id: vGraph_nsd_vldS3
    mgmt-network: true
  - id: vGraph_nsd_vldLB1
    mgmt-network: true
  - id: vGraph_nsd_vldLB2
    mgmt-network: true
  - id: vGraph_nsd_vldLB3
```

```
      mgmt-network: true
    - id: vGraph_nsd_vldLBFW
      mgmt-network: true
    - id: vGraph_nsd_vldFW
      mgmt-network: true
    - id: vGraph_nsd_vldFORW1
      mgmt-network: true
    - id: vGraph_nsd_vldFORW2
      mgmt-network: true
    - id: vGraph_nsd_vldC1
      mgmt-network: true
    - id: vGraph_nsd_vldC2
      mgmt-network: true
    - id: vGraph_nsd_vldFORW3
      mgmt-network: true
    - id: vGraph_nsd_vldFORW4
      mgmt-network: true
    - id: vGraph_nsd_vldFORW5
      mgmt-network: true
    - id: vGraph_nsd_vldFORW6
      mgmt-network: true
    - id: vGraph_nsd_vldC3
      mgmt-network: true
    - id: vGraph_nsd_vldNAT1
      mgmt-network: true
    - id: vGraph_nsd_vldNAT2
      mgmt-network: true
    - id: vGraph_nsd_vldFW3
      mgmt-network: true
    - id: vGraph_nsd_vldC4
      mgmt-network: true
    - id: vGraph_nsd_vldC5
      mgmt-network: true
  vnfd-id:
    - vEND_vnfd
    - vFORWARD_vnfd
    - vFW_1_vnfd
    - vFW_2_vnfd
    - vFW_3_vnfd
    - vLB_vnfd
    - vNAT_vnf_vnfd
```

Fatto questo si può effettuare l'on-boarding del package del servizio appena definito.Adesso si dipone di un grafo pronto per l'istanziazione.

# Capitolo 7

# Conclusioni

Questo lavoro di tesi ha identificato, tramite lo studio degli orchestratori, e realizzato una forma di interazione tra l'orchestratore nfv OpenSource MANO e il framework VEREFOO, verificando la possibilità di aggiungere un ulteriore grado di automatizzazione al funzionamento di OSM. Abbiamo presentato il modo con cui si può interagire con OSM per poter definire funzioni e servizi, mostrando passo per passo i procedimenti da svolgere. Infine si è mostrato come si presenta un servizio di rete modellato a partire da un grafo prodotto da verefoo. Va considerato però che uno dei problemi introdotti inizialmente non viene in realtà risolto. o almeno non completamente. Certamente la configurazione della sicurezza nel grafo viene calcolata in modo automatico, ma nella soluzione proposta la creazione delle funzioni e del servizio deve essere effettuata manualmente da un operatore, mantenendo la possibilità di errori. Un passo importante è comunque stato svolto nella possibilità di diminuire l'interazione umana nella definizione di un servizio che possa prevedere anche un certo livello di sicurezza, calcolato in modo opportuno da verefoo.

## 7.1 Lavori futuri

Il problema della soluzione proposta è , come già detto, che il tramite tra OSM e verefoo è un attore umano. Una possibile estensione di questo lavoro può consistere nell'esplorazione di un'interazione con l'orchestratore che non presenti la necessità di interazione manuale , diminuendo ulteriormente la possibile presenza di errori all'interno di un servizio di rete. Capire se è possibile un'interazione bidirezionale, quindi poter far utilizzare verefoo direttamente da OSM, il quale comunicherà egli stesso il grafo di cui dover definire le funzioni di sicurezza, o solo la loro configurazione. Successivamente l'ouput di verefoo va elaborato in modo opportuno per poter essere espresso tramite descrittori di funzioni e servizio che devono essere caricati su OSM. Il punto centrale sarà capire se è possibile definire un sistema di traduzione dell'output di verefoo in descrittori utilizzabili da OSM.

# Bibliografia


[1] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, vol. 56, no. 3, oct 2023. [Online]. Available: https://doi.org/10.1145/3616401

[2] ——, "A novel approach for security function graph configuration and deployment," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 457–463.

[3] D. Bringhenti, R. Sisto, and F. Valenza, "A novel abstraction for security configuration in virtual networks," *Computer Networks*, vol. 228, p. 109745, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623001901

[4] ——, "Security automation for multi-cluster orchestration in kubernetes," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 480–485.

[5] E. T. S. Institute, "Network functions virtualisation; an introduction, benefits, enablers, challenges and call for action," 2012. [Online]. Available: :http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[6] ——, "Network functions virtualisation; infrastructure overview," 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_nfv-inf001v010101p.pdf

[7] ——, "Network functions virtualisation;management and orchestration," 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[8] D. Bringhenti, R. Sisto, and F. Valenza, "A demonstration of verefoo: an automated framework for virtual firewall configuration," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 293–295.

[9] ——, "Automating vpn configuration in computer networks," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2024.

[10] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023.

[11] D. Bringhenti and F. Valenza, "Optimizing distributed firewall reconfiguration transients," *Computer Networks*, vol. 215, p. 109183, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912862200281X

[12] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2024.

[13] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, *Automatic and optimized firewall reconfiguration*, 2024. [Online]. Available: https://hdl.handle.net/11583/2985072

[14] O. E. U. A. G. (EUAG), "Osm scope, functionality, operation and integration guidelines," 2019. [Online]. Available: https://osm-download.etsi.org/ftp/Documentation/201902-osm-scope-white-paper/#!index.md

[15] OSM, "Osm data model," 2016. [Online]. Available: https://osm.etsi.org/wikipub/index.php/Release_0_Data_Model_Details

[16] "Tacker documentation." [Online]. Available: https://docs.openstack.org/tacker/latest/

[17] OSM, "Configuring ip addresses for vnfs." [Online]. Available: https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html#how-to-configure-ipv4-ipv6-dual-stack-addresses-statically-in-the-ns-descriptor

[18] ——, "Osm quickstart." [Online]. Available: https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html#installing-osm

[19] ——, "Open source mano's vnf on-boarding guide." [Online]. Available: https://osm.etsi.org/docs/vnf-onboarding-guidelines/